European Journal of Operational Research 201 (2010) 966–970

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Short Communication

A note on the single machine scheduling to minimize the number of tardy jobs with deadlines

Cheng He [a,b,*], Yixun Lin [a], Jinjiang Yuan [a]

[a] Department of Mathematics, Zhengzhou University, Zhengzhou, Henan 450052, People's Republic of China
[b] Institute of Science, Information Engineering University of PLA, Zhengzhou, Henan 450001, People's Republic of China

ARTICLE INFO

Article history:
Received 23 June 2008
Accepted 11 May 2009
Available online 18 May 2009

Keywords:
Scheduling
Number of tardy jobs
Deadline
Independent set

ABSTRACT

It is known that the single machine scheduling problem of minimizing the number of tardy jobs is polynomially solvable. However, it becomes NP-hard if each job has a deadline. Recently, Huo et al. solved some special cases by a backwards scheduling approach. In this note we present a dual approach—forwards greedy algorithms which may have better running time. For example, in the case that the due dates, deadlines, and processing times are agreeable, the running time of the backwards scheduling algorithm is $O(n^2)$, while that of the forwards algorithm is $O(n \log n)$.

1. Introduction

The single machine scheduling problem of minimizing the number of tardy jobs, denoted by $1\|\sum U_j$, is one of the classical problems that have polynomial-time algorithms. Some generalizations are extensively studied in the literature (see [1]). Among them, Lawler [5] showed that the problem with deadline constraint, namely $1|\bar{d}_j|\sum U_j$, is NP-hard. In a recent paper of Huo et al. [3], polynomial-time algorithms are presented for the following special cases:

(1) $d_i \leqslant d_j$ implies $\bar{d}_i \leqslant \bar{d}_j$ and $p_i \leqslant p_j$;
(2) $p_i \geqslant p_j$ implies $d_i - d_j \leqslant p_i - p_j$;
(3) a combination of the above two cases.

They used a backwards scheduling approach based on the concept of *tight schedule*.

In this note we propose a dual approach, the forwards greedy algorithms. This approach is based on the concept of independence system. As is well known, theory of independence system and matroid plays an important role in combinatorial optimization [6]. It is natural to ask which scheduling problems can be efficiently solved by using this theoretic tool.

Let us recall the concept of independence system [4,6]. Let E be a finite set and $\mathscr{I} \subseteq 2^E$ a family of subsets of E. Then $(E, \mathscr{I})$ is called an *independence system* if (i) $\emptyset \in \mathscr{I}$; (ii) $I \in \mathscr{I}, I' \subseteq I \Rightarrow I' \in \mathscr{I}$. Here, the subsets in $\mathscr{I}$ are called *independent sets* and those in $2^E \setminus \mathscr{I}$ are called *dependent sets*. Moreover, a *matroid* is an independence system that the greedy algorithm solves its maximum weight independent set problem for any weight function. In the sequel, we will establish an independence system for the considered scheduling models and show that a greedy algorithm works.

The rest of this note is organized as follows. In Section 2, we study Case (1), called the case of agreeable data. In Section 3, we apply the same approach to Case (2), called the case with convexity condition. Section 4 discusses the case of equal processing time.

2. The case of agreeable data

We are given n jobs $J_1, J_2, \ldots, J_n$ with the processing times $p_1, p_2, \ldots, p_n$, due dates $d_1, d_2, \cdots, d_n$, and deadlines $\bar{d}_1, \bar{d}_2, \ldots, \bar{d}_n$ respectively. We consider the special case that $d_i \leqslant d_j$ implies $\bar{d}_i \leqslant \bar{d}_j$ and $p_i \leqslant p_j$, denoted by $1|\bar{d}_j, agreeable|\sum U_j$ in short. Without loss of generality, we assume that all jobs have been indexed such that

$$d_1 \leqslant d_2 \leqslant \cdots \leqslant d_n, \quad p_1 \leqslant p_2 \leqslant \cdots \leqslant p_n, \quad \bar{d}_1 \leqslant \bar{d}_2 \leqslant \cdots \leqslant \bar{d}_n.$$

We denote by $E = \{1, 2, \ldots, n\}$ the set of jobs. Let $S_j(\sigma)$ and $C_j(\sigma)$ denote the starting time and completion time, respectively, of job j in the schedule σ.

Definition. For a subset $S \subseteq E$, the modified due dates are defined by

$$d'_j = \begin{cases} d_j & \text{if } j \in S \\ \bar{d}_j & \text{if } j \in E \setminus S \end{cases} \quad (j = 1, 2, \ldots, n).$$

* Corresponding author. Address: Department of Mathematics, Zhengzhou University, Zhengzhou, Henan 450052, People's Republic of China.
E-mail address: hech202@sina.com (C. He).

0377-2217/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2009.05.013

Here, jobs $j \in S$ are said to be *normal* (their due dates are unchanged) and jobs $j \in E \setminus S$ are said to be *relaxed* (each due date d_j is postponed to $\bar{d}_j$). A subset $S \subseteq E$ is called an *independent set* if there exists a schedule σ of all jobs without tardy jobs with respect to $\{d'_j\}(C_j(\sigma) \leqslant d'_j$ for all $j)$. Such a schedule is called a *feasible schedule* for S.

Note that if there exists a schedule without tardy jobs (i.e., all jobs are on time), then the schedule in EDD order (the earliest due date first rule) will do the same. So, in order to decide whether a set S is independent or not, we need only construct the schedule $\sigma(S)$ in the EDD order with respect to $\{d'_j\}$. A set S is independent if and only if $\sigma(S)$ is feasible (all jobs are on time). We will call this schedule $\sigma(S)$ the *checking schedule*.

It is clear that a subset S_0 of an independent set S is also independent, as S_0 has fewer restriction. Therefore all independent sets of E form an independence system $(E, \mathscr{I})$. Moreover, we have the following obvious fact:

Lemma 2.1. *Problem* $1|\bar{d}_j, agreeable|\sum U_j$ *is equivalent to finding an independent set with maximum cardinality in* $(E, \mathscr{I})$.

So our problem is to solve the maximum independent set problem in $(E, \mathscr{I})$. It is known in [7] that the problem $1\|\sum U_j$ can be solved by the greedy algorithm of the corresponding independence system, i.e., a dual version of the Moore algorithm. Now we apply a similar algorithm to the present case. Assume that the problem $1|\bar{d}_j, agreeable|\sum U_j$ is feasible (i.e., $\emptyset$ is independent). Otherwise we have nothing to do.

Greedy Algorithm

(0) *Let* $E = \{1, 2, \ldots, n\}$ *be the set of jobs in EDD order with respect to* $\{d_i\}$. *Set* $S = S' = \emptyset$.
(1) If $E = \emptyset$, *then stop.*
(2) *Choose a job* $k \in E$ *with the smallest processing time* p_k*; when there is a tie, choose the one with the smallest* d_k *(and further the smallest* $\bar{d}_k$*). Set* $E := E \setminus \{k\}$.
(3) *If* $S \cup \{k\}$ *is independent, then set* $S := S \cup \{k\}$*; otherwise set* $S' := S' \cup \{k\}$. *Return to (1).*

To prove the correctness of the algorithm, let us see the following key lemma.

Lemma 2.2. *Consider the first step of the algorithm. If* $S = \{1\}$ *is independent, then there exists a maximum independent set I containing job 1 and 1 is scheduled first in a feasible schedule for I. If* $S = \{1\}$ *is not independent and* $\bar{d}_1 \leqslant d_2$*, then there exists a maximum independent set I such that 1 is scheduled first in a feasible schedule for I (but* $1 \notin I$*).*

Proof. We show the first statement. Let I be a maximum independent set but $1 \notin I$. Suppose that i is the first job in $\sigma = \sigma(I)$, where $i \neq 1$. Then $p_1 \leqslant p_i$ and $\bar{d}_1 \leqslant \bar{d}_i$. We construct a schedule σ' from σ by exchanging i and 1. Since $C_i(\sigma') = C_1(\sigma) \leqslant \bar{d}_1 \leqslant \bar{d}_i$ and $C_1(\sigma') = p_1 \leqslant d_1$, we see that i meets its deadline and 1 is on time with respect to σ'. Moreover, the jobs between 1 and i are completing no later (as $p_1 \leqslant p_i$) and those after i are unchanged. Therefore σ' is a feasible schedule for $I^* = (I \setminus \{i\}) \cup \{1\}$ and 1 is scheduled first in it.

The second statement is more obvious. Since $\{1\}$ is dependent, it cannot be included in any maximum independent set. So $d'_1 = \bar{d}_1 \leqslant d_2 \leqslant \cdots \leqslant d_n$, thus 1 can be scheduled first in an EDD order with respect to $\{d'_j\}$, i.e., scheduled first in $\sigma(I)$ for a maximum independent set I. This completes the proof. □

Theorem 2.3. *Greedy Algorithm correctly solves the problem* $1|\bar{d}_j, agreeable|\sum U_j$.

Proof. We show the result by induction on n. It is trivially true for $n = 1$. Suppose the assertion holds for the case with less than n jobs. By Lemma 2.2, if $S = \{1\}$ is independent, then there exists an optimal solution I containing job 1 and 1 is scheduled at the first position of its feasible schedule. So we can solve the problem by fixing 1 at the first place, as this does not change the optimal value. Using the inductive hypothesis to the remaining $n - 1$ jobs, the result follows.

If $S = \{1\}$ is dependent and $\bar{d}_1 \leqslant d_2$, we can also fix 1 at the first place and use the inductive hypothesis. Otherwise $(\bar{d}_1 > d_2)$ we postpone job 1 and consider job 2. Let $\{k\}$ be the first independent set in the algorithm. Then $d'_i = \bar{d}_i$ for $1 \leqslant i \leqslant k-1$. If $\bar{d}_1 \leqslant d_k$, then 1 is always scheduled at the first place for an EDD order of $\{d'_j\}$. Thus 1 can be fixed at the first place as before. Otherwise $d_k < \bar{d}_1 \leqslant \cdots \leqslant \bar{d}_{k-1}$, we claim that there exists a maximum independent set I containing job k which is at the first place of $\sigma = \sigma(I)$. To see this, assume that I is a maximum independent set with the first job i, where $i > k$. So $p_k \leqslant p_i$ and $\bar{d}_k \leqslant \bar{d}_i$. Similar to Lemma 2.2, let σ' be the schedule which results from σ by exchanging i and k. Then $C_i(\sigma') = C_k(\sigma) \leqslant \bar{d}_k \leqslant \bar{d}_i$ and the jobs between i and k are completing no later (as $p_k \leqslant p_i$). Thus σ' is a feasible schedule for $I^* = (I \setminus \{i\}) \cup \{k\}$, proving the claim. Therefore we can fix k at the first place and go to the induction procedure, completing the proof. □

The above Greedy Algorithm is only an elementary framework, which may take much time to decide the independence (feasibility). In the following we propose an implementation scheme which can save some running time. Herein, the current S is not necessarily an independent set, but just a temporary candidate and some jobs may be dropped from it later. S' is also a temporary set for abandoned jobs (its jobs will be reconsidered in the subsequent steps). Moreover, we use E to denote the current set of jobs under consideration, which may include some ones in S'. This E is always treated as a list sequenced by the EDD order with respect to the modified due dates $\{d'_j\}$. So, if a job k that replaces d_k by $\bar{d}_k$ is put into E, we have to reorder the jobs in $E \cup \{k\}$. Suppose that E was originally sequenced by $d'_{j_1} \leqslant d'_{j_2} \leqslant \cdots d'_{j_m}$. Then k will be inserted at the place q such that $d'_{j_{q-1}} \leqslant \bar{d}_k \leqslant d'_{j_q}$. That is, make them to satisfy the new EDD order. It is known in sorting technique that this can be done in $O(\log|E|)$ time (say, by using binary search). In addition, we denote $S = \{i_1, i_2, \ldots, i_l\}$ as an ordered set $(l = |S|)$. Notice that we use the updating d_k to represent d'_k in the algorithm process.

Algorithm 1.

Step 0: Sequence all jobs in $E = \{1, 2, \ldots, n\}$ so that $d_1 \leqslant d_2 \leqslant \cdots \leqslant d_n$ and $p_i \leqslant p_j$ if $d_i = d_j (i < j)$, and further $\bar{d}_i \leqslant \bar{d}_j$ if $d_i = d_j$ and $p_i = p_j (i < j)$.
Step 1: Construct the initial schedule $\pi_0 = \sigma(\emptyset)$, i.e., the schedule according to the order of E. If π_0 is infeasible with respect to $\{\bar{d}_j\}$, then stop (the problem is infeasible). Otherwise let $S = S' = \emptyset, t = l = 0$.
Step 2: If $E = \emptyset$, then stop (the current S is a maximum independent set).
Step 3: Take the first job k in E.
(3.1) If $t + p_k \leqslant d_k$ and $k \notin S'$, then set $E := E \setminus \{k\}, l := l + 1, S := S \cup \{k\}, i_l := k$, and $t := t + p_k$.
(3.2) If $t + p_k \leqslant d_k$ and $k \in S'$, then set $E := E \setminus \{k\}$, $t := t + p_k$.
(3.3) If $t + p_k > d_k$ and $k \notin S'$, then set $S' := S' \cup \{k\}, d_k := \bar{d}_k$, and adjust the order of E (shift k to the place satisfying EDD order).
(3.4) If $t + p_k > d_k$ and $k \in S' (d_k = \bar{d}_k)$, then we have $t > 0$ (otherwise $p_k > \bar{d}_k$, the problem is infeasible, contradicting the decision of *Step 1*). Set

$S := S \setminus \{i_l\}, S' := S' \cup \{i_l\}, d_{i_l} := \bar{d}_{i_l}, E := E \cup \{i_l\}$, and adjust the order of E (shift i_l to the place satisfying EDD order). Reset $t := t - p_{i_l}$ and $l := l - 1$.

Return to *Step 2*.

Note that when we go back from (3.4) to *Step 2* (namely to *Step 3*), job k is still contained in E as its first element. So we execute *Step 3* on k again. If (3.2) occurs, then we get rid of k from E; If (3.4) occurs, the last job of S (the current i_l) is shifted from S to S'. We keep on running the procedure in this way.

Theorem 2.4. *Algorithm 1 correctly solves the problem* $1|\bar{d}_j, agreeable|\sum U_j$ *in* $O(n \log n)$ *time.*

Proof. Let S_l and S'_l be the final sets of S and S', respectively, in Algorithm 1. Meanwhile, let S_0 and S'_0 be the final sets of S and S', respectively, in the Greedy Algorithm. We still use S and S' to denote the current sets in the specific algorithm. When the algorithm terminates, we get a schedule σ, for which all jobs $j \in S_l$ are on time $(C_j(\sigma) \leqslant d_j)$ and all jobs $j \in S'_l$ meet their deadlines $(C_j(\sigma) \leqslant \bar{d}_j)$. So S_l is an independent set and $\sum U_j = |S'_l|$. To prove that S_l is a maximum independent set, we may compare it with the maximum independent set S_0 obtained by the Greedy Algorithm. For any job $k \in S'_l$, we put it into S' if and only if (3.3) or (3.4) is executed (putting it into S will violate the feasibility). When executing the Greedy Algorithm on this job k, we can see the same result that $S \cup \{k\}$ is not independent. So $k \in S'_0$. Therefore $S'_l \subseteq S'_0$, and thus $S_0 \subseteq S_l$. Since S_0 is maximum, so is S_l. In this way, we show the correctness of the algorithm. Let us next see the running time. *Step 0* and *Step 1* take $O(n \log n)$ time for sorting the jobs. We call *Step 2* and *Step 3* a stage. During the process of algorithm, each job is examined at most twice: one time for due date d_k (entering S) and one time for deadline $\bar{d}_k$ (entering S'). Therefore the number of stages is at most $2n$. In each stage, (3.1) and (3.2) can be executed in constant time. In (3.3) and (3.4), adjusting the order of E (inserting a number into a nondecreasing sequence) can be done in $O(\log n)$ time. So the overall running time is $O(n \log n)$. This completes the proof. □

3. The case with convexity condition

This section deals with the case with condition that $p_i \geqslant p_j$ implies $d_i - d_j \leqslant p_i - p_j$, i.e., $p_j + d_i \leqslant p_i + d_j$. In many combinatorial optimization problems, this type of conditions are called *Monge condition* or *discrete convexity* (see Burkard [2]). So we call the condition "$p_i \geqslant p_j \Rightarrow d_i - d_j \leqslant p_i - p_j$" a *convexity condition* here and denote this case by $1|\bar{d}_j, convex|\sum U_j$.

Note that all data d_i, p_i, $\bar{d}_i$ are not necessarily agreeable now. However, we still number all jobs of $E = \{1, 2, \ldots, n\}$ in EDD order $(d_1 \leqslant d_2 \leqslant \cdots \leqslant d_n)$. Of course it holds that $d_i \leqslant \bar{d}_i$ for $i = 1, 2, \ldots, n$. As before, we call a subset $S \subseteq E$ an independent set if there exists a schedule such that all jobs are on time (feasible) with respect to the modified due dates

$$d'_j = \begin{cases} d_j & \text{if } j \in S \\ \bar{d}_j & \text{if } j \in E \setminus S \end{cases} \quad (j = 1, 2, \ldots, n).$$

We may take the schedule of EDD order with respect to $\{d'_j\}$ as the *checking schedule*, denoted by $\sigma(S)$. It is clear that if $S \subseteq I$ and I is independent, then S is also independent and the checking schedule of I is also feasible for S. So $(E, \mathscr{I})$ constitutes an independence system, where $\mathscr{I} \subseteq 2^E$ is the family of all independent sets. The main goal of this section is to show that the Greedy Algorithm mentioned above also works.

It should be emphasized that in the checking schedule $\sigma(S)$, the jobs of S are always sequenced in EDD order of $\{d_i\}$, regardless of the order of $\{p_i\}$, and the jobs of $E \setminus S$ are in EDD order of $\{\bar{d}_i\}$. Two types of jobs in S and in $E \setminus S$ may appear alternately, depending on the order of $\{d'_i\}$. In particular, the first chosen job k_1 is not necessarily put at the first place of the corresponding feasible schedule.

Lemma 3.1. *In Greedy Algorithm, suppose that the current* S *is an independent set included in some maximum independent set* I. *Let* $k \in E \setminus (S \cup S')$ *be a job with the smallest processing time* p_k. *If* $S \cup \{k\}$ *is independent, then there exists a maximum independent set* I^* *including* $S \cup \{k\}$.

Proof. Suppose that the maximum independent set I including S does not contain k. Then $D = I \cup \{k\}$ is a dependent set and thus strictly includes $S \cup \{k\}$. So $I \setminus S \neq \emptyset$. We focus our attention on the following two schedules (job sequences): $\sigma = \sigma(I)$ is the checking schedule of I, and $\sigma^0 = \sigma(S \cup \{k\})$ is the schedule for checking the independence of $S \cup \{k\}$ during the greedy algorithm.

Let us first show that there exists some $i \in I \setminus S$ preceding k in σ. If this is not the case, i.e., all jobs in $I \setminus S$ are after k in σ, let A be the set of jobs scheduled not later than k in σ. Then $A \cap (I \setminus S) = \emptyset$. Denote by σ_A the partial schedule of A in σ, and denote by σ^0_A the partial schedule of A in σ^0. The only difference of σ_A and σ^0_A may be the position of k: since k is changed from a relaxed job in σ_A to a normal job in σ^0_A, k may be moved earlier to meet its due date d_k. Now we construct a new schedule σ' from σ by replacing σ_A by σ^0_A. Then σ' is a feasible schedule of $I \cup \{k\}$, a contradiction.

Therefore $I_0 = \{i \in I \setminus S : i \text{ precedes } k \text{ in } \sigma(I)\} \neq \emptyset$. Let i_0 be the job in I_0 with the largest completion time $C_{i_0}(\sigma^0)$. So $\bar{d}_{i_0} = \max\{\bar{d}_i : i \in I_0\}$. It follows from the algorithm that $p_k \leqslant p_{i_0}$. We proceed to show the following:

Claim $I^* = (I \cup \{k\}) \setminus \{i_0\}$ *is independent.*

This transformation from I to I^* means that i_0 is changed from a normal job (with respect to d_{i_0}) to a relaxed job (with respect to $\bar{d}_{i_0}$), k is changed from a relaxed job (with respect to $\bar{d}_k$) to a normal job (with respect to d_k). There are two cases to consider:

Case 1: $\bar{d}_{i_0} \geqslant C_k(\sigma)$.

We construct a schedule σ' from σ by exchanging i_0 and k. By the convexity condition, $d_k \geqslant d_{i_0} - (p_{i_0} - p_k) \geqslant C_{i_0}(\sigma) - (p_{i_0} - p_k) = C_k(\sigma')$. So job k becomes an on-time job in σ'. On the other hand, since $C_{i_0}(\sigma') = C_k(\sigma) \leqslant \bar{d}_{i_0}, i_0$ becomes a relaxed job satisfying the deadline. Furthermore, as $p_k \leqslant p_{i_0}$, the jobs between i_0 and k are completing no later in σ' than in σ. Hence σ' is a feasible schedule for $I^* = (I \cup \{k\}) \setminus \{i_0\}$.

Case 2: $\bar{d}_{i_0} < C_k(\sigma)$, i.e., $\bar{d}_i < C_k(\sigma)$ for all $i \in I_0$.

Denote $T = C_k(\sigma)$. Then the jobs scheduled in $[0, T]$ for σ are the same as those in $[0, T]$ for σ^0. Furthermore, for any job j scheduled in $[C_{i_0}(\sigma^0), T]$ for σ^0, due to the choice of i_0, we see that either $j \in S$ (with $d'_j = d_j$) or $j \in E \setminus I$ (with $d'_j = \bar{d}_j$). So, from σ^0 to σ, the order of i_0 and j is unchanged, as d'_j is unchanged. Therefore, this job j must be scheduled after i_0 in σ.

Now we construct a schedule σ' from σ by the following transformation:

(a) Take i_0 and the jobs in $[C_{i_0}(\sigma^0), T]$ for σ^0 and schedule them in the same time interval as that in σ^0.
(b) Shift k earlier to the place that $C_k(\sigma') = C_{i_0}(\sigma) - p_{i_0} + p_k$.
(c) Keep the order of other jobs in $[0, T]$ unchanged and keep the jobs after T fixed.

By (a), i_0 and the jobs between $C_{i_0}(\sigma^0)$ and T satisfy their modified due dates (since they are scheduled as in σ_0). By (b) and the convexity, $C_k(\sigma') = C_{i_0}(\sigma) - p_{i_0} + p_k \leqslant d_{i_0} - p_{i_0} + p_k \leqslant d_k$. So k is

on time. By (c) and $p_k \leqslant p_{i_0}$, the jobs between k and i_0 are completing no later in σ' than in σ. Furthermore, the jobs before $S_k(\sigma')$ are scheduled as in σ. So σ' is a feasible schedule for I^*.

To summarize, the claim is proved, as required. □

Theorem 3.2. *The above-mentioned Greedy Algorithm correctly solves the problem* $1|\bar{d}_j, convex|\sum U_j$ *in* $O(n^2)$ *time.*

Proof. By Lemma 3.1, throughout the algorithm process, the current set S is always included in an optimal solution I^*. So S is optimal whenever the algorithm terminates. It is clear that the algorithm has at most n iterations and each iteration takes $O(n)$ time to decide the independence. Hence the overall running time is $O(n^2)$. □

4. The case of equal processing time

This section studies the case of equal processing time, denoted by $1|\bar{d}_j, p_i = p|\sum U_j$, which has a simple combinatorial structure.

We first formulate this case as a bipartite matching problem. So it is a special case of two-matroid intersection problem [4,6]. Let G be a bipartite graph with bipartition (U, V), where $U = \{J_1, J_2, \ldots, J_n\}$ is the set of jobs, $V = \{T_1, T_2, \ldots, T_n\}$ is the set of positions; and (J_i, T_j) is an edge if and only if $jp \leqslant \bar{d}_i$. The weight of each edge (J_i, T_j) is defined by

$$w_{ij} = \begin{cases} 0 & \text{if } jp \leqslant d_i \\ 1 & \text{if } d_i < jp \leqslant \bar{d}_i. \end{cases}$$

It is easy to see that the scheduling problem $1|\bar{d}_j, p_i = p|\sum U_j$ is equivalent to finding a minimum weight perfect matching of G. So it can be efficiently solved by bipartite matching algorithms (see [4,6]). However, we show that a variant of Greedy Algorithm works. Instead of considering in the order of $\{p_j\}$, we pay attention to the order of deadlines $\{\bar{d}_j\}$.

Algorithm 2

Step 0: Sequence all jobs in $E = \{1, 2, \ldots, n\}$ in the order $\bar{d}_1 \leqslant \bar{d}_2 \leqslant \cdots \leqslant \bar{d}_n$ and $d_i \leqslant d_j$ if $\bar{d}_i = \bar{d}_j (i < j)$. Set $S = S' = \emptyset$.
Step 1: Check whether the initial schedule (with $S = \emptyset$) is feasible or not. If not, stop (the problem is infeasible).
Step 2: If $E = \emptyset$, then stop (S is optimal).
Step 3: Choose the first job k in E. Set $E := E \setminus \{k\}$.
Step 4: If $S \cup \{k\}$ is independent, then set $S := S \cup \{k\}$; otherwise set $S' := S' \cup \{k\}$. Return to *Step 2*.

To see the correctness of the algorithm, we need a result similar to Lemma 3.1.

Lemma 4.1. *In Algorithm 2, suppose that the current S is an independent set included in some maximum independent set I. Let $k \in E \setminus (S \cup S')$ be a job with the smallest deadline $\bar{d}_k$. If $S \cup \{k\}$ is independent, then there exists a maximum independent set I^* including $S \cup \{k\}$.*

Proof. The proof is also similar to that of Lemma 3.1. Suppose that $k \notin I$. Then $I \cup \{k\}$ strictly includes $S \cup \{k\}$ and thus $I \setminus S \neq \emptyset$. Let $\sigma = \sigma(I)$ and $\sigma^0 = \sigma(S \cup \{k\})$. It is clear that $I_0 = \{i \in I \setminus S : i \text{ precedes } k \text{ in } \sigma\} \neq \emptyset$. Let i_0 be the job in I_0 with the smallest due date d_{i_0}. Then $\bar{d}_k \leqslant \bar{d}_{i_0}$ by the choice of k in the algorithm. There are two cases to consider:

(1) If $C_{i_0}(\sigma) \leqslant d_k$, then we construct a schedule σ' from σ by exchanging i_0 and k. Clearly, $C_k(\sigma') = C_{i_0}(\sigma) \leqslant d_k, C_{i_0}(\sigma') = C_k(\sigma) \leqslant \bar{d}_k \leqslant \bar{d}_{i_0}$. Hence σ' is feasible for $I^* = (I \cup \{k\}) \setminus \{i_0\}$.

(2) If $C_{i_0}(\sigma) > d_k$, then we construct σ' from σ as follows: (a) shift i_0 to the place of k; (b) shift k earlier to the place that $C_k(\sigma') = C_k(\sigma^0)$; (c) fix the order of other jobs. It is easy to verify that σ' is feasible for I^*. In fact, $C_{i_0}(\sigma') = C_k(\sigma) \leqslant \bar{d}_k \leqslant \bar{d}_{i_0}$. On the other hand, let $t_0 = S_{i_0}(\sigma)$ and let A be the set of jobs scheduled before t_0 in σ. Then those of $A \cup \{k\}$ can be scheduled before $t_0 + p$ as in σ^0. So $C_k(\sigma') = C_k(\sigma^0) \leqslant d_k$. And the other jobs are feasible.

The proof is completed. □

Theorem 4.2. *Algorithm 2 correctly solves the problem* $1|\bar{d}_j, p_i = p|\sum U_j$ *in* $O(n^2)$ *time.*

Proof. The correctness is based on Lemma 4.1. To see the running time, observe that the sortings in *Step 0* and *Step 1* take $O(n \log n)$ time. And the number of iterations (from *Step 2* to *Step 4*) is at most n. In each iteration, the independence decision takes $O(n)$ time. Therefore the overall running time is $O(n^2)$, as desired. □

The above algorithm could be called a forwards greedy algorithm. Symmetrically, we have a backwards greedy algorithm as follows.

Algorithm 2*.

Step 0: The set of jobs in $E = \{1, 2, \ldots, n\}$ is treated as an ordered set with the order that $d_1 \leqslant d_2 \leqslant \cdots \leqslant d_n$ (and $\bar{d}_i \leqslant \bar{d}_j$ if $d_i = d_j (i < j)$). Meanwhile, we keep another ordered set $D = \{i_1, i_2, \ldots, i_n\}$ with the order that $\bar{d}_{i_1} \leqslant \bar{d}_{i_2} \leqslant \cdots \leqslant \bar{d}_{i_n}$ (and $d_{i_k} \leqslant d_{i_l}$ if $\bar{d}_{i_k} = \bar{d}_{i_l} (k < l)$), where $\{i_1, i_2, \ldots, i_n\}$ is a permutation of $\{1, 2, \ldots, n\}$. We use a doubly linked list L to store each job with two corresponding labels of E and D. Let $S = \emptyset$ and $t = \Sigma_{j=1}^{n} p_j$.
Step 1: If $E = \emptyset$, then stop (S is optimal).
Step 2: Take the last job k of E. If $d_k \geqslant t$, then set $S := S \cup \{k\}$ and $E := E \setminus \{k\}, D := D \setminus \{k\}$. Set $t := t - p$ and return to *Step 1*.
Step 3: If $d_k < t$, then take the last job j of D. If $\bar{d}_j < t$, then stop (the problem is infeasible). Otherwise, choose j_0 as the job with the smallest due date d_{j_0} from among those j with $\bar{d}_j \geqslant t$. Set $E := E \setminus \{j_0\}$ and $D := D \setminus \{j_0\}$. Set $t := t - p$ and return to *Step 1*.

Theorem 4.3. *Algorithm 2* correctly solves the problem* $1|\bar{d}_j, p_i = p|\sum U_j$ *in* $O(n \log n)$ *time.*

Proof. To show the correctness, it suffices to have a claim that there exists an optimal schedule such that job k (in the case of *Step 2*) or job j_0 (in the case of *Step 3*) is scheduled last. If this is not the case, we may exchange it to the last place without changing the cost. We next consider the running time. In *Step 0*, we sort the jobs in E and D (and build the list L) in $O(n \log n)$ time. There are at most n stages (*Steps 1–3*) in the algorithm. In each stage, *Step 2* can be performed in constant time. Note that when a job k is taken from E, we can get the same element in D by the doubly linked list L, and then delete it from both E and D. In *Step 3*, the construction of $J_t = \{j \in D : \bar{d}_j \geqslant t\}$ takes $O(\log n)$ time. In order to choose j_0 from J_t with the smallest d_j, we define a sequence (or queue) Q of J_t in the nondecreasing order of due dates. This can be done by sorting in $O(|J_t| \log n)$ time. The first element of Q is the required job j_0, which is deleted immediately after *Step 3* is executed. In the next time that *Step 3* is executed, for the new t, some more jobs would be put into J_t. Then we can insert these new elements into sequence Q in $O(h \log n)$ time, where h is the number of new elements. And the first element of Q is the new j_0. Let $n_1, n_2, \ldots, n_l$

be the number of new elements in J_t when *Step 3* is executed successively. Then the total running time of choosing j_0 in all times of *Step 3*. $O(n_1 \log n + n_2 \log n + \cdots + n_l \log n) = O(n \log n)$. Moreover, when j_0 is taken from D, we can locate it in E by the list L and delete it immediately. Therefore, the overall running time is $O(n \log n)$, as desired. □

Acknowledgements

The authors would like to thank the referees for their helpful comments. This work was supported by NSFC (Grant No. 10671183) and NFSC-RGC (Grant No. 70731160633) and SRFDP (Grant No. 20070459002).

References

[1] P. Brucker, Scheduling Algorithms, third ed., 2003.
[2] R.E. Burkard, Monge properties, discrete convexity and applications, European Journal of Operational Research 176 (2007) 1–14.
[3] Y.M. Huo, J.Y.T. Leung, H.R. Zhao, Bi-criteria scheduling problems: Number of tardy jobs and maximum weighted tardiness, European Journal of Operational Research 177 (2007) 116–134.
[4] B. Korte, J. Vygen, Combinatorial Optimization: Theory and Algorithms, fourth ed., Springer-Verlag, 2008.
[5] E.L. Lawler, Scheduling a single machine to minimize the number of late jobs, Unpublished manuscript.
[6] E.L. Lawler, Combinatorial Optimization: Networks and Matroids, Holt Rinehart and Winston, New York, 1976.
[7] Y.X. Lin, X.M. Wang, Necessary and sufficient conditions of optimality for some classical scheduling problems, European Journal of Operational Research 176 (2007) 809–818.