Minimizing Total Weighted Tardiness for Scheduling Equal-Length Jobs on a Single Machine

Evgeny R. Gafarov

V.A. Trapeznikov Institute of Control Sciences of the Russian Academy of Sciences, Profsoyuznaya st. 65, 117997 Moscow, Russia,
email: axel73@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 26, 2018

Abstract

In this paper, we consider the problem of minimizing total weighted tardiness for equal-length jobs with arbitrary release dates on a single machine. This problem is mentioned as a minimal open problem, see

http://www2.informatik.uni-osnabrueck.de/knust/class/dateien/classes/ein_ma/ein_ma ,

i.e., its complexity status is still open. The latest results on this problem were presented in the years 2000 and 2005, namely solution algorithms for special cases. We present some properties of this problem and discuss possible ways for future research.

Keywords: Scheduling, Single machine problems, Maximization problems, Total weighted tardiness, Number of tardy jobs

MSC classification: 90 B 35, 90 C 27, 68 Q 25, 68 W 40

1 Introduction

The scheduling 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 processing of the jobs starts at time 0. For each job $j \in N$, a constant processing time $p_j = p \in Z^+$, a due date $d_j \in Z^+$, a release date $r_j \in Z^+$ (i.e., the earliest possible starting time) and a weight $w_j \in Z^+$ are given.

A schedule π is uniquely determined by a permutation of the jobs of the set N. Let $C_j(\pi)$ be the completion time of job j in the 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. Moreover, let $T_j(\pi) = \max\{0, C_j(\pi) - d_j\}$ be the tardiness of job j in the schedule π. Denote by $S_j = C_j - p$ the starting time of job j in the schedule π.

In an active schedule, a job cannot be started earlier without violating the feasibility (i.e., without delaying the beginning of another operation). Without loss of generality, we consider only active schedules.

For the single machine problem of minimizing total weighted tardiness subject to given release dates and equal-length processing times, the objective is to find an optimal sequence π^* that minimizes total weighted tardiness, i.e.,

$$\sum_{j=1}^{n} f_j(C_j) = \sum_{j=1}^{n} w_j T_j.$$

We denote this problem by $1|r_j, p_j = p|\sum w_j T_j$ according to the traditional three-field notation $\alpha|\beta|\gamma$ for scheduling problems proposed by Graham et al. [6], where α describes the machine environment, β gives the job characteristics and further constraints and γ describes the objective function. If $w_j = 1, j \in N$, for all weights, we denote this special case by $1|r_j, p_j = p|\sum T_j$. For the single machine problem of minimizing the weighted number of tardy jobs, the objective is to minimize

$$\sum_{j=1}^{n} f_j(C_j) = \sum_{j=1}^{n} w_j U_j,$$

and the notation is $1|r_j, p_j = p|\sum w_j U_j$.

In [10, 3], Baptiste presented polynomial time dynamic programming algorithms to solve the problems $1|r_j, p_j = p|\sum T_j$ and $1|r_j, p_j = p|\sum w_j U_j$. Both algorithms are based on the following two properties:

- there exists a set

  $$\Theta = \{t : t = r_j + kp, j \in N, k = 0, 1, 2, \ldots, n-1\}$$

  of possible starting times of all jobs, where $|\Theta| \leq n^2$;
- there exists a dominance rule since a partial order of the jobs in an optimal schedule is known. Job i dominates job j if $d_i \leq d_j$. There exists an optimal schedule, where job j is processed after all jobs i with $r_i \leq S_j$ and $d_i \leq d_j$.

The special case of the problem $1|r_j, p_j = p|\sum w_j T_j$ with a common due date can be solved by Baptiste's algorithm, where job j is processed after all jobs i with $r_i \leq S_j$ and $w_i \geq w_j$.

The special case of the problem $1|r_j, p_j = p|\sum w_j T_j$, where the release dates are a multiple of p, can be reduced to an assignment problem and thus, solved in polynomial time.

In [11], Verma and Dessouky considered a single machine problem with earliness and tardiness penalties $1|p_j = p|\sum(\alpha_j E_j + \beta_j T_j)$, where $E_j(\pi) = \max\{0, d_j - C_j(\pi)\}$. They formulated this problem as a time-indexed ILP and showed that, when certain criteria are met, there exists an integral optimal solution to the LP relaxation, which means that there exists a polynomial time solution procedure. In the ILP formulation, they defined a variable $x_{j,t}$ for all relevant jobs j and possible completion times t, where $x_{j,t} = 1$ if job j is completed at time t and $x_{j,t} = 0$ otherwise. In the LP relaxation, we have $x_{j,t} \in [0, 1]$. Verma and Dessouky described a guaranteed way to convert a fractional solution into an integral solution with equal objective function value.

In [1], a similar ILP formulation and the same LP relaxation for the problem $1|r_j, p_j = p|\sum w_j T_j$ were presented. The authors stated that, if the instance does not contain two jobs i

and j such that $d_i < d_j$ and $w_i < w_j$, then the LP relaxation can be converted into an optimal solution, and this case can be solved in polynomial time. Unfortunately, the authors did not provide a conversion procedure and link to the result of Verma and Dessouky [11], although in contrast to the problem $1|r_j, p_j = p|\sum w_j T_j$, in the problem $1|p_j = p|\sum(\alpha_j E_j + \beta_j T_j)$, there is a common release date. Moreover, the set of completion times of job j considered belongs to another interval.

The rest of this paper is organized as follows. In Section 2, some properties of the problem are presented. Solution procedures for the problem are considered in Section 3. Two solution algorithms for special cases are presented in Section 4. The complexity status of the problem is discussed in Section 5. Some maximization scheduling problems with equal-length jobs are formulated and considered in Section 6.

2 Properties of the problem $1|r_j, p_j = p|\sum w_j T_j$

We consider the processing order of two jobs i and j in an optimal schedule.

Property 1. *Let t_1 be the starting time of the earlier job from the set $\{i, j\}$ and t_2 be the starting time of the second one, where $t_2 \geq t_1 + p$. Then the following dominance rules hold:*

1. *If $w_i \geq w_j$ and $d_i \leq d_j$ and $t_1 \geq r_i$, then job i precedes job j;*
2. *If $w_i < w_j$ and $d_i \leq d_j$ and $t_1 \geq \max\{r_i, r_j\}$, then:*
   (a) *If $t_2 + p \leq d_j$ (job j is on-time if it is processed from time t_2) then i precedes j;*
   (b) *If $t_1 > d_j - p$ (both jobs are tardy), then job j precedes job i;*
   (c) *Let $d_i - p < t_1 \leq d_j - p$. Moreover, let in the schedule $\pi = (\pi_1, i, \pi_2, j, \pi_3)$ the starting times be $S_i(\pi) = t_1$ and $S_j(\pi) = t_2$ and in the schedule $\pi' = (\pi_1, j, \pi_2, i, \pi_3)$, they are $S_i(\pi') = t_2$ and $S_j(\pi') = t_1$. Then job i precedes job j if the following condition holds:*

$$\sum w_j T_j(\pi) < \sum w_j T_j(\pi') \iff w_i(t_2 - t_1) - w_j(t_2 + p - d_j) > 0$$
$$\iff t_2 < \frac{w_j(d_j - p) - w_i t_1}{w_j - w_i};$$

   (d) *Let $t_1 + p \leq d_i \leq d_j < t_2 + p$. Then job i precedes job j if the following condition holds:*

$$\sum w_j T_j(\pi) < \sum w_j T_j(\pi') \iff w_i(t_2 + p - d_i) > w_j(t_2 + p - d_j)$$
$$\iff t_2 < \frac{w_j(d_j - p) - w_i(d_i - p)}{w_j - w_i}.$$

This dominance rules are illustrated in Fig. 1(a), where the origin is (r_{max}, r_{max}) with $r_{max} = \max\{r_i, r_j\}$, point

$$A = \left(d_i - p; \frac{w_j(d_j - p) - w_i(d_i - p))}{w_j - w_i}\right)$$

and point B is the intersection of two lines:

$$t_2 = t_1 + p \qquad \text{and} \qquad t_2 = \frac{w_j(d_j - p) - w_i t_1}{w_j - w_i}.$$

There are no points below the line $t_2 = t_1 + p$.

We note that, if $t_1 < r_j$, then it is difficult to find such relationships since in the schedule π', jobs that are processed in the interval $[t_1 + p, t_2)$ can be moved to the right.

In the dynamic programming algorithm of Baptiste [3], intervals $[s, e], s, e \in \Theta$, are considered, where jobs from the subset $N' = \{1, 2, \dots, k\}, k \leq n$, are processed in those intervals, i.e., no more than k jobs in each interval. The running time of the DP algorithm depends on the number of intervals considered and is computed in [3] as $O(n^4)$, since $|\Theta| = O(n^2)$. The question is: How many points e should be considered for each fixed value s, i.e., is there an instance with $O(n^2)$ points e?

Property 2. *For fixed values s and k, there exist $O(k^2)$ points $e \in \Theta$.*

Proof. We construct the following instance, where

$$\frac{k}{4} = \left\lceil \frac{k}{4} \right\rceil \qquad \text{and} \qquad p \gg k.$$

For the first $\frac{k}{4}$ jobs, we have

$$r_1 = s, \quad r_2 = r_1 + 2p, \quad r_3 = r_2 + 2p, \quad \ldots,$$

i.e.,

$$r_i = s + 2p(i-1), \quad i = 1, \ldots, \frac{k}{4}.$$

For the next $\frac{k}{4}$ jobs, we have

$$r_i = r_{i-\frac{k}{4}+1} + 1, \quad i = \frac{k}{4}, \frac{k}{4} + 1, \ldots, \frac{k}{2}.$$

For the next $\frac{k}{2}$ jobs, we assume

$$r_i = \frac{k}{2}p + 1 + \left(i - \frac{k}{2}\right), \quad i = \frac{k}{2} + 1, \ldots, k.$$

In an active schedule, there are between $\frac{k}{4}$ to $\frac{k}{2}$ jobs processed at the beginning before time $r_{\frac{k}{2}+1}$ and between $\frac{k}{2}$ to $\frac{3k}{4}$ jobs processed from time $r_i, i \in [\frac{k}{2} + 1, k]$, without idle time.

Then there exist $O(k^2)$ active schedules and $O(k^2)$ possible completion times e. $\square$

Property 3. *Let*

$$T = \left\{ t : t \in \Theta \bigcap [t_1, t_1 + p] \right\},$$

i.e., the set of points that belong to an interval of the length p. Then $|T| = O(n)$.

3 Solution procedures

The problem under consideration can be solved by dynamic programming with the functional equations presented in [8].

Suppose that we are given a schedule and that the first k jobs performed under that schedule are represented by the set $N \setminus J$ and the remaining $n - k$ jobs by the set J. Now, let $F(J, t)$ denote the minimal total weighted tardiness of the jobs J, subject to the constraint that no job is started before $t \in \Theta$. The solution of the problem involves finding the value of $F(N, 0)$ and the schedule of those jobs which yields this function value. With some reflections, it becomes apparent that

$$F(J, t) = \min_{i \in J} \left\{ w_i \max\{0, t + p - d_i\} + F(J \setminus \{i\}, t + p) \right\},$$

and $F(\emptyset, t) = 0, \forall t \in \Theta$.

A computational procedure for the solution of this problem only requires to evaluate these functional equations. The running time of this dynamic programming algorithm is $O(n^3 2^n)$.

There can be given another algorithm with the same running time. We consider 2^n subsets $N' \subseteq N$, where

$$|r_{j_1} - r_{j_2}| \geq p, \;\; j_1, j_2 \in N'.$$

For each subset N', let $S_j = r_j, j \in N'$, and for each job $i \in N \setminus N'$, the possible starting time belongs to the set Θ' which is a subset of the $n - |N'|$ smallest values of the set

$$\{t : t = r_j + kp, j \in N', k = 1, 2, \ldots, n - |N'|\}.$$

Then the problem is reduced to an assignment problem which can be solved in $O((n - |N'|)^3)$ time.

Baptiste's algorithm [3] for the special case with equal weights can be used as a heuristic to solve the problem with arbitrary weights.

If we assume in this algorithm that job j is processed after all jobs i with $r_i \leq S_j$ and $w_i \geq w_j$, then we can present a counterexample, where in an optimal job schedule π, job j is processed before job i and

$$C_j - d_j = 1, \qquad C_i - d_i = 1, \qquad C_j < C_i.$$

Thus, Baptiste's algorithm does not lead to an optimal solution for this instance.

3.1 Local search

Local search is a heuristic method to solve optimization problems. Local search algorithms move from one solution to another solution in the space of candidate solutions (the search space) by applying local changes, until a solution deemed optimal is found or a time bound is elapsed.

We consider the following local changes of a schedule and also their simultaneous realization. Let an active schedule be denoted by the job sequence $\pi = (\pi_1, i, \pi_2, j, \pi_3)$.

- **left-shift**: Schedule job j immediately after the completion of the partial sequence π_1, i.e., we obtain the neighbor $\pi' = (\pi_1, j, i, \pi_2, \pi_3)$;
- **right-shift**: Schedule job i immediately after the completion of the partial sequence π_2, i.e., we obtain the neighbor $\pi' = (\pi_1, \pi_2, i, j, \pi_3)$;
- **pairwise interchange**: Interchange the jobs i and j, i.e., we obtain the neighbor $\pi' = (\pi_1, j, \pi_2, i, \pi_3)$.

The question is:

- Can we reach an optimal schedule from an initial one by a descent procedure, i.e. by a sequence of these local changes such that in each step, we perform only one of the modifications described above and after each step, the objective function value does not increase.

Property 4. *There exists a schedule, where each left-shift increases the optimal function value, but a simultaneous realization of several ones decreases it.*

Proof. Consider the instance with

$$r_1 = 2, r_2 = 0, r_3 = 9, r_4 = 6, p = 3, d_1 = 6, d_2 = 3, d_3 = 12, d_4 = 9, w_1 = w_3 = 100, w_2 = w_4 = 1$$

(see Fig.1(b)) and let the initial schedule be $\pi = (1, 2, 3, 4)$. By a left-shift of job 2 to the position before job 1, we increase the objective function value. Analogously, a left-shift of job 4 before job 3 in π increases it. However, the simultaneous realization of both left-shifts leads to the job sequence $\pi' = (2, 1, 4, 3)$ which decreases the objective function value. For the sequence π', the total weighted tardiness is equal to 0 and for the sequence π, it is equal to 11. □

Corollary 1. *Let π be a job sequence, where each single left-shift increases the objective function value. Then the relative error of π can become arbitrarily large.*

The same proof can be used for the following property as well.

Property 5. *There exists a schedule, where each pairwise interchange increases the objective function value, but a simultaneous realization of several ones decreases it.*

Property 6. *Let π be an initial schedule and $N' = \{j_1, j_2, \ldots, j_k\}$ be a given subset of jobs to be shifted to the left. The order of the jobs from the set N' is fixed, i.e., job j_1 precedes job j_2, job j_2 precedes job j_3, etc. Then an optimal left-shift combination can be computed in $O(n^8)$ operations.*

Proof. An optimal left-shift combination can be computed by the following dynamic programming algorithm. At each stage $l = k, k-1, \ldots, 1$, for job $j_l \in N'$, we consider all possible positions x_l in a job sequence, where the order of the jobs $N \setminus N'$ remains the same, and all possible starting times $t_l \in \Theta$. This means that the state of a system in the dynamic programming algorithm is defined by the vector (x_l, t_l). For each state, we compute the function value $F_l(x_l, t_l)$ which is the total weighted tardiness of the jobs processed in an active schedule from time t_l on by a sub-sequence started with the job j_l. This function is used in the next step for the job j_{l-1}. There are $O(n^{1+2})$ states at each stage. For each state (x_l, t_l), the states obtained at the previous stage are considered and $O(n)$ operations are needed to compute $F_l(x_l, t_l)$. So, the running time of the dynamic programming algorithm is equal to $O(n^8)$. $\square$

Property 7. *Assume that for a job sequence* $(1,2,3)$*, the two adjacent pairwise interchanges* $1 \Longleftrightarrow 2$ *and* $2 \Longleftrightarrow 3$ *increase the objective function value. There exists an instance, where the interchange* $1 \Longleftrightarrow 3$ *leads to a better job sequence.*

Such an instance can be given as follows:

$$r_1 = r_2 = r_3 = 0, p = 3, d_1 = 5, d_2 = 7, d_3 = 8, w_1 = 1, w_2 = w_3 = 5.$$

For the job sequence $(1,2,3)$, we have $\sum w_j T_j = 5$ and for the job sequence $(3,2,1)$, we have $\sum w_j T_j = 4$.

Corollary 2. *If we consider a local search procedure with pairwise job interchanges, then also pairs of non-adjacent jobs have to be considered, i.e.,* $O(n^2)$ *pairs of jobs.*

Property 8. *Let for the job sequence* $(1,2,\ldots,n-1,n)$ *all pairwise interchanges increase the objective function value. There exists an instance, where a left-shift leads to a better job sequence* $(n,1,2,\ldots,n-1)$.

Proof. For an explanation, see Fig.1(c). Consider the following instance:

$$\begin{cases} n > 3, & \\ r_i = 0, & i = 1\ldots,n, \\ p = 2, & \\ w_i = pw_n - 1, & i = 1\ldots,n-2, \\ w_{n-1} = pw_n + 1, & i = 1\ldots,n-2, \\ d_n = 0, & \\ d_i = (i+1)p - 1, & i = 1\ldots,n-1, \end{cases} \tag{1}$$

For the job sequence $\pi = (1,2,\ldots,n-1,n)$, all pairwise interchanges increase the objective function value. For the sequence $\pi' = (n,1,2,\ldots,n-1)$, we have

$$\begin{aligned} F(\pi') &= F(\pi) + \sum_{i=1}^{n-1} w_n - p(n-1)w_n \\ &= F(\pi) + p(n-1)w_n - (n-2) + 1 - p(n-1)w_n < F(\pi). \end{aligned}$$

$\square$

Property 9. *There exists a schedule, where each left-shift increases the objective function value, but there exists a right-shift which decreases it.*

3.2 Relaxation of the problem

In this section, we consider a relaxation of the problem, i.e., an instance I', where the values r_i are rounded such that the distance between them is a multiple of p. For example

$$r'_i = \left\lfloor \frac{r_i}{p} \right\rfloor p, \qquad d'_j = d_j - (r_j - r'_j).$$

The instance I' is reduced to the assignment problem and can be solved in polynomial time. Let π be an optimal job sequence for an initial instance I and π' be an optimal job sequence for a modified instance I' with rounded parameters.

Now, the questions are:

- What is the relative error
$$\frac{F(\pi) - F(\pi')}{F(\pi)}$$
for the instance I?
- What is the difference between $F(\pi)$ for I and $F'(\pi')$ for I', where F' denotes the function value for the rounded data.?

Property 10. *There exists an instance, where*

$$\frac{F(\pi)}{F'(\pi')}$$

can become arbitrarily large.

Let $n = 2, p = 3, r_1 = 1, r_2 = 3, d_1 = 4, d_2 = 6, w_1 = w_2 = 1$. For $\pi = \pi' = (1, 2)$, we get $F(\pi) = 1$. Moreover, we have $F'(\pi') = 0$, where $r'_1 = 0, d'_1 = 3, r'_2 = r_2, d'_2 = d_2$.

Property 11. *There exists an instance, where*

$$\frac{F(\pi) - F(\pi')}{F(\pi)}$$

can become arbitrarily large.

Consider the following instance:

$$\begin{cases} p = 4, & \\ r_n = 0, & \\ r_i = \frac{p}{2}, & i = 1 \ldots, n-1, \\ d_i = r_i + 2p - 1, & i = 1 \ldots, n-1, \\ d_n = p, & \\ w_n = 1, & \\ w_i = np, & i = 1 \ldots, n-1. \end{cases} \tag{2}$$

The job sequence $\pi' = (1, 2, \ldots, n-1, n)$ is optimal for the instance I', where job n is the only tardy one. For the instance I, there is only one tardy job n as well. The job sequence $\pi = (n, 1, 2, \ldots, n-1)$ is optimal for the instance I with no tardy jobs.

We can consider another relaxation, where r'_i is calculated such that $\sum_{j=1}^{n} |r_j - r'_j|$ is minimized. For this relaxation, Properties 10 and 11 hold as well. To prove this, we only need to consider n additional jobs with $r_j = np, j = n+1, \ldots, 2n$.

4 Special cases of the problem $1|r_j, p_j = p|\sum w_j T_j$

In this section, algorithms to solve two special cases are considered. As it was mentioned above, the special case with a common due date can be solved by Baptiste's algorithm.

First, we consider the special case, where $d_{max} - d_{min} \le p$. This special case can be solved by the following algorithm.

Choose two straddling jobs j_1 and j_2 that will be processed in an active schedule one by one from a starting time S_{j_1} such that no other jobs can be processed in the interval $[d_{min}, d_{max}]$, i.e.,

$$S_{j_1} \le d_{min}, C_{j_2} \ge d_{max}.$$

Then the sub-problem with the subset of jobs $N \setminus \{j_1, j_2\}$ can be solved by a modification of Baptiste's algorithm in $O(n^7)$ time, where job j is processed after all jobs i with $r_i \le S_j$ and $w_i \ge w_j$. In the modified algorithm, only intervals $[s, e]$ are considered which have no intersections with the interval $[S_{j_1}, C_{j_2}]$.

There are $O(n^2)$ pairs of jobs j_1 and j_2 and $O(n)$ possible starting times $S_{j_1} \in \Theta$ according to Property 3 and a single point S_{j_2} for a chosen value S_{j_1} since only active schedules are considered. Thus, this special case can be solved in $O(n^{2+1+7})$ time.

Second, we consider the following special special case with n jobs satisfying the following inequalities:

$$\begin{cases} w_1 \le w_2 \le \ldots \le w_{n-1}, \\ d_1 \ge d_2 \ge \ldots \ge d_{n-1}, \end{cases} \tag{3}$$

To solve this problem, we have to choose a starting time $S_n \in \Theta$ and to solve the rest of the problem by a modification of Baptiste's algorithm. Thus, this special case can be solved in $O(n^{2+7})$ time.

5 Computational results for the problem $1|r_j, p_j = p|\sum w_j T_j$

In this section, we present some results of a numerical experiment, where we investigate the number of possible starting times from the set Θ when k jobs are already scheduled.

We consider the following special case:

$$\begin{cases} w_1 \leq w_2 \leq \ldots \leq w_n, \\ d_1 \leq d_2 \leq \ldots \leq d_n, \end{cases} \tag{4}$$

with $n = 10$.

We generated the instances as follows. For each $p \in \{5, 10, 15, 20, 25, 30\}$, we generated 10 instances with $r_j \in [0, (n-2)p], d \in [0, (n-1)p], w \in [1, 120]$. For each instance, a branch and bound algorithm is used with the following branching procedure *Branching(j,* Π*)*, where j is the job to be scheduled in the procedure and Π is a partial schedule, for which the starting times $S_i, i = 1, 2, \ldots, j-1$, are defined. In the procedure, we calculate the set Θ_j according to the sequence Π and the release date r_j. Let $S_{j'} < S_{j''}, j', j'' \leq j-1$, and there is no other $j''' \leq j-1$, for which $S_{j'} < S_{j'''} < S_{j''}$. Then we introduce the sets

$$\begin{aligned} \Omega_1 &= \{S_{j'} + kp, k = 1, 2, \ldots, n-j : S_{j'} + kp + p \leq S_{j''}, S_{j'} + kp \geq r_j\}, \\ \Omega_2 &= \{r_i + kp, k = 1, 2, \ldots, n-j, i \geq j, r_j \leq r_i + kp \leq S_{j''} - p, r_i > S_{j'} + p\} \text{ and} \\ \Omega_3 &= \{S_{j''} - kp, k = 1, 2, \ldots, n-j : S_{j''} - kp - p \geq S_{j'}, S_{j''} - kp \geq r_j\}. \end{aligned}$$

Let t be the maximal value from the set Ω_2, where $(S_{j''} - t)$ is a multiple of p. If such a t exists, then we delete all the values from the set Ω_2 which are larger than t and all the values from the set Ω_3 which are less than t. Then we add to the set Θ_j the sets $\Omega_1, \Omega_2, \Omega_3$ and the value r_j if $S_{j'} + p < r_j < S_{j''} - p$.

In the branching procedure, we consider all possible starting times $t \in \Theta_j$. Let $TWT(\Pi)$ be the total weighted tardiness of the jobs in the sequence Π. If

$$TWT(\Pi) + w_j \max\{t + p - d_j, 0\} \geq UB,$$

then t is excluded from the further considerations, where an upper bound UB on the optimal function value is the best objective function value found so far.

First, we report the results exemplary for one of the considered instances with the following data.

$$\begin{aligned} p = 10, \quad & r_j \in \{72, 58, 29, 56, 49, 58, 55, 70, 57, 72\}, \\ & d_j \in \{66, 71, 73, 75, 76, 76, 82, 85, 88, 88\}, \\ & w_j = \{22, 30, 42, 56, 68, 75, 75, 94, 103, 111\}, \end{aligned}$$

For this instance, we have 33 346 541 nodes, and the numbers of nodes considered for $j = 1, 2, \ldots, n$ are

$$\{58; 1\ 926; 43\ 059; 588\ 149; 3\ 820\ 730; 9\ 432\ 283; 14\ 004\ 668; 4\ 636\ 929; 818\ 539; 200\}.$$

If we use the dominance rules presented in Property 1, then the number of nodes considered is 5 303 348, and the numbers of nodes for $j = 1, 2, \ldots, n$ are

$$\{58; 1\ 666; 26\ 343; 233\ 762; 962\ 857; 1\ 457\ 811; 1\ 979\ 532; 531\ 006; 110\ 311; 2\}.$$

According to results of the experiment, these dominance rules reduce the number of nodes in the set Θ_j on 47%. In Tables 1 and 2, some more detailed results for the instances with $p \in \{5, 10, 15\}$ and $p \in \{20, 25, 30\}$, respectively, are reported. In columns 1 - 3, we give the release dates, due dates and weights of the 10 jobs, respectively. In column 4, the schedule is described by the starting times of the jobs. In column 5, we give the optimal function value TWT. In columns 7 and 8, the total number $TNN1$ of nodes considered without using of domination rules and the total number of nodes $TNN2$ considered with the use of the dominance rules are given. Finally, column 8 gives for each instance the percentage (%) of branches considered when using the dominance rules.

We also note that the running time of the branch and bound algorithm is exponential, and it does not solve instances with $n = 20$ in 60 minutes on a PC with Intel Core 2 Duo CPU P8600 2,4 GHz and 4 GB RAM.

Table 1: Results for $p \in \{5, 10, 15\}$

r_j	d_j	w_j	Schedule	TWT	$TNN1$	$TNN2$	%
$p = 5$							
21,0,35,31,26, 11,10,25,11,14	8,26,35,35,38, 40,42,43,44,44	6,13,20,20,58, 59,84,90,110, 116	50,0,40,45,30, 15,10,25,20,35	782	199344	107496	54
17,36,8,34,20, 25,16,39,6,23	19,19,28,32,32, 37,38,39,43,44	14,21,23,46,50, 60,67,89,103, 107	51,46,11,36,21, 26,16,41,6,31	2227	620833	298294	48
16,26,10,5,25, 29,11,19,31,14	30,31,33,34,36, 39,39,43,44,44	9,15,44,50,52, 64,77,89,91,113	50,45,10,5,25, 30,15,20,40,35	601	196973	112646	57
35,35,37,2,22, 9,16,8,35,21	16,22,23,31,35, 35,40,41,42,44	22,27,31,54, 65,77,84,105, 105,108	50,45,40,2,23, 13,18,8,35,28	2296	205273	80800	39
7,18,10,31,4, 7,4,20,19,0	11,20,30,30,33, 34,35,37,41,44	9,9,45,48,50, 62,64,85,87,117	41,46,10,31,5, 15,20,25,36,0	882	264585	207667	78
$p = 10$							
7,65,56,9,17, 39,77,16,64,79	35,50,56,70,74, 81,86,87,87,89	39,46,54,55,64, 71,84,97,104,104	7,97,57,17,27, 47,77,37,67,87	4132	100035	54247	54
53,63,4,23,26, 27,18,10,49,28	36,36,42,55,55, 68,75,81,85,88	1,24,26,27,27, 46,58,79,95,107	94,84,4,24,34, 44,54,14,64,74	1460	592976	416717	70
32,56,10,4,49, 78,57,42,40,23	41,51,67,72,83, 84,84,85,87,89	11,28,29,34,41, 48,56,61,77,85	34,56,14,4,96, 86,66,44,76,24	1972	414289	316671	76
24,76,5,36,21, 57,70,36,59,52	27,35,48,69,71, 73,76,79,83,88	9,23,26,34,71, 72,102,105,113, 114	107,97,5,36,21, 57,87,46,67,77	4608	6476173	3329563	51
65,2,8,3,29, 34,37,33,47,57	16,46,47,62,64, 79,79,83,83,87	25,31,32,49,79, 90,94,100,101, 108	92,2,12,22,32, 42,52,62,72,82	2690	849744	367675	43
$p = 15$							
97,105,22,98,42, 32,114,47,79,86	67,89,103,112, 114,116,118,130, 131,134	5,36,36,47,47, 48,53,103,112, 116	158,143,22,98, 52,37,128,67, 82,113	4386	1709603	1166960	68
93,31,60,55,28, 90,51,29,106,85	50,68,68,76, 86,89,105,119, 122,129	12,13,17,27,50, 52,60,63,63,90	166,151,136,58, 28,90,73,43, 106,121	5719	4417446	1328448	30
54,37,9,104,93, 64,35,26,20,61	41,109,118,123, 126,127,128,129, 130,133	6,11,20,35,45, 56,59,63,91,109	144,129,9,114, 99,69,39,54,24, 84	1303	386234	318040	82
19,58,44,22,101, 44,61,27,81,119	65,70,79,80, 88,90,102,112, 122,134	2,18,29,43,50, 56,67,81,95,110	157,142,52,22, 112,67,82,37,97, 127	4610	1950750	843607	43
18,51,73,14,57, 50,72,57,56,119	54,73,86,89, 95,105,110,116, 132,134	1,13,20,35,44, 68,85,95,96,111	18,155,140,33, 65,50,80,95,110, 125	3307	3689263	1928262	52

Table 2: Results for $p \in \{20, 25, 30\}$

r_j	d_j	w_j	Schedule	TWT	$TNN1$	$TNN2$	%
$p = 20$							
71,25,58,69, 93,122,72,123, 95,70	102,116,125,140, 142,151,164,166, 166,175	3,6,8,10,43, 51,67,87,93, 113	218,25,58,198, 98,178,78,138, 118,158	3924	29709961	6809732	23
83,67,52,154, 92,60,38,49, 65,80	73,103,103,117, 146,156,166,171, 172,175	18,18,47,58, 67,69,77,81, 105,118	198,218,58,178, 98,78,38,118, 138,158	10092	1849948	1203732	65
101,41,18,116, 101,42,59,138, 104,140	86,111,117,123, 137,158,166,169, 170,176	2,19,23,24, 25,27,52,59, 110,110	201,41,18,181, 101,61,81,141, 121,161	2692	2074218	884291	43
55,84,135,56, 36,98,60,5, 28,43	83,98,106,106, 111,112,130,149, 165,170	2,22,34,37, 52,54,63,72,82, 113	188,128,168,68, 48,108,88,5, 28,148	5002	758835	471049	62
145,95,25,148, 40,28,84,135, 46,52	87,87,93,122, 153,155,160,166, 171,178	18,29,36,38, 42,54,75,84, 90,96	205,105,25,185, 45,65,85,145, 125,165	7412	1288063	579338	45
$p = 25$							
5,93,118,175, 140,129,22,63, 20,74	121,129,146,150, 167,168,169,173, 178,211	26,31,32,51, 52,56,84,89, 96,106	5,105,230,205, 155,130,30,80, 55,180	8275	46808	29428	63
193,142,193,194, 168,175,107,186, 164,59	152,197,200,200, 218,219,220,222, 222,224	25,33,44,46, 50,55,64,71, 84,97	317,142,292,267, 242,217,107,192, 167,59	17845	2103722	444158	21
176,177,167,25, 48,113,65,80, 66,170	100,120,136,142, 152,186,201,202, 205,221	5,17,26,28, 58,81,100,106, 109,118	250,225,175,25, 50,125,75,100, 150,200	5221	973995	807576	83
137,41,14,164, 146,186,74,36, 119,74	83,92,127,132, 167,174,196,208, 217,220	5,11,31,50, 58,66,80,83, 114,116	246,41,14,171, 146,221,91,66, 119,196	9240	513204	278014	54
138,79,40,50, 109,2,163,156, 132,190	72,105,110,151, 153,169,200,204, 208,221	4,48,53,57, 59,60,60,74, 80,118	240,90,40,65, 115,2,215,165, 140,190	3652	217856	173628	80
$p = 30$							
55,1,219,107, 32,190,92,17, 80,130	119,129,143,150, 180,184,198,253, 255,263	7,9,36,40, 53,84,87,102, 112,112	280,1,250,121, 61,190,151,31, 91,220	9333	641049	342412	53
46,239,103,164, 226,50,209,83, 117,51	117,129,163,166, 208,210,245,264, 268,269	19,28,52,53, 79,81,84,106, 110,114	46,316,106,166, 286,76,226,136, 196,256	19060	494327	325578	66
139,13,155,191, 31,182,175,158, 53,101	173,191,192,208, 214,221,227,241, 246,257	15,16,17,31, 44,45,61,80, 84,101	139,13,289,259, 43,229,199,169, 73,103	6502	620830	377214	61
191,75,230,187, 26,51,77,152, 148,45	112,119,154,220, 222,230,232,242, 246,266	9,13,26,26, 58,68,88,99, 106,110	298,86,268,208, 26,56,116,178, 148,238	6376	536269	269281	50
79,132,172,203, 126,81,179,178, 183,113	166,178,205,210, 219,241,242,247, 254,255	2,13,20,27, 66,83,95,98, 114,119	358,328,298,268, 143,81,208,178, 238,113	9216	22248262	7730427	35

6 Complexity of the problem $1|r_j, p_j = p|\sum w_jT_j$

To the best of our knowledge, there are two common ways to prove the NP-hardness of a single machine problem with classical constraints p_j, w_j, d_j, r_j, D_j and classical objective functions $C_{max}, L_{max}, \sum w_jC_j, \sum w_jT_j, \sum w_jU_j$:

- by a reduction from the Partition problem (Knapsack problem, 3-Partition) by assuming that p_j depends on the values b_j from the Partition problem. In such a reduction, the number of possible completion times of the jobs is not restricted by $O(n^2)$;
- by a reduction from a graph-theoretic problem (e.g., Clique) if precedence relations are given.

However, both these ways do not lead to a proof of NP-hardness of the problem $1|r_j, p_j = p|\sum w_jT_j$.

In such proofs mentioned above, a special case of a single machine problem is considered and the structure of an optimal schedule for this case is known, see e.g. [7]. However, for a problem with equal-length jobs, it is likely to construct a polynomial dynamic programming to find the best solution if its structure is known.

So,

- **on the one hand**, problem $1|r_j, p_j = p|\sum w_jT_j$ does not contain difficulties like an exponential number of possible completion times or precedence relations and other single machines problems with equal-length jobs and classical constraints and functions are solvable in polynomial time,
- **on the other hand**, no dominance rules for this problem are known and an effective solution procedure is not known.

We conjecture that problem $1|r_j, p_j = p|\sum w_jT_j$ is NP-hard, but to prove it, a new way (i.e., another kind of a reduction from an NP-hard problem) has to be considered.

7 Maximization problems

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 section, we consider single machine problems with an *opposite* criterion, namely we consider the maximization of makespan, total completion time or number of tardy jobs.

Investigations of problems with *opposite* optimization criteria itself are an important theoretical task. In addition, such problems separately have practical interpretations and applications [2, 4, 5]. The maximal makespan can be also used to reduce the set Θ in minimization problems.

Here we consider only active schedules (note that the maximization problem considered in this paper would be trivial when allowing arbitrarily inserted unnecessary idle times, since the maximal objective function value can become arbitrarily large in this case).

In the single machine makespan maximization problem, the objective is to find an optimal schedule π^* that maximizes $C_{max}(\pi) = \max_{j=1}^{n}\{C_j\}$. We denote this problem by $1|r_j, p_j = p|\max C_{max}$. Analogously, we denote by $1|r_j, p_j = p|\max\sum C_j$ and $1|r_j, p_j = p|\max\sum U_j$ the two problems of maximizing total completion time and the number of tardy jobs, respectively.

In this section, we present a polynomial time algorithm to solve the problems $1|r_j, p_j = p|\max C_{max}$ and $1|r_j, p_j = p|\max\sum C_j$ and some properties of the problem $1|r_j, p_j = p|\max\sum U_j$.

7.1 Solution algorithm for the problems $1|r_j, p_j = p|\max C_{max}$ and $1|r_j, p_j = p|\max\sum C_j$

Let the jobs be numbered according to $r_1 \leq r_2 \leq \cdot \leq r_n$. Without loss of generality, we assume $r_1 = 0$.

The idea of the algorithm is as follows. One by one, we choose a job for the next position in a job sequence. We choose a job j according to its release date r_i in order to make the gap (an idle time) between the completion time of the previous job in a job sequence and r_j as large as possible but less than p so that no other job can be processed between the previous job and time r_j.

Algorithm 1.

```
1. Let $t := 0$ be the completion time of the last scheduled job, $N_u := N$ be the set of
   unscheduled jobs, and $\pi := ()$ be the job sequence.
2. FOR $l := 1$ TO $n$ DO
   2.1. Choose a job $j := argmax_{i \in N_u}\{r_i, t \le r_i < t + p\}$.
   2.2. IF there is no such job $j$ THEN
            Choose any job $j$ from the set $N_u$ with $r_i \le t$.
            IF there is no such job $j$ THEN
              $t := \min_{i \in N_u}\{r_i, t < r_i\}$. GOTO Step 2.1.
   2.2. $\pi := (\pi, j)$. $N_u := N_u \setminus \{j\}$. $t := C_j(\pi)$.
3. $\pi$ is an optimal schedule.
```

Algorithm 1 requires $O(n)$ operations, and $O(n \log n)$ operations are needed to order the jobs according to $r_1 \le r_2 \le \ldots \le r_n$.

Lemma 1. *Algorithm 1 constructs an optimal schedule for the problems* $1|r_j, p_j = p| \max C_{max}$ *and* $1|r_j, p_j = p| \max \sum C_j$.

Proof. Assume there exists an optimal sequence $\pi^* = (l, \pi_1, j, \pi_2)$, where the first job is l, but job j had to be chosen first by Algorithm 1. If $r_l \le r_j$, then for the job sequence $\pi = (j, \pi_1, l, \pi_2)$, we have $C_i(\pi) \ge C_i(\pi^*)$ for each $i \in N \setminus \{j\}$ and $C_{max}(\pi) \ge C_{max}(\pi^*)$.

If $r_j < r_l$, then the job sequence π^* is not active, since job 1 can be scheduled first from time $S_1 = 0$.

The rest of the proof can be done by induction. □

7.2 Properties of the problem $1|r_j, p_j = p| \max \sum U_j$

It is known that the problem $1|r_j| \max \sum U_j$ is NP-hard [5] but the special case $1|r_j = 0| \max \sum U_j$ can be solved in $O(n^2)$ time [2]. For this special case, there exists an optimal job sequence (G, W), where all jobs from the set G are on-time and all jobs from the set W are tardy and processed in order of non-decreasing due dates.

Next, we present some properties of the problem $1|r_j, p_j = p| \max \sum U_j$.

Property 12. *There exists an optimal job sequence, where the tardy job i precedes the on-time job j and $r_j < r_i$.*

For an illustration, see Fig.1(d), where jobs $i = 2$ and 4 are tardy and jobs $j = 1$ and 3 are on-time.

Property 13. *There exists an optimal job sequence, where the tardy job i precedes the tardy job j and $d_i < d_j$.*

For an explanation, see Fig.1(d), where $i = 2$, $d_2 = r_2 + p - 1$ and $j = 1$, $d_1 = r_2 + p$.

Property 14. *There exist instances for which Algorithm 1 does not construct an optimal schedule.*

For an explanation, see Fig.1(e), where job 2 is the only tardy job. In addition, in Fig.1(f), we show that a maximal total gap does not lead to an optimal schedule, where job 3 is the only tardy job.

Let there are k jobs j for which $r_j \in [0, p]$. Then Algorithm 1 can be modified to consider k schedules, where the first job in a job sequence is one of them. However, we suppose that there exists an instance for which all k schedules are not optimal.

To solve the problem, the following heuristic can be used: if in a job sequence π, job j is on-time and $S_j(\pi) > r_j$ holds, then we try to schedule it earlier. Additionally, we can consider a reason why a job is started in a sequence π not from its release date. Either it is tardy in the sequence π, or we need to try to schedule it earlier.

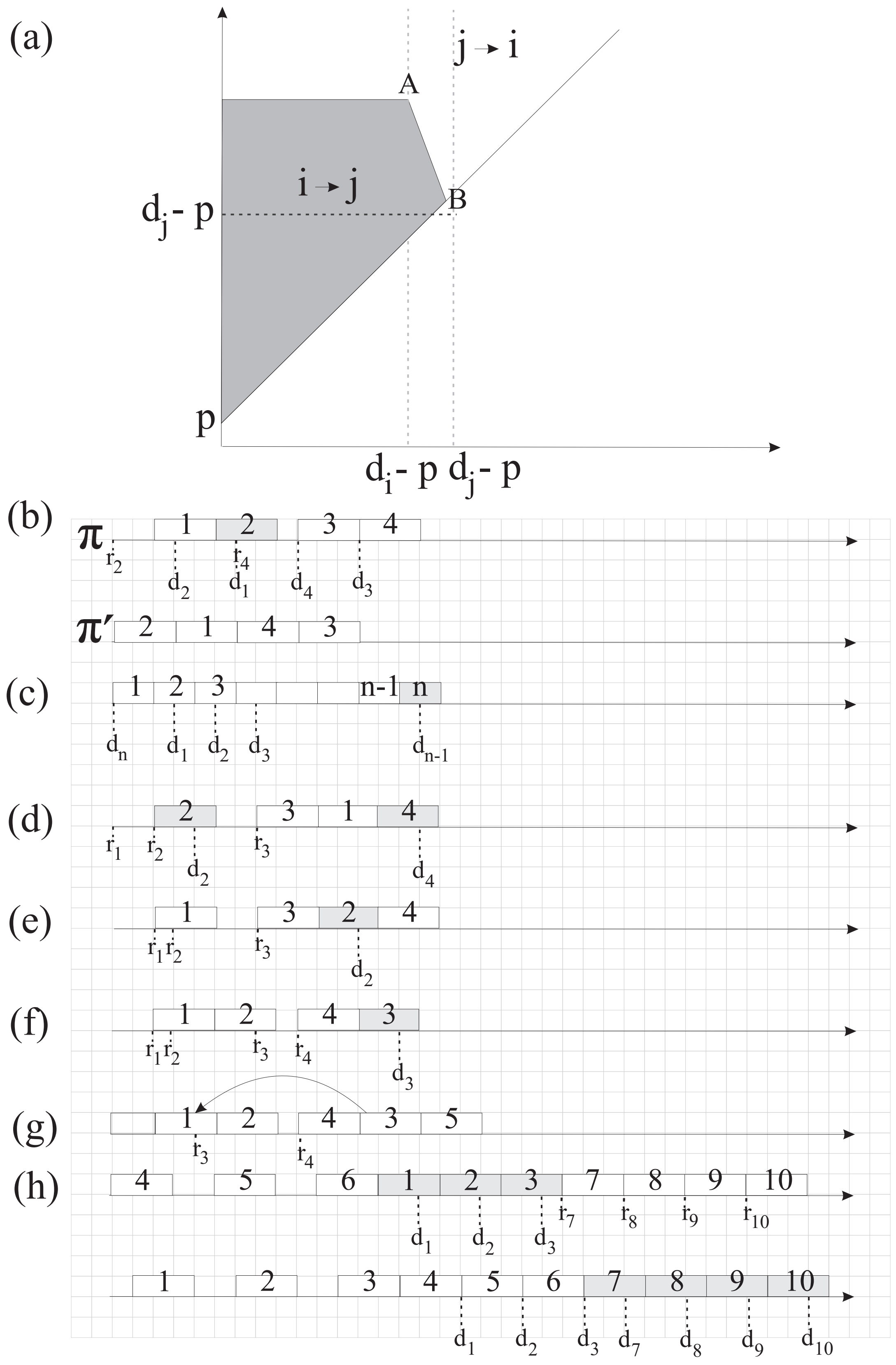

Figure 1: Examples

Property 15. *If in a job sequence π job j is on time and i is tardy and both of them are started not earlier than from time $t \geq \max\{r_i, r_j\}$ on, then there exists a job sequence π', where job j precedes job i and $\sum U_j(\pi') \geq \sum U_j(\pi)$.*

This property means that, if two jobs are processed not earlier than from time $t \geq \max\{r_i, r_j\}$ on, then an on-time job precedes a tardy one.

Property 16. *If in a job sequence π both jobs j and i are tardy, $d_j \leq d_i$ and both jobs are processed not earlier than from time $t \geq \max\{r_i, r_j\}$ on, then there is a job sequence π', where job j precedes job i and $\sum U_j(\pi') \geq \sum U_j(\pi)$.*

This property means that, if tardy jobs are processed not earlier than from time $t \geq \max\{r_i, r_j\}$ on, then they can be processed according to non-decreasing due dates.

Property 17. *If we choose a job from the two jobs j and i to be processed from time*

$$\max\{r_i, r_j\} \leq t \leq \min\{d_j - p, d_i - p\}$$

on (and the second will be processed later), then choose a job with the largest due date.

Property 18. *Let for the job sequence $\pi = (\pi_1, j_1, j_2, j_3, j_4, \pi_2, j_5, \pi_3)$, the following inequality hold:*

$$S_{j_2}(\pi) < r_{j_5} < C_{j_2}(\pi), \qquad C_{j_1} > r_{j_5} - p, S_{j_4} = r_{j_4} < r_{j_5} + 2p \quad \text{and} \quad C_{j_5}(\pi) < d_{j_5}.$$

Then for the job sequence $\pi' = (\pi_1, j_5, j_3, j_4, \pi_2, j_1, j_2, \pi_3)$, we have $F(\pi) \leq F(\pi')$.

For an explanation, see Fig.1(g). This property means that we have to exclude such situations by interchanging job j_5 with the jobs j_2, j_3.

Property 19. *There exists a schedule, where each pairwise interchange decreases the optimal function value, but the simultaneous realization of several ones increases it.*

For an explanation, see Fig.1(h). For the job sequence $\pi = (4, 5, 6, 1, 2, 3, 7, 8, 9, 10)$, each pairwise interchange, e.g., the interchanges $4 \leftrightarrow 1$, $5 \leftrightarrow 2$ etc., reduce the number of tardy jobs, but their simultaneous realization $\pi = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)$ increases it.

8 Concluding remarks

Some properties of the classical scheduling problem $1|r_j, p_j = p|\sum w_j T_j$ and several solution procedures were considered. The complexity status of the problem remains open. To the best of our knowledge, no pseudo-polynomial algorithms or approximation schemes are known for this problem.

We conjecture that problem $1|r_j, p_j = p|\sum w_j T_j$ is NP-hard, but for a proof a new way (another kind of a reduction from an NP-hard problem) has to be considered.

In addition, we formulated some new maximization scheduling problems with equal-length jobs and presented some of their properties.

Acknowledgements

This work has been partially supported by DAAD (Deutscher Akademischer Austauschdienst): 91695276 and by RFBR (Russian Foundation for Basic Research): 18-07-00656

References

[1] van den Akker J.M., Diepen G., Hoogeveen J.A. (2010). Minimizing Total Weighted Tardiness on a Single Machine with Release Dates and Equal-Length Jobs, Journal Scheduling, 13, 561 – 576.

[2] Aloulou M.A., Kovalyov M.Y., Portmann M.-C. (2007). Evaluation Flexible Solutions in Single Machine Scheduling via Objective Function Maximization: the Study of Computational Complexity, RAIRO Oper. Res. 41, 1 – 18.

[3] Baptiste, P. (2000). Scheduling Equal-Length Jobs on Identical Parallel Machines. Discrete Applied Mathematics, 103(1–3), 21 – 32.

[4] Gafarov E.R., Lazarev A.A. and Werner F. (2012). Transforming a Pseudo-Polynomial Algorithm for the Single Machine Total Tardiness Maximization Problem into a Polynomial One. Annals of Operations Research. 196(1), 247 – 261.

[5] Gafarov E.R., Lazarev A.A. and Werner F. (2013). Single Machine Total Tardiness Maximization Problems: Complexity and Algorithms. Annals of Operation Research. 207, 121 – 136.

[6] Graham, R.L., Lawler, E.L., Lenstra J.K., Rinnooy Kan A.H.G. (1979). Optimization and Approximation in Deterministic Machine Scheduling: a Survey. Ann. Discrete Math. 5, 287 – 326.

[7] Du J., Leung J. Y.-T. (1990). Minimizing Total Tardiness on One Processor is NP-hard. Math. Oper. Res. 15, 483 – 495.

[8] Held, M. and Karp, R. M. (1962). A Dynamic Programming Approach to Sequencing Problems. J. Soc. Indust. Appl. Math. 10, 196 – 210.

[9] Kravchenko S.A., Werner F. (2011). Parallel Machine Problems with Equal Processing Times: a Survey, Journal Scheduling, 14, 435 – 444.

[10] Leung, J. Y.-T. (2004). Handbook of Scheduling. New York: Chapmann and Hall/CRC.

[11] Verma, S., Dessouky, M. (1998). Single-Machine Scheduling of Unit-Time Jobs with Earliness and Tardiness Penalties. Mathematics of Operations Research, 23(4), 930 – 943.