Algorithms for Maximizing the Number of Tardy Jobs or Total Tardiness on a Single Machine

Evgeny R. Gafarov [a], **Alexander A. Lazarev** [b]

Institute of Control Sciences of the Russian Academy of Sciences,
Profsoyuznaya st. 65, 117997 Moscow, Russia,
email: [a] *axel73@mail.ru,* [b] *jobmath@mail.ru*

Frank Werner

Fakultät für Mathematik, Otto-von-Guericke-Universität Magdeburg,
PSF 4120, 39016 Magdeburg, Germany,
email: frank.werner@mathematik.uni-magdeburg.de

November 20, 2009

Abstract

In this paper, we consider two scheduling problems on a single machine, where a specific objective function has to be maximized in contrast to usual minimization problems. We propose exact algorithms for the single machine problem of maximizing total tardiness $1||\max\sum T_j$ and for the problem of maximizing the number of tardy jobs $1||\max\sum U_j$. In both cases, it is assumed that the processing of the first job starts at time zero and there is no idle time between the jobs. We show that problem $1||\max\sum U_j$ is polynomially solvable. For several special cases of problem $1||\max\sum T_j$, we present exact polynomial algorithms. Moreover, we give an exact pseudo-polynomial algorithm for the general case of the latter problem and an alternative exact algorithm.

Keywords: Scheduling, single machine problems, maximization problems, total tardiness, number of tardy jobs

MSC: 90 B 35

Introduction

Typically, in scheduling theory problems are considered where a specific objective function has to be minimized. For instance, the minimization of makespan is a very popular optimization criterion. When a sum function is considered, often total completion time, total tardiness or the number of tardy jobs have to be minimized.

In this paper, we consider a single machine problem with two 'opposite' criteria, namely we consider the maximization of total tardiness and the maximization of the number of tardy jobs. On one side, investigations of such problems with the 'opposite' optimization criterion itself are an important theoretical task. For instance, they can be used for proposing algorithms and deriving properties of an optimal schedule of this opposite problem which might be taken to compute upper bounds for the original problems. Algorithms for such problems with opposite optimization criterion can be used to compute parts of optimal schedules for the original problems, or to cut bad sub-problems in the branching tree of branch-and-bound algorithms. For example, the algorithm for problem $1||\max\sum U_j$ can be used to compute the maximal number of tardy jobs, and later we can use this number to reduce the search for an optimal solution of problem $1||\max\sum T_j$ (see Section 2 of this paper). In addition, it is interesting from a theoretical point whether polynomially solvable cases for the minimization problem are also easy for the maximization case, or vice versa. Moreover, there are often relationships between the minimization and maximization variants, e.g. between problem $1||\max\sum U_j$ and the problem of minimizing the number of early jobs $1||\min\sum E_j$ when idle times are not allowed.

On the other side, such problems separately have practical interpretations and applications. For instance, a company may additionally earn money for the case of latest completion times of the individual orders. Another situation arises when the company is considered as a customer, and one wants to know the worst variant of a schedule, which is computed in a 'black box' (e.g. a plant). Finally, we mention the problem of a 'saboteur'. That is, a saboteur wants to construct the worst schedule in a competitive ('enemy') plant.

It is well-known that problem $1|r_j|L_{max}$ is NP-hard in the strong sense while it has been shown in [1] that the 'opposite' maximization problem $1|r_j|\max f_{min}$ can be polynomially solved for any non-decreasing functions $f_j(t)$ for all $j \in N$ in $O(n^2)$ time.

The problems under consideration can be formulated as follows. We are given a set $N = \{1, 2, \ldots, n\}$ of n independent jobs that must be processed on a single machine. Preemptions of a job are not allowed. The machine

can handle only one job at a time. All the jobs are assumed to be available for processing at time 0. For each job j, $j \in N$, a processing time $p_j > 0$ and a due date d_j are given. A schedule $\pi = (j_1, j_2, \ldots, j_n)$ is uniquely determined by a permutation of the jobs of set N. We assume that *idle times are not allowed.* Let $C_{j_k}(\pi) = \sum_{l=1}^{k} p_{j_l}$ be the completion time of job j_k in schedule π. If $C_j(\pi) > d_j$, then job j is tardy and we have $U_j = 1$, otherwise $U_j = 0$. If $C_j(\pi) \leq d_j$, then job j is on-time. The objective is to find an optimal schedule π^* that maximizes the number of tardy jobs, i.e., $F(\pi) = \sum_{j=1}^{n} U_j(\pi)$. We will denote this problem by $1||\max \sum U_j$. It is well-known that problem $1||\min \sum U_j$ is polynomially solvable in $O(n \log n)$ time by Moore's algorithm [2].

If $C_j(\pi) < d_j$, then job j is early (in fact, it is completed before the due date) and we set $E_j = 1$, otherwise $E_j = 0$. We will denote by $1||\min \sum E_j$ the problem of minimizing the number of early jobs, where idle times between the processing of the jobs are not allowed. There is a relationship between problems $1||\max \sum U_j$ and $1||\min \sum E_j$. Let $d_j,\ p_j \in Z$ for all $j \in N$ and δ be a real number with $0 < \delta < 1$. If we set $d'_j = d_j + \delta$, then problems $1|d_j|\max \sum U_j$ and $1|d'_j|\min \sum E_j$ are identical. If one job is tardy for problem $1|d_j|\max \sum U_j$ in schedule π, then $C_j(\pi) > d_j$ and $C_j(\pi) > d_j + \delta = d'_j$, i.e. job j is not early.

Moreover, let $T_j(\pi) = \max\{0, C_j(\pi) - d_j\}$ be the tardiness of job j in schedule π. We will denote the problem of maximizing total tardiness without idle times between the processing of jobs by $1||\max \sum T_j$. In this paper, notation $\{\pi\}$ denotes the set of jobs contained in sequence π. Notation $i \in \pi$ means $i \in \{\pi\}$. Notation $j \rightarrow i$ means, that the processing of job j precedes the processing of job i, and notation $(j \rightarrow i)_\pi$ means that this holds in schedule π.

The rest of the paper is organized as follows. In Section 1, we present a polynomial algorithm for problem $1||\max \sum U_j$. In Section 2, we consider the problem of maximizing total tardiness on a single machine. In particular, we present polynomial algorithms for several special cases of this problem. First, we adapt in Section 2.1 the elimination rules by Emmons to the maximization problem. In Section 2.2, we present an overview on polynomially solvable and NP-hard cases for minimizing total tardiness and we give an overview on the results we present in Section 2 for the maximization case. In Section 2.3, we propose several properties of optimal solutions. In Sections 2.4 - 2.7, we develop algorithms for polynomially solvable cases of the problem of maximizing total tardiness. For the general case, an exact pseudo-polynomial algorithm is given in Section 2.8, and an alternative exact algorithm is given in Section 2.9.

1 A polynomial algorithm for problem $1||\max\sum U_j$ without idle times

In this section, we first discuss a relationship between problem $1|r_j|C_{max}$ and the problem of maximizing the number of tardy jobs. Then we present an algorithm for constructing a schedule with all jobs tardy provided that such a schedule does exist. Finally, we present in Section 1.3 properties of an optimal schedule and present Algorithm A3 for solving this problem.

1.1 A relationship with problem $1|r_j|C_{max}$.

Let us set $r_j = d_j - p_j + \delta$ for all $j \in N$. We consider problem $1|r_j|C_{max}$. There exists an exact Algorithm **A1** for this problem with time complexity $O(n \log n)$. However, we note that in an optimal schedule, there are idle times in general. To determine an optimal schedule, the following algorithm computes the starting time S_j of each job $j \in N$.

Algorithm A1:

1. Renumber the jobs according to non-decreasing release dates, i.e., $r_1 \leq r_2 \leq \ldots \leq r_n$;
2. $t := 0$;
3. For $j = 1, \ldots, n$ do:
   $S_j := \max(r_j, t)$;
   $t := S_j + p_j$;
4. $C_{max} = S_n + p_n$.

If $C_{max} > \sum_{j=1}^{n} p_j$, then there is at least one tardy job in each schedule without idle times.

1.2 An algorithm for constructing a schedule with all jobs tardy

Next, we present an algorithm that constructs a schedule having only tardy jobs if it exists. The algorithm is as follows.

Algorithm A2:

1. $t := 0,\ N' := N$;
2. For $i = 1, \ldots, n$ do:

2.1. Select a job $j \in N'$ such that $t + p_j > d_j$ (i.e., this job can be tardy). If there are several such jobs, select the job with maximal processing time;

2.2. If there is no such job, then a schedule, where all jobs are tardy, does not exist. Stop.

2.3. $t := t + p_j,\ N' := N' \setminus \{j\}$;

Lemma 1 *Assume that there exists a schedule, where all jobs are tardy. Then Algorithm* **A2** *constructs such a schedule.*

Proof. We prove the lemma indirectly. Assume that Algorithm **A2** has constructed a subsequence $\pi = (j_1, j_2, \ldots, j_k), k < n$, as the first part of the schedule, where all jobs are tardy. Moreover, let there exist a schedule $\pi_0 = (\pi_a, j_1, \pi_b, j_2, \ldots),\ \pi_a = (j_1^a, j_2^a, \ldots)$, where all n jobs are tardy. Then, in schedule $\pi_1 = (j_1, \pi_a, \pi_b, j_2, \ldots)$, all jobs are also tardy since $p_{j_1} > d_{j_1}$. Similarly, in schedule $\pi_2 = (j_1, j_2, \pi_a, \pi_b, \ldots)$, all jobs are also tardy, and so on. However, in schedule $\pi_k = (j_1, j_2, \ldots, j_k, j_1^a, \ldots)$ job j_1^a is not tardy. This yields a contradiction. $\square$

1.3 Properties of an optimal schedule for problem $1||\max \sum U_j$

In this section, we present some properties of an optimal schedule and an algorithm for solving this problem.

Lemma 2 *There exists an optimal schedule of the form $\pi = (S, F)$, where all jobs $j \in F$ are tardy and all jobs $j \in S$ are not tardy.*

Proof. Assume that there exists an optimal schedule $\pi = (j_1, j_2, j_3, \ldots, j_{k-1}, j_k, \ldots, j_n)$, where job j_{k-1} is tardy and job j_k is not tardy. Then $\pi' = (j_1, j_2, j_3, \ldots, j_k, j_{k-1}, \ldots, j_n)$ is also an optimal schedule. Performing such adjacent pairwise interchanges of two jobs repeatedly, we get the schedule $\pi = (S, F)$. $\square$

Lemma 3 *There exists an optimal schedule $\pi = (S, F)$, which has no jobs $j \in F$ and $i \in S$ with $p_j > p_i$ and $d_j \geq d_i$.*

Proof. Assume that there exists an optimal schedule $\pi = (j_1, j_2, j_3, \ldots, j_k, j_{k+1}, \ldots, j_{l-1}, j_l, \ldots, j_n)$, where $p_{j_l} > p_{j_k}, d_{j_l} \geq d_{j_k}$, $C_{j_l}(\pi) > d_{j_l}$ and $C_{j_k}(\pi) \leq d_{j_k}$. Then schedule $\pi' =$

$(j_1, j_2, j_3, \ldots, j_l, j_{k+1} \ldots, , j_{l-1}, j_k, \ldots, j_n)$ is also optimal since $C_k(\pi') = C_{j_l}(\pi) > d_{j_l} \geq d_{j_k}$ and for all $j_i = j_{k+1}, \ldots, j_{l-1}$, inequality $C_{j_i}(\pi') > C_{j_i}(\pi)$ holds since $p_{j_l} > p_{j_k}$. $\square$

Now we set $r_j = d_j - p_j + \delta$ for all jobs $j \in N$.

Lemma 4 *There exists an optimal schedule $\pi = (S, F)$, where all tardy jobs $j_{k+1}, j_{k+2}, \ldots, j_n$ are processed according to non-decreasing release dates, i.e., $r_{j_i} \leq r_{j_{i+1}}, i = k+1, \ldots, n-1$.*

Proof. Assume that there exists an optimal schedule $\pi = (\pi_1, i, j, \pi_2)$, where jobs i and j are tardy, but $r_j < r_i$. It is well-known that, if job i is tardy, then $C_i(\pi) - p_i \geq r_i > r_j$. Let us consider schedule $\pi' = (\pi_1, j, i, \pi_2)$. We have $C_j(\pi') - p_j = C_i(\pi) - p_i \geq r_i > r_j$ and $C_i(\pi') = C_i(\pi) + p_j > d_i + p_j > d_i$. Thus, π' is an optimal schedule, where both jobs i and j are also tardy. $\square$

Algorithm A3.

Step 1. $S := \emptyset; F := N$;

Step 2. We construct a schedule of the form $\pi = (j_1^s, j_2^s, \ldots, j_k^s, j_{k+1}^f, j_{k+2}^f, \ldots, j_n^f)$. All jobs from set $S = \{j_1^s, j_2^s, \ldots, j_k^s\}$ are processed at the beginning of the schedule. All jobs from set $F = \{j_{k+1}^f, j_{k+2}^f, \ldots, j_n^f\}$ are processed according to non-decreasing release dates: $r_{j_{k+1}^f} \leq r_{j_{k+2}^f} \leq \ldots \leq r_{j_n^f}$;

Step 3. We find the last on-time job $j_l^f \in F$. If there is no such job, then GOTO Step 6;

Step 4. We select the job $j^* \in \{j_l^f, j_{l+1}^f, \ldots, j_n^f\}$ with the largest processing time: $p_{j^*} \geq p_i$ for all $i \in \{j_l^f, j_{l+1}^f, \ldots, j_n^f\}$;

Step 5. $S := S \cup \{j^*\}$, $F := F \setminus \{j^*\}$, GOTO Step 2;

Step 6. An optimal schedule $\pi = (j_1^s, j_2^s, \ldots, j_k^s, j_{k+1}^f, j_{k+2}^f, \ldots, j_n^f)$ has been obtained. STOP.

Lemma 5 *Let in schedule $\pi = (j_1, \ldots, j_l, j_{l+1}, \ldots, j_n)$, $r_{j_1} \leq r_{j_2} \leq \ldots \leq r_{j_n}$, job j_l be the last on-time job. Then there exists an optimal schedule, where job $j^* \in \{j_l, j_{l+1}, \ldots, j_n\}$, $p_{j^*} \geq p_i$, for all $i \in \{j_l, j_{l+1}, \ldots, j_n\}$ is on-time.*

Proof. Let there be an optimal schedule $\pi = (\pi_1, i, \pi_2, j^*, \pi_3, \pi_4)$, where job j^* is tardy. It is obvious that in any schedule, not all jobs $j_l, j_{l+1}, \ldots, j_n$ are tardy (see Lemma 4). Then there is an on-time job $i \in \{j_l, j_{l+1}, \ldots, j_n\},\ i \neq j^*$. According to Lemma 4, we assume that all jobs from set $\{j^*, \pi_3, \pi_4\}$ are processed according to non-decreasing release dates. If $r_{j^*} \geq r_i$, then $r_{j^*} + p_{j^*} \geq r_i + p_i$, i.e., $d_{j^*} \geq d_i$, and schedule $\pi' = (\pi_1, j^*, \pi_2, i, \pi_3, \pi_4)$ is also optimal. In schedule π', job j^* is not tardy and job i is tardy.

Now, let $r_{j^*} < r_i$. Assume that we have for all $j \in \{\pi_4\},\ r_j \geq r_i$ and for all $j \in \{\pi_3\},\ r_j < r_i$. Let us consider schedule $\pi'' = (\pi_1, j^*, \pi_2, \pi_3, i, \pi_4)$. For all $j \in \{\pi_2\}$, we have $C_j(\pi'') \geq C_j(\pi)$ since $p_{j^*} \geq p_i$. In schedule π'', all jobs from $\{\pi_3\} \bigcup \{i\} \bigcup \{\pi_4\}$ are tardy since in schedule $\pi''' = (\pi_1, \pi_2, j^*, \pi_3, i, \pi_4)$, only job j^* can be on-time according to the conditions of the lemma. Therefore, schedule π'' is optimal. $\square$

Lemma 6 *Algorithm* **A3** *constructs an optimal schedule for problem* $1||max \sum U_j$

Proof. We prove the lemma by induction. The algorithm is correct for $n = 1$.

Assume now that the algorithm is correct for $n-1$ jobs. We consider an instance with n jobs. Assume that Algorithm **A3** has constructed the schedule $\pi = (S, F)$, but there is an optimal schedule $\pi' = (S', F')$ with $|F'| \geq |F|$. If we use the algorithm for the $n-1$ jobs $1, 2, \ldots, j-1, j+1, \ldots, n$ with $j = j^*$, then we have an optimal schedule $(S \setminus \{j^*\}, F)$. Schedule $(S' \setminus \{j^*\}, F')$ is appropriate for the corresponding instance with $n-1$ jobs. Thus, we get $|F'| \leq |F|$ and $|F'| = |F|$. $\square$

Such an algorithm for problem $1||\min \sum E_j$ has been presented for the first time in [3] (unfortunately, no proof was given there).

2 Algorithms for problem $1||\max \sum T_j$ without idle times

In this section, we wish to construct an optimal schedule π^* which maximizes the total tardiness value $F(\pi) = \sum_{j=1}^{n} \max\{0, C_j(\pi) - d_j\}$, where $C_j(\pi)$ is the completion time of job j in schedule π. Again, idle times between the processing of jobs are not allowed and the first job starts at time 0.

Problem $1\ ||\ \min \sum T_j$ is NP-hard in the ordinary sense [4, 5]. A pseudo-polynomial dynamic programming algorithm of time complexity $O(n^4 \sum p_j)$

has been proposed by Lawler [6]. The state-of-the-art algorithms by Szwarc et al. [7] can solve special instances [8] of this problem for $n \leq 500$ jobs.

In the next subsection, we present some rules for eliminating specific solutions from the search for an optimal solution when maximizing total tardiness.

2.1 Elimination rules by Emmons

In this subsection, we adapt the elimination rules given by Emmons for the minimization problem to the maximization problem under consideration. We define B_j as the set of predecessors of job j in all optimal schedules, and A_j denotes the set of successors of job j in all optimal schedules, i.e., $B_j = \{i \in N : i \rightarrow j \text{ in all optimal schedules}\}$ and $A_j = \{i \in N : j \rightarrow i \text{ in all optimal schedules}\}$. Moreover, we define $c_j = \sum_{k=1}^{n} p_k - \sum_{i \in A_j} p_i$ and $s_j = \sum_{i \in B_j} p_i$. Then, we have $s_j + p_j \leq C_j \leq c_j$ for all optimal schedules.

Lemma 7 *If $p_j \geq p_i$ and $d_j \geq d_i$, or $p_j \geq p_i$, $d_j < d_i$ and $d_j \geq c_j$, or $p_j \geq p_i$, $d_j < d_i$ and $d_i \leq s_i + p_j$, then there exists an optimal schedule π, for which we have $(j \rightarrow i)_\pi$.*

Proof. We consider the following three cases.

a) Let $p_j \geq p_i$ and $d_j \geq d_i$. Assume that there is an optimal schedule $\pi = (\pi_1, i, \pi_2, j, \pi_3)$. For schedule $\pi' = (\pi_1, j, \pi_2, i, \pi_3)$, we have $F(\pi') - F(\pi) \geq (T_j(\pi') - T_j(\pi)) + (T_i(\pi') - T_i(\pi))$. If $C_j(\pi') > d_j$, then $F(\pi') - F(\pi) \geq -(p_i + \sum_{k \in \pi_2} p_k\}) + (p_i + \sum_{k \in \pi_2} p_k\}) \geq 0$. Otherwise, if $C_j(\pi') \leq d_j$, then $F(\pi') - F(\pi) \geq -\max\{0, C_j(\pi) - d_j\} + \max\{0, C_j(\pi) - d_i\} \geq 0$.

b) Let $p_j \geq p_i$, $d_j < d_i$ and $d_j \geq c_j$. Assume that there is an optimal schedule $\pi = (\pi_1, i, \pi_2, j, \pi_3)$. For schedule $\pi' = (\pi_1, j, i, \pi_2, \pi_3)$, we have $F(\pi') - F(\pi) \geq (T_j(\pi') - T_j(\pi)) + (T_i(\pi') - T_i(\pi)) = (0-0) + (0-0) \geq 0$.

c) Let $p_j \geq p_i$, $d_j < d_i$ and $d_i \leq s_i + p_j$. Assume that there is an optimal schedule $\pi = (\pi_1, i, \pi_2, j, \pi_3)$. For schedule $\pi' = (\pi_1, j, \pi_2, i, \pi_3)$, we have $F(\pi') - F(\pi) \geq (T_j(\pi') - T_j(\pi)) + (T_i(\pi') - T_i(\pi)) \geq (-p_i - \sum_{k \in \pi_2} p_k) + (\sum_{k \in \pi_2} p_k + p_i) = 0$.

In all cases schedule π' is also optimal. $\square$

Lemma 8 *If $p_j \geq p_i$ and $d_i > c_i - p_i$ and $d_j \leq s_j + p_j$, then there exists an optimal schedule π, for which we have $(i \rightarrow j)_\pi$.*

Proof. Assume that there exists an optimal schedule $\pi = (\pi_1, j, \pi_2, i, \pi_3)$. For schedule $\pi' = (\pi_1, i, j, \pi_2, \pi_3)$, we have $F(\pi') - F(\pi) \geq (T_j(\pi') - T_j(\pi)) + (T_i(\pi') - T_i(\pi)) = (p_i) - (\max\{c_i - d_i, 0\}) > 0$. $\square$

The following Table 1 compares the elimination rules for the minimization and maximization problems.

Table 1: Elimination rules

$p_j \geq p_i$	$1\|\|\min \sum T_j$	$p_j \geq p_i$	$1\|\|\max \sum T_j$
$(i \rightarrow j)$	$d_i \leq d_j$ or $d_i \leq s_j + p_j$	$(j \rightarrow i)$	$d_i \leq d_j$ or $(d_i > d_j$ and $d_j > c_j)$ or $(d_i > d_j$ and $d_i < s_i + p_i)$ or $(d_i > d_j$ and $d_i < s_i + p_j)$
$(j \rightarrow i)$	$d_i + p_i \geq c_j$ and $(d_i > d_j$ or $d_i > s_j + p_j)$	$(i \rightarrow j)$	$d_i > c_i - p_i$ and $d_j \leq s_j + p_j$

2.2 Special cases

Some polynomially solvable special cases for the problem of minimizing total tardiness are summarized e.g. by Lazarev and Werner [9]. In this subsection, we give an overview on the results obtained in this paper for the maximization problem, and we compare the results with those available for the minimization of total tardiness in Table 2.

2.3 Properties of an optimal schedule for problem $1||\max \sum T_j$

In this subsection, we present some properties of an optimal schedule for the problem under consideration.

Lemma 9 *There exists an optimal schedule $\pi = (S, F) = (SPT, LPT)$, where all jobs $j \in F$ are tardy and all jobs $i \in S$ are on-time. All jobs from the set S are processed in SPT (shortest processing time) order and all jobs from the set F are processed in LPT (longest processing time) order.*

Table 2: Polynomially solvable and NP-hard special cases

Special case	$1\|\|\min \sum T_j$		$1\|\|\max \sum T_j$	
	P/NP	Algorithm	P/NP	Algorithm
$d_j = d$	P	SPT	P	LPT *see Lemma 7*
$p_j = p$	P	EDD	P	$\pi_{opt} = (1, 2, \ldots, n)$, $d_1 \geq d_2 \geq \ldots \geq d_n$ *Lemma 7*
$d_1 \leq d_2 \leq \ldots \leq d_n$ $p_1 \leq p_2 \leq \ldots \leq p_n$	P	SPT	P	LPT *Lemma 7*
$d_{max} - d_{min} \leq 1,\ p_j \in Z$	P	$O(n^2)$ [9, 10]	P	$O(n^2)$ *Lemma 19*
$d_1 \leq d_2 \leq \ldots \leq d_n$ $d_i - d_{i-1} > p_i, i = 2, \ldots, n$	P	$O(n^2)$ [9, 10]	open	-
Canonical LG [5] $d_1 \leq d_2 \leq \ldots \leq d_n$ $p_1 \geq p_2 \geq \ldots \geq p_n$ $d_n - d_1 \leq p_n$ $\ldots$	NP [5]	$O(np_{min})$ [9, 10]	P	$O(n^5)$ *Lemma 21*
Canonical DL [4]	NP [4]	$O(n \sum p_j)$ [9, 10]	P	$O(n^5)$ *Lemma 22*
$d_1 + p_1 \leq d_2 + p_2 \leq \ldots \leq d_n + p_n$ $p_1 < p_2 < \ldots < p_n$	P	$O(n^3)$	P	$O(n^2)$ *Lemma 23*

Proof.

1) Assume that there exists an optimal schedule $\pi = (\pi_1, j, \pi_2, i, \pi_3)$, where job j is tardy and job i is on-time. For schedule $\pi' = (\pi_1, i, j, \pi_2, \pi_3)$, we have: $F(\pi') - F(\pi) \geq (T_j(\pi') - T_j(\pi)) + (T_i(\pi') - T_i(\pi)) = (p_i) + (0) > 0$. Therefore, we have a contradiction, since the schedule π' is better, and π is not optimal.

2) We consider an optimal schedule $\pi = (S, F)$, where all jobs $j \in F$ are tardy and all jobs $i \in S$ are on-time. Now we prove that all jobs $j \in F$ are processed according to an LPT order. Assume that there exists an optimal schedule $\pi = (\pi_1, j_1, j_2, \pi_2)$, where j_1 and j_2 are tardy and $p_{j_1} < p_{j_2}$. For schedule $\pi' = (\pi_1, j_2, j_1, \pi_2)$, we have $F(\pi') - F(\pi) = (T_{j_1}(\pi') - T_{j_1}(\pi)) + (T_{j_2}(\pi') - T_{j_2}(\pi)) \geq (p_{j_2}) - (\min\{p_{j_1}, T_{j_2}(\pi)\}) > 0$. Therefore, we have a contradiction and $\pi = (\pi_1, j_1, j_2, \pi_2)$ is not optimal.

3) We consider an optimal schedule $\pi = (S, F)$, where all jobs $j \in F$ are tardy and all jobs $i \in S$ are on-time. Now, we prove that all jobs $i \in S$ can be processed in an SPT order in an optimal schedule. For all jobs $i \in S$, we have $d_i \geq \sum_{k \in S} p_k$, otherwise, if $d_i < \sum_{k \in S} p_k$, then schedule $\pi' = (S \setminus \{i\}, i, F)$ is better (i.e., it has a larger value of total tardiness), and we have a contradiction. Therefore, jobs $i \in S$ can be processed in any order. $\square$

Define U_{max} as the maximal number of tardy jobs for the instance of the problem. Then $n - U_{max}$ is the minimal number of on-time jobs. We can compute the value U_{max} by means of Algorithm **A3** in $O(n \log n)$ time.

Let N_1 be the set of $n - U_{max} - 1$ jobs with minimal processing times, and N_2 be the set of $n - U_{max}$ jobs with minimal processing times. Define N_i as the set $N_1 \bigcup \{i\}$, if $i \notin N_1$, otherwise $N_i = N_2$.

Lemma 10 *If $d_i < \sum_{k \in N_i} p_k$, then there is no optimal schedule, where job i is on-time.*

Proof. We prove the lemma indirectly. Assume that there exists an optimal schedule $\pi = (S, F)$, where jobs $j \in F$ are tardy and jobs $i \in S$ are on-time. Let there be a job $i \in S$ with $d_i < \sum_{k \in N_i} p_k$. Then $d_i < \sum_{k \in N_i} p_k \leq \sum_{k \in S} p_k$, and schedule $\pi' = (S \setminus \{i\}, i, F)$ is better, i.e., $F(\pi') > F(\pi)$. $\square$

Lemma 11 *If job j is tardy in an optimal schedule, then all jobs $i \in N,\ p_i < p_j,\ d_i \leq d_j$ and all jobs $i \in N,\ p_i = p_j,\ d_i < d_j$ are tardy in this schedule.*

Proof. We can prove this lemma in a similar way as Lemma 7, i.e., if job $i \in N,\ p_i < p_j,\ d_i \leq d_j$ is on-time, then we can interchange jobs j and i. $\square$

Lemma 12 *If for job j, we have*

$$\left(\sum_{i=1}^{n} p_i - \sum_{k \in \{i \in N,\ p_i < p_j,\ d_i \leq d_j\} \bigcup \{i \in N,\ p_i = p_j,\ d_i < d_j\}} p_k \right) \leq d_j,$$

then there is no optimal schedule, where job j is tardy.

Proof. From Lemma 9 we know that all tardy jobs are processed in an LPT order. Then, by Lemma 11, Lemma 12 holds. $\square$

Lemma 13 *Let j be the first tardy job in an optimal schedule $\pi = (S, j, F)$. Then for each on-time job $i \in S$, we have $d_i \geq \max\{C_j(\pi) - p_j, d_j\}$.*

Proof. If for a job $i \in S$ we have $d_i < C_j(\pi) - p_j$, then job i is tardy in schedule $\pi' = (S \setminus \{i\}, i, j, F)$, and we have $F(\pi') > F(\pi)$. If $d_i < d_j$, then for schedule $\pi' = (S \setminus \{i\}, j, i, F)$, we have $F(\pi') > F(\pi)$. $\square$

Lemma 14 *Let j be a tardy job in an optimal schedule $\pi = (\pi_1, \pi_2, j, \pi_3)$, where all jobs from π_2 are tardy and $|\{\pi_2\}| = k > 0$. Then $T_j(\pi) \geq k \cdot p_j$.*

Proof. If $T_j(\pi) < k \cdot p_j$, then for schedule $\pi = (j, \pi_1, \pi_2, \pi_3)$, we have $F(\pi') > F(\pi)$. $\square$

Renumber the jobs according to the rule: $d_1 \leq d_2 \leq \ldots \leq d_n$, if $d_i = d_{i+1}$, then $p_i \leq p_{i+1}$. Let j be the job with maximal processing time $j := argmax\{d_i : p_i = \max_{k \in N} p_k\}$.

Lemma 15 *There is an optimal schedule, where the first tardy jobs is from the set $\{1, 2, \ldots, j\}$. If $d_j < d_{j+1}$, then in all optimal schedules the first tardy jobs is from the set $\{1, 2, \ldots, j\}$.*

Proof. We prove this lemma indirectly. Let there be an optimal schedule $\pi = (\pi_1, i, \pi_2)$, where $i > j$ and job i is the first tardy job. Then job j is on-time in this optimal schedule, since all tardy jobs are processed in an LPT order in an optimal schedule. Now assume $\pi = (j, \pi_1, i, \pi_2)$. For schedule $\pi' = (\pi_1, i, j, \pi_2)$, we have $F(\pi') = F(\pi)$ if $d_j = d_i$, and $F(\pi') > F(\pi)$ if $d_j < d_i$. $\square$

Denote $M_j = \{i \in N, p_i > p_j\}$ and $O_j = \{i \in N, i > j\}$.

Lemma 16 *If a job j is the first tardy job in an optimal schedule π, then for each job $i \in M_j$ we have $\sum_{k \in M_j} p_k \leq d_i$.*

The proof is obvious since all tardy jobs are processed in an LPT order in an optimal schedule, i.e. all jobs $i \in M_j$ are not tardy.

Lemma 17 *If a job j is the first tardy job in an optimal schedule π, then $\sum_{k \in O_j} p_k + p_j > d_j$.*

We can use Lemmas 10 – 17 to compute an optimal schedule faster.

2.4 An exact algorithm for the special case $d_{max} - d_{min} \leq 1$

Define $z = \lfloor d_{max} \rfloor$. The following Algorithm **T1** constructs an optimal solution for the special case under consideration.

Algorithm T1.

Step 1. $S := N,\ F := \emptyset,\ P_f := 0$;

Step 2. If there is a job $j \in S$ with $\sum_{i=1}^{n} p_i - P_f - p_j \geq z + 1$, then GOTO Step 3, otherwise GOTO Step 5;

Step 3. We select a job $j^*(S) \in S$ with minimal processing time p_j. If there are several such jobs, then among these jobs, we select one with minimal due date d_j:

$$j^*(S) = \arg\min_{j \in S}\{d_j : p_j = \min_{i \in S} p_i\}$$

Step 4. $S := S \setminus \{j^*(S)\},\ F := F \bigcup \{j^*(S)\},\ P_f := P_f + p_{j^*(S)}$, GOTO Step 2;

Step 5. Now we can choose only two tardy jobs. For each pair of two jobs $j_1,\ j_2 \in S$, we consider two schedules $\pi' = (\pi_1, j_1, j_2, \pi_2)$ and $\pi'' = (\pi_1, j_2, j_1, \pi_2)$, where $\{\pi_1\} = S \setminus \{j_1, j_2\},\ \{\pi_2\} = F$, and all jobs in F are processed in an LPT order. Then we select the best schedule among $2\frac{\lceil S \rceil \cdot (|S|-1)}{2}$ schedules considered.

Lemma 18 *If $\sum_{i=1}^{n} p_i - p_{j^*(N)} \geq z + 1$, then there is an optimal schedule, where job $j^*(N)$ is the last tardy job.*

Proof. Let $j^* = j^*(N)$. Moreover, assume that there exists an optimal schedule $\pi = (\pi_1, j^*, \pi_2, \alpha, i)$. If $p_{j^*} = p_i, d_{j^*} \leq d_i$, then schedule $\pi' = (\pi_1, i, \pi_2, \alpha, j^*)$ is also optimal.

If $p_{j^*} < p_i$, then job j^* is not tardy in schedule π. We consider schedule $\pi' = (\pi_1, i, \pi_2, \alpha, j^*)$. We have:

a) If $C_i(\pi') \leq d_i$, then $T_i(\pi') - T_i(\pi) = -\sum_{k=1}^{n} p_k + d_i$, $T_{j^*}(\pi') - T_{j^*}(\pi) = \sum_{k=1}^{n} p_k - d_{j^*}$, $T_\alpha(\pi') - T_\alpha(\pi) \geq 1$ since $p_i \geq p_{j^*} + 1$ ($p_j \in Z$ for all $j \in N$).

Then $F(\pi') - F(\pi) \geq (T_{j^*}(\pi') - T_{j^*}(\pi)) + (T_i(\pi') - T_i(\pi)) + (T_\alpha(\pi') - T_\alpha(\pi)) \geq 0$ since $d_{max} - d_{min} \leq 1$.

b) If $C_i(\pi') > d_i$, then $T_i(\pi') - T_i(\pi) = -\sum_{k\in\pi_2\bigcup\{\alpha,j^*\}} p_k$, $T_{j^*}(\pi') - T_{j^*}(\pi) \geq \sum_{k\in\pi_2\bigcup\{\alpha,j^*\}} p_k - 1$ since $d_{max} - d_{min} \leq 1$.

Moreover, $T_\alpha(\pi') - T_\alpha(\pi) \geq 1$ since $p_i \geq p_{j^*} + 1$ ($p_j \in Z$ for all $j \in N$). Then $F(\pi') - F(\pi) \geq 0$.

□

Lemma 19 *Algorithm* **T1** *constructs an optimal schedule in* $O(n^2)$ *time.*

Proof. The optimality of Steps 3-4 results from Lemma 18. The time complexity of Step 5 is $O(n^2)$. □

2.5 An exact algorithm for the special case 'Canonical LG' [5]

We consider the following special case (see [5]):

$$\begin{cases} p_1 > p_2 > \ldots > p_{2n+1}, \\ d_1 < d_2 < \ldots < d_{2n+1}, \\ d_{2n+1} - d_1 < p_{2n+1}, \\ p_{2n+1} = M = n^3 b, \\ p_{2n} = p_{2n+1} + b, \\ p_{2i} = p_{2i+2} + b,\ i = n-1, \ldots, 1, \\ p_{2i-1} = p_{2i} + \delta_i,\ i = n, \ldots, 1, \\ d_{2n+1} = \sum_{i=1}^{n} p_{2i} + p_{2n+1} + \frac{1}{2}\delta, \\ d_{2n} = d_{2n+1} - \delta, \\ d_{2i} = d_{2i+2} - (n-i)b + \delta,\ i = n-1, \ldots, 1, \\ d_{2i-1} = d_{2i} - (n-i)\delta_i - \varepsilon\delta_i,\ i = n, \ldots, 1, \end{cases} \qquad (LG)$$

where $\delta_1, \delta_2, \ldots, \delta_n \in Z$, $\delta = \sum_{i=1}^{n} \delta_i$, $b = n^2\delta$, $0 < \varepsilon < \frac{\min_i \delta_i}{\max_i \delta_i}$.
We define

$N = \{1, 2, \ldots, 2n, 2n+1\} = \{V_1, V_2, V_3, V_4, \ldots, V_{2i-1}, V_{2i}, \ldots, V_{2n-1}, V_{2n}, V_{2n+1}\}$.

Lemma 20 *[5] For case LG, there are either* n *or* $n+1$ *tardy jobs in each schedule.*

Lemma 21 *For case LG, we can construct an optimal schedule with* $O(n^5)$ *operations.*

Proof.

Without loss of generality, we consider only the case when $n \geq 2$. The proof is as follows. First, we can show that in all optimal schedules, job V_{2n+1} is processed in the last position (see Step 0). Next, one can show that in the last positions of all optimal schedules, there are processed $(\lfloor \frac{n}{2} \rfloor - 1)$ tardy jobs with minimal processing times (Steps 1 and 2). Then we can prove that the first $n - (\lfloor \frac{n}{2} \rfloor + 2)$ tardy jobs in all optimal schedules are the jobs with maximal processing times (and minimal due dates)(see Step 3). For this reason, we know at least $n - 3$ tardy jobs in an optimal schedule. Therefore, we must choose at most 4 further tardy jobs since for this special case, there are either n or $n + 1$ tardy jobs in all schedules. Then we argue that $O(n^4)$ schedules have to be considered and $O(n^5)$ time is required (see Step 4).

0. First, we show that there is an optimal schedule, where job V_{2n+1} is processed in the last position of the schedule. We prove this indirectly. Assume that there is an optimal schedule π, where another job is processed in the last position.

0.1. Assume that there is an optimal schedule of the form $\pi = (SPT, LPT) = (\pi_1, V_{2n+1}, \pi_2, V_{2j})$. Consider schedule $\pi = (\pi_1, V_{2j}, \pi_2, V_{2n+1})$. We have $p_{2j} - p_{2n+1} = p_{2j} - p_{2n} + b = (n - j + 1)b$ and

$d_{2n+1} - d_{2j} = \delta + d_{2n} - d_{2j} = \delta + (n - (n-1))b + (n - (n-2))b + \ldots + (n-j)b - (n-j)\delta = b + 2b + \ldots + (n-j)b - (n-j-1)\delta = \frac{(n-j)(n-j+1)}{2}b - (n-j-1)\delta$.

Then $F(\pi') - F(\pi) \geq (n-1)(p_{2j} - p_{2n+1}) - (d_{2n+1} - d_{2j}) = (n-1)(n-j+1)b - \frac{(n-j)(n-j+1)}{2}b + (n-j-1)\delta$. We have $n - 1 - \frac{n-j}{2} = \frac{2n-2-n+j}{2} = \frac{n-2+j}{2} \geq \frac{1}{2}$ for $n > 1$ and $j \geq 1$. Thus, $F(\pi') - F(\pi) > 0$. We have a contradiction and schedule π is not optimal.

0.2. Assume that there is an optimal schedule of the form $\pi = (SPT, LPT) = (\pi_1, V_{2n+1}, \pi_2, V_{2j-1})$. Consider schedule $\pi = (\pi_1, V_{2j-1}, \pi_2, V_{2n+1})$. We have $p_{2j-1} - p_{2n+1} = p_{2j} - p_{2n} + b + \delta_j > (n - j + 1)b$ and

$d_{2n+1} - d_{2j-1} = \delta + d_{2n} - d_{2j} + (n-j)\delta_j + \varepsilon\delta_j = \frac{(n-j)(n-j+1)}{2}b - (n - j - 1)\delta + (n-j)\delta_j + \varepsilon\delta_j$.

We have $F(\pi') - F(\pi) > 0$ for $n > 1$ and $j \geq 1$. We have a contradiction and schedule π is not optimal.

1. Next, we show that in the last positions of all optimal schedules, there are processed the jobs with minimal processing times. We prove this indirectly. Let we have an optimal schedule, where this rule does not hold. We go from the end of the schedule to the beginning. In the schedule, let position k be the *first position 'from the end'*, where job V_{2i} (or V_{2i-1}) must be processed, but this position is occupied by job V_{2j} (or V_{2j-1}). Assume that $n > 1$ and $j < i$, i.e., there is a schedule $\pi = (\pi_1, V_{2i}, \pi_2, V_{2j}, \pi_3, V_{2n+1})$, $\pi_3 = (V_{2(i+1)-1}, V_{2(i+1)}, \ldots, V_{2n-1}, V_{2n})$, where job V_{2i} is on-time. It is known that in an optimal schedule, all jobs are processed according to an LPT order (see Lemma 9).

   First, we consider the case when k is an even number. Then $i = n - \frac{k-2}{2} = n - \frac{k}{2} + 1$. In schedule π, there are at least $n - k$ tardy jobs before job V_{2j}. Therefore, we have $j > \frac{n-k}{2}$.

   We have to consider the following cases for the two V-jobs.

1.1. Case of jobs V_{2i} and V_{2j}. Assume that we have $\pi = (\pi_1, V_{2i}, \pi_2, V_{2j}, \pi_3, V_{2n+1})$, $\pi_3 = (V_{2(i+1)-1}, V_{2(i+1)}, \ldots, V_{2n-1}, V_{2n})$, where job V_{2i} is on-time. Let us consider schedule $\pi' = (\pi_1, V_{2j}, \pi_2, V_{2i}, \pi_3, V_{2n+1})$.

   We have $p_{2j} - p_{2i} = (i-j)b$, $d_{2i} - d_{2j} = (n-(i-1))b + (n-(i-2))b + \ldots + (n-j)b - (i-j)\delta = (n-i)(i-j)b + \sum_{i=1}^{i-j} b - (i-j)\delta = (n-i)(i-j)b + \frac{(i-j)(i-j+1)}{2}b - (i-j)\delta = (n-i+\frac{i-j+1}{2})(i-j)b - (i-j)\delta$. We compute the value k for which we get $F(\pi') > F(\pi)$. We must have: $(n-k)(i-j)b \geq (n-i+\frac{i-j+1}{2})(i-j)b \Rightarrow 2n-2k \geq 2n-2i+i-j+1 \Rightarrow 2k \leq i+j-1 < n-\frac{k}{2}+1+\frac{n-k}{2}-1 \Rightarrow 2k < \frac{3}{2}n-k \Rightarrow k < \frac{n}{2}$.

1.2. Case of jobs V_{2i} and V_{2j-1}. Assume that we have $\pi = (\pi_1, V_{2i}, \pi_2, V_{2j-1}, \pi_3, V_{2n+1})$, where job V_{2i} is on-time. Let us consider schedule $\pi' = (\pi_1, V_{2j-1}, \pi_2, V_{2i}, \pi_3, V_{2n+1})$.

   We have $p_{2j-1} - p_{2i} = (i-j)b + \delta_j$, $d_{2i} - d_{2j-1} = (n-(i-1))b + (n-(i-2))b + \ldots + (n-j)b - (i-j)\delta + (n-j)\delta_j + \varepsilon\delta_j = (n-i)(i-j)b + \sum_{i=1}^{i-j} b - (i-j)\delta + (n-j)\delta_j + \varepsilon\delta_j = (n-i)(i-j)b + \frac{(i-j)(i-j+1)}{2}b - (i-j)\delta = (n-i+\frac{i-j+1}{2})(i-j)b - (i-j)\delta + (n-j)\delta_j + \varepsilon\delta_j$. Analogously, we compute the value k for which we have $F(\pi') > F(\pi)$. We must have: $(n-k)(i-j)b > (n-i+\frac{i-j+1}{2})(i-j)b$. Thus, we have $k < \frac{n}{2}$.

1.3. We try to put only job V_{2i} (not V_{2i-1}) at this position, since V_{2i} stays at this position, if V_{2i} is tardy, i.e., if job V_{2i} is tardy in schedule π,

then this job occupies position k 'from the end'.

2. Now, let us consider the case when k is an odd number. Then $i = n - \frac{k-3}{2}$. We try to put job V_{2i-1} (or V_{2i}) at this position. We have to consider the following cases for the two V-jobs.

2.1. Case of jobs V_{2i-1} and V_{2j}. We have $\pi = (\pi_1, V_{2i-1}, \pi_2, V_{2j}, \pi_3, V_{2n+1})$ with $\pi_3 = (V_{2i}, V_{2(i+1)-1}), V_{2(i+1)}, \ldots, V_{2n-1}, V_{2n})$, where job V_{2i-1} is on-time. Let us consider schedule $\pi' = (\pi_1, V_{2j}, \pi_2, V_{2i-1}, \pi_3, V_{2n+1})$.

We have $p_{2j} - p_{2i-1} = (i-j)b - \delta_i$, $d_{2i-1} - d_{2j} = (n - i + \frac{i-j+1}{2})(i-j)b - (i-j)\delta - (n-i)\delta_i - \varepsilon\delta_i$. We compute the value k for which we get $F(\pi') > F(\pi)$. We must have: $(n-k)(i-j)b > (n-i+\frac{i-j+1}{2})(i-j)b \Rightarrow 2n-2k > 2n-2j+i-j+1 \Rightarrow 2k < i+j-1 < n-\frac{k}{2}+\frac{3}{2}+\frac{n-k}{2}-1 \Rightarrow 3k < \frac{3}{2}n + \frac{1}{2} \Rightarrow k < \frac{n}{2} + \frac{1}{6}$.

2.2. Case of jobs V_{2i-1} and V_{2j-1}. We have $\pi = (\pi_1, V_{2i-1}, \pi_2, V_{2j-1}, \pi_3, V_{2n+1})$, where job V_{2i-1} is on-time. Let us consider schedule $\pi' = (\pi_1, V_{2j-1}, \pi_2, V_{2i-1}, \pi_3, V_{2n+1})$.

We have $p_{2j-1} - p_{2i-1} = (i-j)b - \delta_i + \delta_j$, $d_{2i-1} - d_{2j-1} = (n - i + \frac{i-j+1}{2})(i-j)b - (i-j)\delta - (n-i)\delta_i - \varepsilon\delta_i + (n-j)\delta_j + \varepsilon\delta_j$. We consider this case in an analogous way. We get $k < \frac{n}{2} + \frac{1}{6}$.

2.3. The cases (V_{2i} and V_{2j}) and (V_{2i} and V_{2j-1}) can be treated in an analogous way. For example, we have the schedule $\pi = (\pi_1, V_{2i}, \pi_2, V_{2j}, \pi_3, V_{2n+1})$ with $\pi_3 = (V_{2i-1}, V_{2(i+1)-1}), V_{2(i+1)}, \ldots, V_{2n-1}, V_{2n})$, and we consider the schedule $\pi' = (\pi_1, V_{2j}, \pi_2, V_{2i-1}, \pi_3, V_{2n+1})$.

3 We show that the first tardy jobs in all optimal schedules are the jobs with maximal processing times (and minimal due dates). We prove this indirectly in an analogous way. Now we go from the middle of the schedule and consider the last n positions in the schedule. Let position k in the schedule be the first position 'from the middle' (but the numbering is 'from the end'), where job V_{2j} (or V_{2j-1}) must be processed, but there stands job V_{2i} (or V_{2i-1}). Assume $n > 1$ and $j < i$, i.e., there is a schedule $\pi = (\pi_1, V_{2j}, \pi_{2.1}, \pi_{2.2}, V_{2i}, \pi_3, V_{2n+1})$, $\pi_{2.2} = (V_1, V_2, \ldots, V_{2(j-1)-1}, V_{2(j-1)})$, where job V_{2j} is on-time. It is known that in an optimal schedule, all jobs are processed according to an LPT order (see Lemma 9). Assume that $n-k$ is an odd number (the case when $n-k$ is even can be analogously considered). Moreover, assume that $j = \frac{n-k-1}{2} + 1 = \frac{n-k}{2} + \frac{1}{2}$. We want to

compute the value k for which we get $F(\pi') > F(\pi)$, where $\pi = (\pi_1, V_{2i}, \pi_{2.1}, \pi_{2.2}, V_{2j}, \pi_3, V_{2n+1})$.

3.1. Case of jobs V_{2i} and V_{2j}. We have the schedule $\pi = (\pi_1, V_{2j}, \pi_{2.1}, \pi_{2.2}, V_{2i}, \pi_3, V_{2n+1})$ with $\pi_{2.2} = (V_1, V_2, \ldots, V_{2(j-1)-1}, V_{2(j-1)})$, where job V_{2j} is on-time. Let us consider the schedule $\pi' = (\pi_1, V_{2i}, \pi_{2.1}, \pi_{2.2}, V_{2j}, \pi_3, V_{2n+1})$. It is obvious that $i \leq n - \frac{k-2}{2}$, while all jobs in π are processed in an LPT order. We want to get $(n-k+1)(p_{2j} - p_{2i}) < (d_{2i} - d_{2j})$. Then $(n-k+1) < \frac{2n-i-j+1}{2} \Rightarrow 2k-1 > i+j \Rightarrow 2k-1 > \frac{n-k}{2} + \frac{1}{2} + n - \frac{k}{2} + 1 \Rightarrow 3k > \frac{3}{2}n + \frac{5}{2} \Rightarrow k > \frac{n}{2} + \frac{5}{6}$.

3.2. Case of jobs V_{2i-1} and V_{2j}. We have the schedule $\pi = (\pi_1, V_{2j}, \pi_{2.1}, \pi_{2.2}, V_{2i-1}, \pi_3, V_{2n+1})$, where job V_{2j} is on-time. Let us consider the schedule $\pi' = (\pi_1, V_{2i-1}, \pi_{2.1}, \pi_{2.2}, V_{2j}, \pi_3, V_{2n+1})$. It is obvious that $i \leq n - \frac{k-3}{2}$, while all jobs in π are processed in an LPT order. We want to get $2k-1 > \frac{n-k}{2} + \frac{1}{2} + n - \frac{k}{2} + \frac{3}{2} \Rightarrow 3k > \frac{3}{2}n + 3 \Rightarrow k > \frac{n}{2} + 1$.

3.3. Case of jobs V_{2i} and V_{2j-1}. We have the schedule $\pi = (\pi_1, V_{2j-1}, \pi_{2.1}, \pi_{2.2}, V_{2i}, \pi_3, V_{2n+1})$, where job V_{2j-1} is on-time. Let us consider the schedule $\pi' = (\pi_1, V_{2i}, \pi_{2.1}, \pi_{2.2}, V_{2j-1}, \pi_3, V_{2n+1})$. We assume $j = \frac{n-k}{2} + \frac{1}{2}$. We have $i \leq n - \frac{k-2}{2}$, since all jobs in π are processed in an LPT order. We want to get $2k-1 > \frac{n-k}{2} + \frac{1}{2} + n - \frac{k}{2} + 1 \Rightarrow 3k > \frac{3}{2}n + \frac{5}{2} \Rightarrow k > \frac{n}{2} + \frac{5}{6}$.

3.4. Case of jobs V_{2i-1} and V_{2j-1}. We have the schedule $\pi = (\pi_1, V_{2j-1}, \pi_{2.1}, \pi_{2.2}, V_{2i-1}, \pi_3, V_{2n+1})$, where job V_{2j-1} is on-time. Let us consider the schedule $\pi' = (\pi_1, V_{2i-1}, \pi_{2.1}, \pi_{2.2}, V_{2j-1}, \pi_3, V_{2n+1})$. We assume $j = \frac{n-k}{2} + \frac{1}{2}$. We have $i \leq n - \frac{k-3}{2}$, since all jobs in π are processed in an LPT order. We want to get $2k-1 > \frac{n-k}{2} + \frac{1}{2} + n - \frac{k}{2} + \frac{3}{2} \Rightarrow 3k > \frac{3}{2}n + 3 \Rightarrow k > \frac{n}{2} + 1$.

4. Now, we know that in all optimal schedules, $n - (\lfloor \frac{n}{2} \rfloor + 2)$ tardy jobs are the jobs with maximal processing times and there are $(\lfloor \frac{n}{2} \rfloor - 1)$ tardy jobs with minimal processing times. For this reason, we know at least $n-3$ tardy jobs. Therefore, we must choose at most 4 further tardy jobs, since for this special case, there are either n or $n+1$ tardy jobs in all schedules. Thus, we have to consider at most $O(n^4)$ schedules. The time complexity of this part is $O(n^5)$.

□

2.6 An exact algorithm for the special case 'Canonical DL' [4]

For this special case we have $3\bar{n}+1$ jobs:

$$N = \{V_1, V_2, \ldots, V_{2i-1}, V_{2i}, \ldots, V_{2\bar{n}-1}, V_{2\bar{n}}, W_1, W_2, \ldots, W_{\bar{n}+1}\},$$

where $b_1 \geq b_2 \geq \ldots \geq b_{2\bar{n}},\ b_i \in Z,\ i = 1, 2, \ldots, 2\bar{n},\ \delta = \frac{1}{2}\sum_{i=1}^{\bar{n}}(b_{2i-1} - b_{2i})$, and $b = (4\bar{n}+1)\delta$.

Moreover,

$$\begin{aligned}
a_1 &= b_1 + (9\bar{n}^2 + 3\bar{n})\delta + 5\bar{n}(b_1 - b_{2\bar{n}}),\\
a_2 &= b_2 + (9\bar{n}^2 + 3\bar{n})\delta + 5\bar{n}(b_1 - b_{2\bar{n}}),\\
&\ldots,\\
a_{2i-1} &= b_{2i-1} + (9\bar{n}^2 + 3\bar{n} - i + 1)\delta + 5\bar{n}(b_1 - b_{2\bar{n}}),\\
a_{2i} &= b_{2i} + (9\bar{n}^2 + 3\bar{n} - i + 1)\delta + 5\bar{n}(b_1 - b_{2\bar{n}}),\\
&\ldots,\\
a_{2n-1} &= b_{2n-1} + (9\bar{n}^2 + 2\bar{n} + 1)\delta + 5\bar{n}(b_1 - b_{2\bar{n}}),\\
a_{2n} &= b_{2n} + (9\bar{n}^2 + 2\bar{n} + 1)\delta + 5\bar{n}(b_1 - b_{2\bar{n}}).
\end{aligned}$$

$$\begin{aligned}
p_{V_i} &= a_i, \qquad 1 \leq i \leq 2\bar{n};\\
p_{W_i} &= b, \qquad 1 \leq i \leq \bar{n}+1;\\
d_{V_i} &= \begin{cases} (j-1)b + \delta + (a_2 + a_4 + \ldots + a_{2j}) & \text{if } i = 2j-1\\ d_{V_{2j-1}} + 2(\bar{n} - j + 1)(a_{2j-1} - a_{2j}) & \text{if } i = 2j \end{cases}\\
d_{W_i} &= \begin{cases} ib + (a_2 + a_4 + \ldots + a_{2i}) & 1 \leq i \leq \bar{n}\\ d_{W_{\bar{n}}} + \delta + b & i = \bar{n}+1 \end{cases}
\end{aligned}$$

Lemma 22 *For the special case 'Canonical DL', we can construct an optimal schedule in $O(n^5)$ time.*

Proof.

Without loss of generality, we consider only the case when $\bar{n} \geq 3$.

The proof can be sketched as follows. We consider properties of 'stable' schedules (see Step 1). A schedule $\pi = (S, F)$ is stable, if for each on-time job $j \in S$ in schedule π, we have $d_j \geq \sum_{i \in S} p_i$, where S denotes the set of on-time jobs. It is obvious that all optimal schedules are stable. First, we investigate *the maximal number* k of on-time jobs from set $\bar{V} =$

$\{V_1, V_2, \ldots, V_{2i-1}, V_{2i}, \ldots, V_{2\bar{n}-1}, V_{2\bar{n}}\}$ in a stable schedule. Then we consider the following property of an optimal schedule: Let i be the last tardy job from set $\bar{V}$ in an optimal schedule $\pi^* = (\pi_1, i, \pi_2)$. Assume that in schedule π^*, there are k on-time jobs. Then, before job i, there are $2\bar{n} - k - 1$ tardy jobs from set $\bar{V}$ in an optimal schedule. Thus, $i \geq 2\bar{n} - k$ since all tardy jobs are processed according to an LPT order. We must have $F(\pi^*) > F(\pi = (i, \pi_1, \pi_2))$ since π^* is optimal. Summarizing, for each stable (and optimal) schedule, we must have $k = \lceil \frac{2}{3}\bar{n} \rceil$ or $\lfloor \frac{2}{3}\bar{n} \rfloor$.

Additionally, we prove that all jobs $W_j,\ j = 1, \ldots, n+1$, are processed at the end of the schedule (see Step 2).

Thus, we must select k on-time jobs from the set $\bar{V}$ and process them at the beginning of the schedule. We investigate the minimal number i of a job V_i, which can be on-time in a stable schedule ($d_i \geq \sum_{l \in S} p_l,\ |S| = k$). At the end, we consider each situation ($k = \lceil \frac{2}{3}\bar{n} \rceil$ or $\lfloor \frac{2}{3}\bar{n} \rfloor$, i is even or odd) separately and get similar results, namely that we have to consider $O(n^4)$ schedules to find an optimal one (see Step 3).

0. It is obvious that in each schedule, at least $\bar{n}$ jobs from set $\{V_1, V_2, \ldots, V_{2i-1}, V_{2i}, \ldots, V_{2\bar{n}-1}, V_{2\bar{n}}\}$ are tardy since $(p_{V_{n+1}} + p_{V_{n+2}} + \ldots + p_{V_{2n}}) + p_{V_i} > d_{max} = d_{W_{2n+1}}$ for all $i = 1, 2, \ldots, n$. In the proof, we consider only schedules of the form $\pi = (SPT, LPT)$. Otherwise, we can re-order the jobs accordingly without loss of optimality.

1. Now we consider the properties of 'stable' schedules. A schedule $\pi = (S, F)$ is stable, if for each on-time job $j \in S$ in schedule π, we have $d_j \geq \sum_{i \in S} p_i$, where S denotes the set of on-time jobs. It is obvious that all optimal schedules are stable.

Now we investigate *the maximal number* of on-time jobs from set $\bar{V} = \{V_1, V_2, \ldots, V_{2i-1}, V_{2i}, \ldots, V_{2\bar{n}-1}, V_{2\bar{n}}\}$. The largest set S (with a maximal number of elements) consists of jobs $V_j, V_{j+1}, \ldots, V_{2\bar{n}}$. Then $P_S = \sum_{i=j}^{2\bar{n}} p_{V_i} > (2\bar{n} - j + 1)a_{2\bar{n}}$.

a) Let j be even, $d_{min}(i \in S) = d_j = a_2 + a_4 + \ldots + a_j + b\frac{(j-2)}{2} + \delta + 2(\bar{n} - \frac{j}{2} + 1)(a_j - a_{j-1}) < (\frac{j}{2} + 1)a_{2\bar{n}}$. Then for a stable schedule, we must have $P_S \leq d_j$. Then we get $\frac{j}{2} + 1 > 2\bar{n} - j + 1 \Rightarrow j + 2 > 4\bar{n} - 2j + 2 \Rightarrow j > \frac{4\bar{n}}{3}$. Let $k = |S|$, i.e., k is the maximal number of on-time jobs in a stable (optimal, too) schedule. Then $k \leq 2\bar{n} - j + 1 < 2\bar{n} - (\frac{4\bar{n}}{3}) + 1 \Rightarrow 3k < 6\bar{n} - 4\bar{n} + 3 \Rightarrow k < \frac{2}{3}\bar{n} + 1$.

b) Let j be odd, $d_{min}(i \in S) = d_j = a_2 + a_4 + \ldots + a_{j-1} + a_{j+1} + b\frac{(j-1)}{2} + \delta <$

$(\frac{j+1}{2}+1)a_{2\bar{n}}$. Then for a stable schedule, we must have $P_S \leq d_j$. Then we get $\frac{j+1}{2}+1 > 2\bar{n}-j+1, \Rightarrow j+3 > 4\bar{n}-2j+2 \Rightarrow j > \frac{4\bar{n}-1}{3}$. Let $k = |S|$, i.e., k is the maximal number of on-time jobs in a stable (optimal, too) schedule. Then $k \leq 2\bar{n}-j+1 < 2n-(\frac{4\bar{n}-1}{3})+1 \Rightarrow 3k < 6\bar{n}-4\bar{n}+1+3 \Rightarrow k < \frac{2}{3}\bar{n}+\frac{4}{3}$.

Thus, we have $k < \frac{2}{3}\bar{n}+1$, i.e., $k \leq \lceil \frac{2}{3}\bar{n} \rceil$.

2. Now, we show that in all optimal schedules, all jobs $W_j,\ j = 1, \ldots, n+1$ are processed at the end of the schedule. We prove this indirectly. Assume that there is an optimal schedule of the form $\pi = (SPT, LPT) = (W_j, \pi_1, V_i, \pi_2)$, where job W_j is on-time and V_i is the last tardy job from set $\bar{V}$. Then, before job V_i, there are at least $2\bar{n} - \lceil \frac{2}{3}\bar{n} \rceil - 1$ tardy jobs from set $\bar{V}$ and $i \geq 2\bar{n} - \lceil \frac{2}{3}\bar{n} \rceil$ since all tardy jobs are processed according to an LPT order. Let us consider schedule $\pi' = (V_i, \pi_1, W_j, \pi_2)$. We have $F(\pi') - F(\pi) \geq (2\bar{n} - \lceil \frac{2}{3}\bar{n} \rceil - 1)(p_{V_i} - p_{W_j}) - (d_{W_j} - d_{V_i})$ and $d_{W_j} - d_{V_i} \leq d_{W_{\bar{n}+1}} - d_{V_i}$.

a) Let i be even. Then $d_{V_i} = (a_2 + a_4 + \ldots + a_i) + 2(\bar{n} - \frac{i}{2} + 1)(a_{i-1} - a_i) + \delta + (\frac{i}{2} - 1)b$, $d_{W_{\bar{n}+1}} - d_{V_i} < (\bar{n}+1)b + (a_{i+2} + \ldots + a_{2\bar{n}}) < (\bar{n}+1)b + \frac{2\bar{n}-i}{2}a_i \leq (\bar{n}+1)b + \frac{2\bar{n}-2\bar{n}+\lceil \frac{2}{3}\bar{n} \rceil}{2}a_i = (\bar{n}+1)b + \frac{1}{2}\lceil \frac{2}{3}\bar{n} \rceil a_i$.

We have $F(\pi') - F(\pi) > (2\bar{n} - \lceil \frac{2}{3}\bar{n} \rceil - 1)(a_i - b) - ((\bar{n}+1)b + (\frac{1}{2}\lceil \frac{2\bar{n}}{3} \rceil)a_i) > (2\bar{n} - \lceil \frac{2\bar{n}}{3} \rceil - 1 - \frac{1}{2}\lceil \frac{2\bar{n}}{3} \rceil)a_i - (3\bar{n})b > 0$ for $\bar{n} \geq 3$.

b) For an odd number i, the proof is analogous. We have $d_{W_{\bar{n}+1}} - d_{V_i} < (\bar{n}+1)b + (a_{i+1} + \ldots + a_{2\bar{n}}) < (\bar{n}+1)b + \frac{2\bar{n}-(i-1)}{2}a_i \leq (\bar{n}+1)b + \frac{2\bar{n}-2\bar{n}+\lceil \frac{2}{3}\bar{n} \rceil + 1}{2}a_i = (\bar{n}+1)b + (\frac{1}{2}\lceil \frac{2}{3}\bar{n} \rceil + \frac{1}{2})a_i$, and thus $F(\pi') - F(\pi) > 0$ for $n \geq 3$.

Now we know that in all optimal schedules, all jobs $W_j,\ j = 1, \ldots, n+1$ are processed at the end of the schedule.

3. For an optimal schedule, we also must have the following property: Let i be the last tardy job from set $\{V_1, V_2, \ldots, V_{2j-1}, V_{2j}, \ldots, V_{2\bar{n}-1}, V_{2\bar{n}}\}$ in an optimal schedule $\pi^* = (\pi_1, i, \pi_2)$. Assume that in schedule π^*, there are k on-time jobs. Then, before job i, there are $2\bar{n} - k - 1$ tardy jobs. Thus, $i \geq 2\bar{n} - k$ since all tardy jobs are processed according to an LPT order. We must have $F(\pi^*) > F(\pi = (i, \pi_1, \pi_2))$ since π^* is optimal, i.e., we must have $C_i(\pi^*) - d_i \geq (2\bar{n} - k - 1)p_i \Rightarrow (a_1 + a_3 + \ldots + a_{2\bar{n}-1}) + (a_2 + a_4 + \ldots + a_{2\bar{n}}) - d_i \geq (2\bar{n} - k - 1)a_i$.

a) Let i be even. Then $d_i = (a_2 + a_4 + \ldots + a_i) + 2(\bar{n} - \frac{i}{2} + 1)(a_{i-1} - a_i) + \delta + (\frac{i}{2} - 1)b$, $C_i(\pi^*) - d_i < (a_1 + a_3 + \ldots + a_{2\bar{n}-1}) + (a_{i+2} + a_{i+4} + \ldots + a_{2\bar{n}}) <$

$(2\bar{n} - \frac{i}{2} + 1)a_i$. We must have $(2\bar{n} - \frac{i}{2} + 1)a_i > (2\bar{n} - k - 1)a_i \Rightarrow \frac{i}{2} - 1 < k + 1 \Rightarrow i < 2k + 4$. Thus, $2\bar{n} - k \leq i < 2k + 4 \Rightarrow 2\bar{n} - 4 < 3k \Rightarrow k > \frac{2}{3}\bar{n} - \frac{4}{3}$.

b) Let i be odd. Then $d_i = (a_2 + a_4 + \ldots + a_{i+1}) + \delta + (\frac{i+1}{2} - 1)b$, $C_i(\pi^*) - d_i < (a_1 + a_3 + \ldots + a_{2\bar{n}-1}) + (a_{i+3} + a_{i+5} + \ldots + a_{2\bar{n}}) < (2\bar{n} - \frac{i+1}{2} + 1)a_i$. We must have $(2\bar{n} - \frac{i+1}{2} + 1)a_i > (2\bar{n} - k - 1)a_i \Rightarrow \frac{i+1}{2} - 1 < k + 1 \Rightarrow i < 2k + 3$. Thus, $2\bar{n} - k \leq i < 2k + 3 \Rightarrow 2\bar{n} - 3 < 3k \Rightarrow k > \frac{2}{3}\bar{n} - 1$.

4. For each stable schedule, we have $\frac{2}{3}\bar{n} - 1 < k < \frac{2}{3}\bar{n} + 1$ and therefore, $k = \lceil \frac{2}{3}\bar{n} \rceil$ or $\lfloor \frac{2}{3}\bar{n} \rfloor$. Thus, we must select k on-time jobs and process them at the beginning of the schedule. Now, we investigate the minimal number i of a job V_i, which can be on-time in a stable schedule ($d_i \geq \sum_{k \in S} p_k$).

a) Let $k = \lceil \frac{2}{3}\bar{n} \rceil$. In a stable schedule, we must have $d_i > \lceil \frac{2}{3}\bar{n} \rceil a_{2\bar{n}}$.

If i is even, then $d_i = (a_2 + a_4 + \ldots + a_i) + \frac{i-2}{2}b + \delta + 2(\bar{n} - \frac{i}{2} + 1)(a_{i-1} - a_i) < (\frac{i}{2} + 1)a_{2\bar{n}}$. We must have $(\frac{i}{2} + 1)a_{2\bar{n}} > \lceil \frac{2}{3}\bar{n} \rceil a_{2\bar{n}}$. We get $i > 2\lceil \frac{2}{3}\bar{n} \rceil - 2 \geq \lceil \frac{4}{3}\bar{n} \rceil - 2$, i.e., $i \geq \lceil \frac{4}{3}\bar{n} \rceil - 1$. Thus, we must select $\lceil \frac{2}{3}\bar{n} \rceil$ jobs from the $2\bar{n} - \lceil \frac{4}{3}\bar{n} \rceil + 1$ shortest jobs. There are at most $\bar{n}$ combinations (and at least 1 combination). Therefore, we can investigate these combinations (and the corresponding schedule) in $O(n^2)$ time.

If i is odd, then $d_i = (a_2 + a_4 + \ldots + a_{i+1}) + \frac{i-1}{2}b + \delta < (\frac{i+1}{2} + 1)a_{2\bar{n}}$. We must have $(\frac{i+1}{2} + 1)a_{2\bar{n}} > \lceil \frac{2}{3}\bar{n} \rceil a_{2\bar{n}}$. We have $i > 2\lceil \frac{2}{3}\bar{n} \rceil - 3 \geq \lceil \frac{4}{3}\bar{n} \rceil - 3$, i.e., $i \geq \lceil \frac{4}{3}\bar{n} \rceil - 2$. So we must select $\lceil \frac{2}{3}\bar{n} \rceil$ jobs from the $2\bar{n} - \lceil \frac{4}{3}\bar{n} \rceil + 2$ shortest jobs. There are at most $\bar{n}^2$ combinations (and at least 1 combination). Thus, we can investigate these combinations (and the corresponding schedules) in $O(n^3)$ time.

b) Let $k = \lfloor \frac{2}{3}\bar{n} \rfloor$. In a stable schedule, we must have $d_i > \lfloor \frac{2}{3}\bar{n} \rfloor a_{2\bar{n}}$.

If i is even, then $d_i = (a_2 + a_4 + \ldots + a_i) + \frac{i-2}{2}b + \delta + (n - \frac{i}{2} + 1)(a_{i-1} - a_i) < (\frac{i}{2} + 1)a_{2\bar{n}}$. We must have $(\frac{i}{2} + 1)a_{2\bar{n}} > \lfloor \frac{2}{3}\bar{n} \rfloor a_{2\bar{n}}$. $i > 2\lfloor \frac{2}{3}\bar{n} \rfloor - 2 \geq \lfloor \frac{4}{3}\bar{n} \rfloor - 3$, $i \geq \lfloor \frac{4}{3}\bar{n} \rfloor - 2$. So we must select $\lfloor \frac{2}{3}\bar{n} \rfloor$ jobs from the $2\bar{n} - \lfloor \frac{4}{3}\bar{n} \rfloor + 2$ shortest jobs. There are at most $\bar{n}^2$ combinations (and at least 1 combination). Thus, we can investigate these combinations (and the resulting schedules) in $O(n^3)$ time.

If i is odd, then $d_i = (a_2 + a_4 + \ldots + a_{i+1}) + \frac{i-1}{2}b + \delta < (\frac{i+1}{2} + 1)a_{2\bar{n}}$. We must have $(\frac{i+1}{2} + 1)a_{2\bar{n}} > \lfloor \frac{2}{3}\bar{n} \rfloor a_{2\bar{n}}$. Then $i > 2\lfloor \frac{2}{3}\bar{n} \rfloor - 3 \geq \lfloor \frac{4}{3}\bar{n} \rfloor - 4$, i.e., $i \geq \lfloor \frac{4}{3}\bar{n} \rfloor - 3$. So we must select $\lfloor \frac{2}{3}\bar{n} \rfloor$ jobs from the $2\bar{n} - \lfloor \frac{4}{3}\bar{n} \rfloor + 3$ shortest

jobs. There are at most $\bar{n}^4$ combinations (and at least 1 combination). Thus, we can investigate these combinations (and the resulting schedules) in $O(n^5)$ time.

□

Thus, we can present the following Algorithm **T3**. All jobs $W_1, \ldots, W_{\bar{n}+1}$ are processed at the end of the schedule in any order. Then we investigate all schedules, which have $k = \lceil \frac{2}{3}\bar{n} \rceil$ or $k = \lfloor \frac{2}{3}\bar{n} \rfloor$ on-time jobs. We select these jobs according to Step 4 of the proof. Finally, we select the best schedule among the considered schedules.

2.7 An exact algorithm for the special case

$d_1 + p_1 \leq d_2 + p_2 \leq \ldots \leq d_n + p_n,\ p_1 < p_2 < \ldots < p_n$

Without loss of generality, let $d_i < \sum_{j=1}^{n} p_j$ for all $i \in N$. This special case can be solved by the following algorithm.

Algorithm T4

0. $P := \sum_{j=1}^{n} p_j,\ \bar{N} := N,\ S := \emptyset,\ \pi = (),\ \Pi := \emptyset$;
1. WHILE *for each job* $j \in \bar{N}$, *there is a job* $i \in \bar{N} \setminus \{j\}$ *such that* $d_i < P - p_j$ DO
   - 1.1. We select one job $j^* \in \bar{N}$ with minimal processing time;
   - 1.2. $\pi = (j^*, \pi),\ P := P - p_{j^*},\ \bar{N} := \bar{N} \setminus \{j^*\}$;
   - 1.3. FOR *each job* i *with* $d_i \geq P$ DO $\bar{N} := \bar{N} \setminus \{i\},\ \bar{S} := \bar{S} \bigcup \{i\}$.
2. WHILE *there is a job* $j \in \bar{N}$ *such that another job* $i \in \bar{N} \setminus \{j\}$ *exists for which* $d_i < P - p_j$ DO
   - 2.1. We investigate all schedules $\pi' = (\pi_1, l, \pi)$, where $\pi_1 = (\bar{N} \setminus \{l\}) \bigcup S,\ d_i \geq P - p_l$ for all $i \in \bar{N} \setminus \{l\}$. We include the best schedule π' into the set Π;
   - 2.2. We select one job $j^* \in \bar{N}$ with minimal processing time;
   - 2.3. We investigate all schedules $\pi'' = (\pi_1, l, j^*, \pi)$, where $\pi_1 = (\bar{N} \setminus \{l, j^*\}) \bigcup S,\ d_i \geq P - p_l - p_{j^*}$ for all $i \in \bar{N} \setminus \{l, j^*\}$. We include the best schedule π'' into the set Π;
   - 2.4. We keep only the best schedule in set Π;
   - 2.5. $\pi = (j^*, \pi),\ P := P - p_{j^*},\ \bar{N} := \bar{N} \setminus \{j^*\}$;

2.6. FOR *each job* i *with* $d_i \geq P$ DO $\bar{N} := \bar{N} \setminus \{i\},\ \bar{S} := \bar{S} \bigcup \{i\}$.

3. Now, we can select additionally at most one tardy job. We select job $j \in \bar{N}$ with the minimal due date d_j and construct the schedule $\pi = (\pi_1, j, \pi),\ \pi_1 = (\bar{N} \setminus \{j\}) \bigcup S$. Then we compare schedule π with the schedule from set Π and select the better one.

Lemma 23 *Algorithm* **T4** *constructs an optimal schedule for this special case in* $O(n^2)$ *time.*

Proof. It is known that, if $p_i < p_j$ and $d_i \leq d_j$, then there is an optimal schedule π, where $(j \rightarrow i)_\pi$ (see Emmons' rules). If there is a schedule $\pi = (\pi_1, i, \pi_2, \alpha, j, \pi_3)$, where $p_i < p_j,\ d_i > d_j$, job i is on-time, jobs $\alpha,\ j$ are tardy, then for schedule $\pi' = (\pi_1, j, \pi_2, \alpha, i, \pi_3)$ we have $F(\pi') - F(\pi) \geq 0$ since $T_\alpha(\pi') - T_\alpha(\pi) = p_j - p_i$, $(T_j(\pi) - T_j(\pi')) - (T_i(\pi') - T_i(\pi)) \leq d_i - d_j$, and $d_i + p_i \leq d_j + p_j \Rightarrow d_i - d_j \leq p_j - p_i$. This fact proves the optimality of steps *1, 2.3. and 2.6.* of the algorithm. It is obvious that the time complexity of Algorithm **T4** is equal to $O(n^2)$. $\square$

2.8 An exact algorithm for problem $1||\max \sum T_j$.

This algorithm is based on Lemma 9, i.e., there is an optimal schedule $\pi = (S, F) = (SPT, LPT)$, where all jobs $j \in F$ are tardy and all jobs $i \in S$ are on-time.

Algorithm T0

1. We enumerate the jobs as follows: $p_1 \geq p_2 \geq \ldots \geq p_n$. If $p_i = p_{i+1}$, then $d_i \geq d_{i+1}$;

2. $\pi_1(t) := (1),\ F_1(t) := \max\{0, p_1 + t - d_1\}$ for all $t \in Z$ with $t \in [0, \sum_{j=2}^{n} p_j]$;

3. FOR $l := 2$ TO n DO

   FOR $t := 0$ TO $\sum_{j=l+1}^{n} p_j\ (t \in Z)$ DO
   $\pi^1 := (l, \pi_{l-1}(t + p_l)),\ \pi^2 := (\pi_{l-1}(t), l)$;
   $F(\pi^1) := \max\{0, p_l + t - d_l\} + F_{l-1}(t + p_l)$;
   $F(\pi^2) := F_{l-1}(t) + \max\{0, \sum_{j=1}^{l} p_j + t - d_l\}$;
   $F_l(t) := \max\{F(\pi^1), F(\pi^2)\}$;
   $\pi_l(t) := \arg\max\{F(\pi^1), F(\pi^2)\}$.

4. We have obtained an optimal schedule $\pi_n(0)$ with the optimal function value $F_n(0)$.

Theorem 1 *Algorithm* **T0** *constructs an optimal schedule in* $O(n\sum p_j)$ *time.*

Proof. We prove the theorem indirectly. Assume that there is an optimal schedule of the form $\pi^* = (SPT, LPT)$, where $F(\pi^*) > F(\pi_n(0)) = F_n(0)$.

Let $\pi' := \pi^*$. For each $l = 1, 2, \ldots, n$, we successively consider the part $\bar{\pi}_l \in \pi'$, $\{\bar{\pi}_l\} = \{1, \ldots, l\}$ of the schedule. Let $\pi' = (\pi_\alpha, \bar{\pi}_l, \pi_\beta)$. If $\bar{\pi}_l \neq \pi_l(t = \sum_{i\in\pi_\alpha} p_i)$ (for the notation, see the last row in Step 3 of Algorithm **T0**, then $\pi' := (\pi_\alpha, \pi_l(\sum_{i\in\pi_\alpha} p_i), \pi_\beta)$. It is obvious that $F((\pi_\alpha, \bar{\pi}_l, \pi_\beta)) \leq F((\pi_\alpha, \pi_l(\sum_{i\in\pi_\alpha} p_i), \pi_\beta))$, and so on. At the end, we have $F(\pi^*) \leq F(\pi') \leq F_n(0)$. Thus, schedule $\pi_n(0)$ is also optimal.

Obviously, the time complexity of Algorithm **T0** is equal to $O(n\sum p_j)$. □

For a practical realization of the algorithm, we can use the idea from [11]. As computational experiments for the partition problem show, for a substantial part of instances, the time complexity time is polynomially bounded which might also be expected for the problem under consideration.

2.9 An alternative exact algorithm for problem $1||\max\sum T_j$.

Renumber the jobs according to the rule: $d_1 \leq d_2 \leq \ldots \leq d_n$, if $d_i = d_{i+1}$, then $p_i \leq p_{i+1}$ (EDD order). Let j^* be the job with maximal processing time $j^* := argmax\{d_i : p_i = \max_{k\in N} p_k\}$. Denote by jf the first tardy job in an optimal schedule $\pi = (S, F)$ and let $P(A) = \sum_{i\in A} p_i$.

According to Lemmas 10 and 13 we know that for all on-time jobs $i \in S$, we have $d_i \geq \max\{d_{jf}, \sum_{k\in S} p_k\}$, where $\sum_{k\in S} p_k = S_{jf}(\pi)$ with $S_{jf}(\pi)$ being the starting time of job jf in schedule π.

We know that a job j can be the first tardy job only when there is no job k with $k < j$, $d_k < d_j$, $p_k > p_j$ (see Lemma 15). Let $J = \{j_1, j_2, \ldots, j^*\}$ be the set of jobs that can be the first tardy job in an optimal schedule, where $N = \{1, 2, \ldots, n\} = \{1, 2, \ldots, j_1 - 1, j_1, j_1 + 1, \ldots, j_2, \ldots, j^*, \ldots, n\}$. If $j_k \in J$ and $j_l \in J$, $j_k < j_l$, then $S_{j_k} < d_{j_l}$ in all optimal schedules, where j_k is the first tardy job. Then for each job $j_k \in J$, we can consider all situations characterized by $S_{j_k} \in (d_\alpha, d_{\alpha+1}], \alpha = j_k, j_k + 1, \ldots, j_{k+1} - 1$ and $S_{j_k} \in (d_{j_k} - p_{j_k}, d_{j_k}]$, where S_{j_k} is the starting time of job j_k in an optimal schedule. It is obvious that there exists a situation $(j_k,\ S_{j_k})$ possibly corresponding to an optimal schedule, where $j_k \in J$ is the first tardy job,

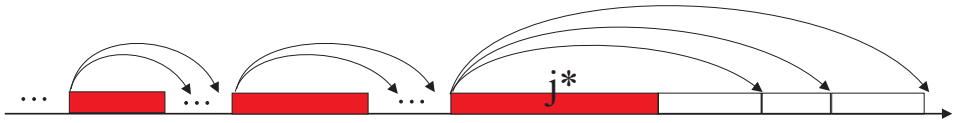

Figure 1: EDD order and choice of a situation in an optimal schedule.

and $S_{j_k} \in (d_\alpha, d_{\alpha+1}], \alpha = j_k, j_k+1, \ldots, j_{k+1}-1$ or $S_j \in (d_{j_k} - p_{j_k}, d_{j_k}]$. We can consider each of these situations separately. There are at most n such situations.

If we assume that $j_k \in J$ is the first tardy job and $S_{j_k} \in (d_\alpha, d_{\alpha+1}]$ in an optimal schedule, then all jobs $(1, \ldots, \alpha)$ are tardy, too. Without lost of generality, we can set $d_{j_k} := d_{\alpha+1}$ and consider a new instance with a modified due date d_{j_k} (see Figure 1 with an EDD schedule).

We propose the following algorithm.

Algorithm T

1. Compute $j^* := argmax\{d_i : p_i = \max_{k \in N} p_k\}$ and the set $J = \{j_1, j_2, \ldots, j^*\}$ of jobs that can be the first tardy job;

2. FOR each job $j_k \in J$ DO:

   - We consider each interval $(d_\alpha, d_{\alpha+1}], \alpha = j_k, j_k+1, \ldots, j_{k+1}-1$ and $(d_{j_k} - p_{j_k}, d_{j_k}]$ separately. Let $S_{j_k} \in (d_\alpha, d_{\alpha+1}]$. We set $F := \{i \in N, d_i < d_{\alpha+1}\} \bigcup \{j_k\}, S := \{i \in N, p_i > p_{j_k}\}$ and $SF := N \setminus F \setminus S$. For this situation, we get an optimal schedule π = BranchFunction(F, S, SF, j_k, α). Now we compare this schedule with the current best schedule.

BranchFunction(F, S, SF, j_k, α)

1. For each job $i \in F \bigcup SF$, we define the sets of jobs $F\_A_i = \{j \in F,\ p_j \geq p_i\}$, $F\_B_i = \{j \in F,\ p_j < p_i\}$, $SF\_A_i = \{j \in SF,\ p_j \geq p_i\}$ and $SF\_B_i = \{j \in SF,\ p_j < p_i\}$ (see Figure 2);

2. **Elimination rule 1** If for job $j \in F$, there is a job $k \in SF\_B_j$ such that $d_k - d_j < |F\_A_j|(p_j - p_k)$, where $|F\_A_j|$ is the number of elements in set F_A_j, then $F := F \bigcup \{k\}$, $SF := SF \setminus \{k\}$. The proof is obvious. If in an optimal schedule π, job k is on-time and j is tardy, then we can exchange these jobs and for the resulting schedule π', we have $F(\pi') > F(\pi)$;

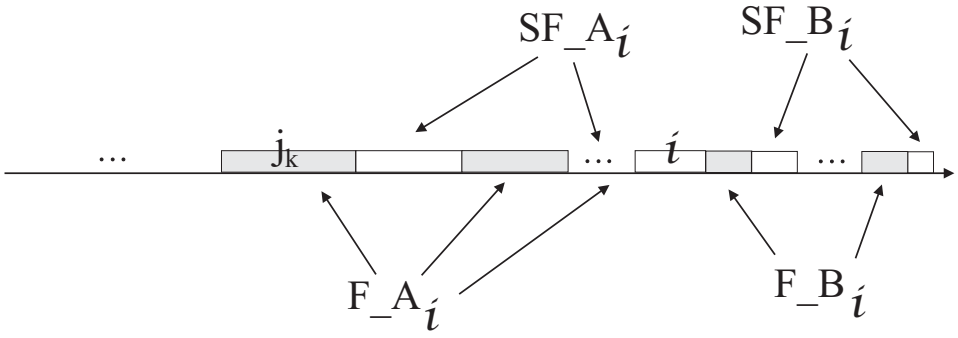

Figure 2: Branch function and sets of jobs

3. **Elimination rule 2** If for job $j \in F$, there is a job $k \in SF\_B_j$ such that $d_k \geq d_j$, then $F := F \bigcup \{k\},\ SF := SF \setminus \{k\}$.

4. **Elimination rule 3** We know that for this situation $S_{j_k} \leq d_{\alpha+1}$ holds. If for all $k \in SF\_A_i \bigcup F\_A_i$, where $d_{\alpha+1} + P(SF\_A_k) + P(F\_A_k) + p_k \geq \sum_{j=1}^{n} p_j - P(SF\_B_i) - P(SF\_B_i)$, we have $d_i - d_k < |F\_A_k|(p_k - p_i)$, then $F := F \bigcup \{i\},\ SF := SF \setminus \{i\}$;

5. **Elimination rule 4** If for all $k \in SF\_B_i \bigcup F\_B_i$, where $d_{\alpha+1} + P(SF\_A_i) + P(F\_A_i) \geq \sum_{j=1}^{n} p_j - P(SF\_B_k) - P(SF\_B_k) - p_k$, we have $d_k - d_i > |F\_A_i \bigcup SF\_A_i|(p_i - p_k)$, then $F := F \bigcup \{i\},\ SF := SF \setminus \{i\}$;

6. We construct the schedule $\pi = (SPT, LPT)$. All jobs from set S are processed at the beginning of the schedule. All jobs from sets F and SF are processed at the end of the schedule in an LPT order. If all jobs from sets F and SF are tardy in the schedule, then RETURN π.

7. Let $l \in F \bigcup SF$ be the last on-time job in schedule π. It is obvious that there is no optimal schedule, where all jobs from set $l \bigcup SF\_B_l \bigcup F\_B_l$ are tardy. Therefore, we must choose one job i from set SF_B_l (or from set $SF\_B_l \bigcup \{l\}$, if $l \in SF$) and put it into set S.

8. Let $i \in SF\_B_l$ (or $i = l$, if $l \in SF$) be the job with the maximal processing time from set SF_B_l (or $i = l$, if $l \in SF$);

9. We compute π_1 = BranchFunction$(F \bigcup \{i\}, S, SF \setminus \{i\}, j_k, \alpha)$ and π_2 = BranchFunction$(F, S \bigcup \{i\}, SF \setminus \{i\}, j_k, \alpha)$ and RETURN the best

schedule from π_1 and π_2.

Concluding remarks

In this paper, we have proposed a polynomial solution algorithm for problem $1||\max\sum U_j$ of maximizing the number of tardy jobs (when the first job starts at time zero and there is no idle time between the jobs). For the single machine problem $1||\max\sum T_j$ of maximizing total tardiness, we have developed a pseudo-polynomial solution algorithm. In addition, for several special cases of problem $1||\max\sum T_j$, we have presented exact polynomial algorithms.

References

[1] *Lazarev, A.: Dual of the Maximum Cost Minimization Problem,* Journal of Mathematical Sciences, Springer, Vol. 44, No. 5, 1989, 642 – 644.

[2] *Moore, J.M.: An n Job, One Machine Sequencing Algorithm for Minimizing the Number of Late Jobs,* Management Sci., Vol. 15, No. 1, 1968, 102 – 109.

[3] *Huang, R.H.; Yang, C.L.: Single-Machine Scheduling to Minimize the Number of Early Jobs,* IEEE International Conference on Industrial Engineering and Engineering Management, 2007, art.no. 4419333, 955 – 957.

[4] *Du J.; Leung J. Y.-T.: Minimizing Total Tardiness on One Processor is NP-hard,* Math. Oper. Res., 1990, Vol. 15, 483 – 495.

[5] *Lazarev, A.A.; Gafarov, E.R.: Special Case of the Single-Machine Total Tardiness Problem is NP-hard,* Journal of Computer and Systems Sciences International, 2006, Vol. 45, No. 3, 450 – 458.

[6] *Lawler, E.L.: A Pseudopolynomial Algorithm for Sequencing Jobs to Minimize Total Tardiness,* Ann. Discrete Math, 1977, Vol. 1, 331 – 342.

[7] *Szwarc, W.; Della Croce, F.; Grosso, A.: Solution of the Single Machine Total Tardiness Problem,* Journal of Scheduling, 1999, Vol. 2, 55 – 71.

[8] *Potts C.N.; Van Wassenhove L.N.: A Decomposition Algorithm for the Single Machine Total Tardiness Problem,* Oper. Res. Lett., 1982, Vol. 1, 363 – 377.

[9] *Lazarev, A.A.; Werner, F.: Algorithms for Special Cases of the Single Machine Total Tardiness Problem and an Application to the Even-Odd Partition Problem,* Mathematical and Computer Modelling, Vol. 49, No. 9-10, 2009, 2061 – 2072.

[10] *Lazarev A.A.; Kvaratskheliya A.G.; Gafarov E.R.: Algorithms for Solving the NP-Hard Problem of Minimizing Total Tardiness for a Single Machine,* Doklady Mathematics, 2007, Vol. 75, No. 1, 130 – 134.

[11] *Lazarev A.A.; Werner F.: A Graphical Realization of the Dynamic Programming Method for Solving NP-Hard Combinatorial Problems,* Computers and Mathematics with Applications, 2009, Vol. 58, No. 4, 619 – 631.