Available online at www.sciencedirect.com
SCIENCE DIRECT®

Computers & Operations Research 32 (2005) 683–705

computers & operations research

www.elsevier.com/locate/dsw

Machine scheduling with earliness, tardiness and non-execution penalties

Refael Hassin*, Mati Shani[1]

Department of Statistics and Operations Research, School of Mathematical Sciences, Tel-Aviv University, Tel-Aviv 69978, Israel

Abstract

The study of scheduling problems with earliness–tardiness (E/T) penalties is motivated by the just-in-time (JIT) philosophy, which supports the notion that earliness, as well as tardiness, should be discouraged. In this work, we consider several scheduling problems. We begin by generalizing a known polynomial time algorithm that calculates an optimal schedule for a given sequence of tasks, on a single machine, assuming that the tasks have distinct E/T penalty weights, distinct processing times and distinct due dates. We then present new results to problems, where tasks have common processing times. We also introduce a new concept in E/T scheduling problems, where we allow the non-execution of tasks and consequently, are penalized for each non-executed task. The notion of task's non-execution, coincides with the JIT philosophy in that every violation or a breach of an agreement, should be penalized. We develop polynomial time algorithms for special cases in E/T scheduling problems with non-execution penalties.
© 2003 Elsevier Ltd. All rights reserved.

1. Introduction

The study of scheduling problems with earliness and tardiness (E/T) penalties is relatively recent. For many years, the research of scheduling problems focused on minimizing measures such as mean flow-time, maximum tardiness, and makespan, all non-decreasing in the completion times of tasks. For these measures, delaying execution of tasks results in a higher cost. However, the current emphasis in industry on the just-in-time (JIT) philosophy, which supports the notion that earliness, as well as tardiness, should be discouraged, has motivated the study of scheduling problems in which tasks are preferred to be ready just at their respective due dates, and both early and tardy products are penalized.

* Corresponding author.
E-mail addresses: hassin@post.tau.ac.il (R. Hassin), kermat@inter.net.il (M. Shani).
[1] Present address: 59 Ben-Gurion St., Kfar Saba 44204, Israel.

0305-0548/$ - see front matter © 2003 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2003.08.012

In this paper, we consider several E/T scheduling problems. We are given a set $\tilde{T}=\{T_1,\dots,T_N\}$ of tasks. Task T_i has an integer processing time $p_i > 0$ and a target starting time $a_i \geqslant 0$ (or equivalently, a due date d_i, where $d_i \geqslant p_i$). There are m parallel machines, $\{M_1,\dots,M_m\}$. Our notation follows that of Garey et al. [1].

A *solution* for $\tilde{T}$, is an *assignment* of each task T_i to a machine M_j and a *schedule* corresponding to that assignment, which determines a starting time s_i for T_i on M_j. The scheduling of starting times, must satisfy that no two tasks assigned to the same machine overlap in their execution time, and that the tasks are to be scheduled non-preemptively; once started, a task T_i must be executed to its completion, p_i time units later.

A *sequence* defines the order in which, tasks are to be processed, whereas a *schedule* is a sequence with starting times calculated for each task. We assume nonnegative earliness penalty weight α_i and nonnegative tardiness penalty weight β_i, associated with task T_i. T_i incurs the earliness penalty $\alpha_i(a_i - s_i)$ if $s_i < a_i$ and it incurs the tardiness penalty $\beta_i(s_i - a_i)$ if $s_i > a_i$. We define $e_i = \max\{0, a_i - s_i\} \equiv (a_i - s_i)^+$ and $t_i = \max\{0, s_i - a_i\} \equiv (s_i - a_i)^+$ and thus, the penalty incurred by T_i is $\alpha_i e_i + \beta_i t_i$. The overall cost of a solution, which we wish to minimize, is the sum of the individual penalties, i.e., $\sum_{i=1}^{N} (\alpha_i e_i + \beta_i t_i)$. We refer to this cost function, as the Total Weighted Earliness and Tardiness problem (TWET—see [2]). In general, we denote the *cost* of solution $\tilde{T}$ by $cost(\tilde{T})$.

In Section 4, we introduce a new type of penalty to the E/T scheduling problems. Assume we are allowed to not-execute one or more of the tasks. Denote by γ_i the penalty incurred if T_i is not executed (processed). Thus, a modified TWET problem, is to minimize $\sum_{i=1}^{N} [(1-x_i)(\alpha_i e_i+\beta_i t_i)+x_i\gamma_i]$ where $x_i = 0$ if T_i is executed and $x_i = 1$, otherwise. The notion of task's non-execution, fits the JIT philosophy in that every violation or a breach of an agreement, should be penalized.

2. Literature review

The research can be classified into two main categories, which reflect the due date specifications:

1. Problems with *common due date* $\{d_i = d\}$, which we denote *CDD*.
2. Problems with *distinct due dates* $\{d_i\}$, which we denote *DDD*.

The problems can be further categorized with respect to other criteria such as, number of machines and cost functions.

2.1. Common due date problems

We distinguish between *restricted* and *unrestricted* problems. The problem is restricted, when no task can start before time zero. The problem is unrestricted, when d is large enough for example, $d \geqslant \sum_{i=1}^{N} p_i$. The restricted problems are often more difficult to solve. Baker and Scudder [3] and Gordon et al. [2] give a comprehensive review on the restricted and the unrestricted problems. In this work, we consider only unrestricted problems.

Baker and Scudder [3] state three necessary properties, that any optimal solution to CDD problems must satisfy:

Property 1. No idle time is inserted.

Property 2. The sequence is *V-shaped*; that is, early tasks are sequenced in non-increasing order of the p_i/α_i ratio and tardy tasks are sequenced in non-decreasing order of the p_i/β_i ratio.

Property 3. The bth task in the sequence, completes precisely at the due date, where b is the smallest integer satisfying the inequality $\sum_{i=1}^{b} \alpha_i \geqslant \sum_{i=b+1}^{N} \beta_i$.

An important special case, is the unrestricted problem with $\alpha_i = \beta_i = 1$ for $1 \leqslant i \leqslant N$ (unit E/T penalty weights). In this problem, which we refer to as the Mean Absolute deviation problem (MAD—see [3]), we minimize $\sum_{i=1}^{N} (e_i + t_i)$. The analysis of this problem is due to Kanet [4], Hall [5] and Bagchi et al. [6]. Kanet [4] and Hall [5] introduce an $O(N^2)$ time algorithm, which solves MAD. Bagchi et al. [6] present a modification which reduces the complexity to an $O(N \log N)$ time algorithm.

The Weighted Earliness and Tardiness problem (WET—see [2]) is a generalization of MAD, where $\alpha_i = \alpha$ and $\beta_i = \beta$ for $1 \leqslant i \leqslant N$. Thus, we minimize $\sum_{i=1}^{N} (\alpha e_i + \beta t_i)$. Bagchi et al. [7], present an $O(N \log N)$ time algorithm for WET.

Recall the three properties that any optimal solution of a CDD problem must hold. Property 1 states that there is no idle time in any optimal solution and property 3 states that a certain task completes precisely at the due date. Thus, the only factor that determines the starting time (or equivalently, the due date) of each task, is its relative position in the sequence. With some algebraic manipulation, the cost function can be formulated in terms of *positional weights*. Denote by B the set of strictly early tasks, i.e., $B \equiv \{T_i | s_i < a\}$ and denote by A the set of tardy tasks, i.e., $A \equiv \{T_i | s_i \geqslant a\}$. Also denote by $B(i)$ the ith task processed in B and by $A(i)$ the ith task processed in A. Thus, the cost function of MAD can be rewritten as $\sum_{i=1}^{|B|} i p_{B(i)} + \sum_{i=1}^{|A|} (|A| - i) p_{A(i)}$ and the cost function of WET can be rewritten as $\sum_{i=1}^{|B|} \alpha i p_{B(i)} + \sum_{i=1}^{|A|} \beta(|A| - i) p_{A(i)}$, where $|B| = \lfloor N\beta/(\alpha+\beta) \rfloor$ and $|A| = \lceil N\alpha/(\alpha+\beta) \rceil$ (where $|A| + |B| = N$). The positional weight is the term $\alpha i p_{B(i)}$ or $\beta(|A| - i) p_{A(i)}$, which gives the contribution of a task to the cost function conditioned on its position in the solution.

Hall and Posner [8] consider the TWET problem with $\alpha_i = \beta_i = w_i$ for $1 \leqslant i \leqslant N$, where w_i is the symmetric E/T penalty weight of T_i. We will refer to this problem as the Symmetric TWET (STWET—see [2]) and the objective is to minimize $\sum_{i=1}^{N} w_i(e_i + t_i)$. They prove that the problem is NP-hard and provide an $O(N \sum_{i=1}^{N} p_i)$ pseudopolynomial time Dynamic programming (DP) algorithm, which solves STWET. They also present special cases in which the STWET problem is polynomial. De et al. [9] and Jurisch et al. [10], provide another special case in which STWET is polynomially solvable. Kovalyov and Kubiak [11], present a fully polynomial approximation scheme to STWET.

Szwarc [12] considers the TWET problem in which tasks have *agreeable ratios*, i.e., $p_i/\alpha_i < p_j/\alpha_j \Rightarrow p_i/\beta_i < p_j/\beta_j \ \forall i, j$ where $1 \leqslant i \neq j \leqslant N$.

In problems with almost equal due dates (AEDD), the due date of each T_i maintains $d_i \in [d, d + p_i]$ for some given large d (i.e., the unrestricted version). Hoogeveen and van de Velde [13] present an $O(N^2)$ time DP algorithm to the AEDD WET problem and a pseudopolynomial $O(N^2 \sum_{i=1}^{N} p_i)$ time DP algorithm to the AEDD STWET problem.

2.2. Distinct due dates problems

The three properties, which any optimal solution for the CDD problems must hold, do not necessarily hold in the DDD problem. James and Buchanan [14] redefine these properties for the DDD scheduling problems. They define a *block* as the maximal set of tasks that are scheduled contiguously without idle time inserted between them. Any optimal solution to a DDD problem, must satisfy the three properties with simple modifications, with respect to blocks of tasks.

Garey et al. [1] prove that even the single machine DDD MAD problem, is NP-hard. To the special case where tasks must be processed in a given order (sequence), they present an $O(N \log N)$ time optimal scheduling algorithm, which we refer to as *Algorithm GTW*. They also prove that even if the tasks are not pre-ordered but have a common length of processing time $p \geqslant 0$, sequencing the tasks with the property $a_i \leqslant a_{i+1}$ for $1 \leqslant i < N$ and applying Algorithm GTW to this sequence, results in an optimal solution. There are other scheduling algorithms that are described in literature, such as of Davis and Kanet [15] or, of Szwarc and Mukhopadhyay [16] (both papers consider the DDD TWET problem). All algorithms are polynomial and their computational effort is at most $O(N^2)$.

The difficult phase however, is the sequencing. Fry et al. [17,18], Kim and Yano [19], Ow and Morton [20], Lee and Choi [21] and James and Buchanan [14] present heuristic procedures for sequencing.

One of the most innovative works on the DDD TWET problem is of Verma and Dessouky [22]. They assume that $p_i = 1$ for $1 \leqslant i \leqslant N$. Note that if $d_i \in \mathbb{N}$ for $1 \leqslant i \leqslant N$, the problem can be formulated as an assignment problem; the difficulty arises when the due dates are fractional. They formulate the problem as an integer linear programming (ILP) and assume the following *dominance condition* over the penalty weights: Tasks are indexed such that both $\alpha_1 \leqslant \alpha_2 \leqslant \cdots \leqslant \alpha_N$ and $\beta_1 \leqslant \beta_2 \leqslant \cdots \leqslant \beta_N$ hold. If the "$\leqslant$" signs are replaced by the "$<$" signs, they refer to it as the *strict dominance condition*. They prove that if the set of tasks satisfies the dominance condition, there exists an integral extremal optimal solution to the linear programming (LP) relaxation of the ILP formulation and thus, the problem is solved in polynomial time. Moreover, if the tasks satisfy the strict dominance condition, *any* extremal optimal solution to the LP relaxation is integral. They also present four cases in which the dominance condition is met. Note that the complexity of the DDD TWET problem, with a common processing time, general due dates and general E/T penalty weights, is still an open question.

Two important papers are worthwhile noting. The first, by Kanet and Sridharan [23], is a review of problems with inserted idle time (IIT). They also define a taxonomy of the IIT problems. The second, by Gordon et al. [2], is a recent and a thorough survey of the CDD assignment and scheduling problems. They summarize most of the work published in the previous decade. Although their survey emphasizes on CDD problems, they also note important works, which consider DDD problems.

2.3. Parallel machines

Machines are *identical*, if they operate at the same speed and thus, p_j is the fixed processing time for T_j. Machines are *uniform*, if machine M_i has a distinct speed u_i and thus, $p_{ij} = p_j/u_i$ is the processing time for T_j on M_i. Machines are *unrelated*, if machine M_i has a distinct task-dependent speed u_{ij} for processing T_j and thus, $p_{ij} = p_j/u_{ij}$ is the processing time for T_j on M_i.

Sundararaghavan and Ahmed [24], Hall [5] and Emmons [25] solve the CDD MAD problem, with parallel identical machines. Hall [5] presents an $O(N^2)$ time algorithm and Sundararaghavan and Ahmed [24] and Emmons [25], present an $O(N \log N)$ time algorithms. Emmons [25] also considers the CDD WET problem, with parallel identical and parallel uniform machines and the $O(N \log N)$ time algorithm is extended to these problems. Kubiak et al. [26] prove that the CDD MAD problem and the problem of minimizing the mean flow time with parallel uniform machines, are equivalent. They also prove that in the case of parallel unrelated machines, the problem can be reduced to a transportation problem.

Webster [27] proves that the CDD STWET problem, which is NP-hard in the case of single-machine, is strongly NP-hard for parallel identical machines. Thus, the CDD TWET problem is also strongly NP-hard for parallel identical machines. Chen and Powell [28] present a Branch-and-Bound algorithm for the CDD TWET problem with parallel identical machines.

2.4. Problem summary

To conclude this review, we summarize the E/T scheduling problems described so far and the problems we shall further address. We adopt the method for problem classification of Lageweg et al. [29] and present in Table 1 the "maximal easy problems" (the most general cases of polynomially solvable problems) and the "minimal hard problems" (the most simple cases of NP problems). Other problems are cited below the table and are related to specific problems described in the table. To simplify, we use the standard three-field notation $a|b|c$ used for scheduling problems (Lawler et al. [30]), where a describes the machine environment, b describes the schedule and task characteristics and c describes the cost function. We use P, Q and R to denote, respectively, parallel identical, uniform and unrelated machines, with an index m denoting a fixed (given) number of m machines. d, d^{res}, $[d, d + p_i]$ and d_i denote, respectively, CDD, restricted CDD, AEDD and DDD problems. p and p_i denote, respectively, common and distinct processing times. γ and γ_i denote, respectively, common and distinct non-execution penalty weights. Unless clearly presented in the b-field, the non-execution penalties are not considered. GENERAL denotes common, general, non-decreasing E/T penalty functions or, $\sum_{i=1}^{N} [h(e_i) + g(t_i)]$. CONVEX denotes common, convex, non-negative and not necessarily symmetrical E/T penalty function P, where $P(0) = 0$ and $P(x) > 0\ \forall x \neq 0$. NPC and SNPC denote, respectively, NP-hard and strongly NP-hard problems and "**?**" denotes that the complexity of the problem is unknown.

Table 1
Problem summary

	Problem	Complexity	References; Algorithms	Remarks
1	$1\|d^{\text{res}}, p_i\|$MAD	NPC	[31], [32]; $O(N \sum_{i=1}^{N} p_i)$	1,2,3,4
2	$1\|d, p_i\|$STWET	NPC	[8]; $O(N \sum_{i=1}^{N} p_i)$	5,6
3	$1\|d, p_i\|$GENERAL	NPC	[33]	7,8
4	$1\|[d, d + p_i]\|$WET	P	[13]; $O(N^2)$	9,10
5	$1\|d_i, p_i\|$MAD	NPC	[1]	11,12
6	$Q_m\|d, p_i\|$WET	P	[25]; $O(N \log N)$	13,14,15
7	$P_m\|d, p_i\|$STWET	SNPC	[27]	16
8	$P_m\|d_i, p\|$CONVEX	P	[Algorithm 11]; $O(N \log N)$	17,18
9	$1\|d_i, p, \gamma_i\|$CONVEX	**?**	[Algorithm 16]; $O(N^2 p)$	19

Table 1 (*continued*)

	Problem	Complexity	References; Algorithms	Remarks
10	$1\|d,p,\gamma\|$STWET	P	[Algorithm 18]; $O(N\log N)$	20
11	$1\|d,p_i,\gamma_i\|$WET	**?**		21
12	$1\|d,p,\gamma_i\|$WET	P	[Algorithm 23]; $O(N)$	
13	$1\left\|d,\frac{p_i}{\gamma_i}=r\right\|$WET	P	[Algorithm 25]; $O(N\log N)$	22

Note: 1. With the restriction that the solution must start at time zero, an efficient heuristic is given in [24]. Enumeration procedures are given in [6] and [34]. An $O(N\log N)$ time $\frac{4}{3}$-approximation algorithm is presented in [35].
2. To the version of WET ($1|d^{\text{res}}, p_i|$WET), an enumeration procedure is presented in [7].
3. The unrestricted version of MAD ($1|d, p_i|$MAD), is considered in [4–6] and $O(N\log N)$ time algorithms are presented. The optimal solution is not unique. The problem reduces to a two parallel identical machines, mean flow time problem.
4. To the version of unrestricted WET ($1|d, p_i|$WET), an $O(N\log N)$ time algorithm is presented in [7]. The uniqueness of the optimal solution depends upon the E/T penalty weights. The problem reduces to a two parallel nonidentical machines, mean flow time problem.
5. The problem is polynomial in special cases (see [8–10] and [Algorithm 18]). A FPTAS approximation is given in [11].
6. The version of TWET ($1|d, p_i|$TWET), is considered in [12]. Pseudopolynomial time algorithms are given to the special case, where tasks have agreeable ratios of E/T penalty weights.
7. In [36], an $O(N^3)$ time greedy heuristic is presented. For a common, convex E/T penalty function, the algorithm has a performance guarantee of $e^{-1}\sim 0.36$.
8. In [37], the E/T penalty weights consist of two parts, one being a variable cost (a function of $|s_i-a_i|$), while the other being a fixed cost incurred once a task is early/tardy. The optimal solution is "W-shaped" if the tasks are agreeably weighted and a pseudopolynomial $O(N^2\sum_{i=1}^{N}p_i)$ time DP algorithm is presented. Under certain conditions, a faster, pseudopolynomial $O(N\sum_{i=1}^{N}p_i)$ time DP approximation algorithm, with a relative error that tends to 0, as N increases, is applicable. In more restrictive special cases, the latter algorithm also calculates the optimal solution.
9. The $O(N^2)$ time algorithm is applicable to E/T problems other than the AEDD, as long as the cost function can be formulated in terms of positional weights, the optimal schedule has no idle time, and there exists an optimal schedule that can be characterized by the task that completes on time and the set of early tasks.
10. The version of STWET ($1|[d,d+p_i]|$STWET) is considered in [13] and a pseudopolynomial $O(N^2\sum_{i=1}^{N}p_i)$ time DP algorithm is presented.
11. In [1], the problem is proved to be polynomially solvable if either: (1) there is a given sequence of tasks, (2) the tasks have a common length of processing times. In the special case of a given sequence, an $O(N\log N)$ time algorithm is presented in [1] and three generalizations to the algorithm are presented in the cases of: (1) each task has a given target window of starting time, (2) tasks are given with consecutive processing constraints, and (3) the cost function is STWET. Heuristic solution procedures are given in [18,19].
12. The version of the TWET problem ($1|d_i, p_i|$TWET), is polynomially solvable if there is a given sequence of tasks and an $O(N\log N)$ time algorithm is given in [Section 3.1]. For a given sequence of tasks, other scheduling algorithms are presented in [15,16]. If $p=1$, TWET is also polynomially solvable and even the parallel uniform problem ($Q_m|d_i, p=1|$GENERAL), can be formulated as an assignment problem. The problem is also polynomially solvable, if there is a common processing time $p\neq 1$, general d_i and the E/T penalty weights satisfy the dominance condition and it is solved through LP in [22]. In [22], they also present special cases of E/T penalty weights in which the dominance condition is met and thus, the problem is polynomially solvable. Heuristic solution procedures are given in [14,17,21]. The complexity of the problem, with a common processing time $p\neq 1$, general d_i and general E/T penalty weights, which do not satisfy the dominance condition, is still an open question.
13. The version of MAD ($Q_m|d, p_i|$MAD), is considered in [25,26]. In [26], the problem is proved to be equivalent to a mean flow time problem.
14. The version of MAD with parallel identical machines ($P_m|d, p_i|$MAD), is considered in [24,5,25] and $O(N\log N)$ time algorithms are presented.
15. The version of MAD with unrelated machines ($R_m|d, p_i|$MAD), is considered in [26] and it reduces to a transportation problem.
16. The version of TWET ($P_m|d, p_i|$TWET), is considered in [28] and a Branch-and-Bound procedure is developed.
17. For the version of the single-machine problem ($1|d_i, p|$CONVEX), an $O(N\log N)$ time algorithm is presented in [Theorem 4].
18. For the version of the two parallel identical machines problem ($P_2|d_i, p|$CONVEX), an $O(N\log N)$ time algorithm is presented in [Algorithm 8].
19. The $1|d_i, p, \gamma_i|$TWET problem, assuming that the E/T penalty weights satisfy the dominance condition, is solved polynomially in Section 4.2.
20. The complexity is unknown for the $1|d, p, \gamma_i|$STWET problem. The complexity is unknown also for the $1|d, p_i=w_i, \gamma|$STWET problem (a case solved polynomially in [8], when $\gamma=\infty$).
21. The case of *agreeably reversed* tasks is considered in Section 4.4. [Algorithm 21] yields the optimal solution in an $O(N\log N)$ time.
22. The complexity is unknown for the $1|d, p_i/\gamma_i|$WET problem (when the ratio p_i/γ_i is not common), even if (1) p_i and γ_i are agreeable, i.e., $p_1\leqslant\cdots\leqslant p_N$ and $\gamma_1\leqslant\cdots\leqslant\gamma_N$, or (2) if the tasks can be ordered in a non-decreasing ratio of processing times and non-execution penalty weights, i.e., $p_1/\gamma_1\leqslant\cdots\leqslant p_N/\gamma_N$.

It is interesting to note that under unrestricted due dates and linear E/T penalty functions, all NPC scheduling problems described in this work, have at least two degrees of freedom in terms of task parameters, whereas polynomial scheduling problems, have a single degree of freedom. For example, in the NPC problem $1|d, p_i|$STWET (Problem 2), tasks have distinct processing times and distinct (although symmetric) E/T penalty weights. On the other hand, in the polynomial problem $1|d, p, \gamma_i|$WET (Problem 12), tasks only have distinct non-execution penalty weights. Therefore, it is plausible to suggest that open problems such as $1|d_i, p, \gamma_i|$TWET and $1|d, p_i, \gamma_i|$WET, which have two or more degrees of freedom, are NPC problems.

In the following sections, we present some new results to specific E/T scheduling problems. In Section 3 we present generalizations of Algorithm GTW to single and parallel machine problems. In Section 4 we present E/T scheduling problems, which include a non-execution penalty.

3. Generalizations of Algorithm GTW

Garey et al. [1], consider the $1|d_i, p_i|$MAD problem under a given sequence of tasks and present an $O(N \log N)$ time scheduling algorithm (Algorithm GTW). They also prove that if the tasks are not pre-ordered but, have a common length of processing time $p \geqslant 0$, it is optimal to sequence the tasks such that $a_i \leqslant a_{i+1}$ for $1 \leqslant i < N$. The solution generated by the algorithm to this optimal sequence, is a minimum cost schedule in which $s_i \leqslant s_{i+1}$ for $1 \leqslant i < N$. We start this section by presenting Algorithm GTW after modifying it to handle the TWET problem. We then show that the special case of common processing times can be generalized under extended conditions, allowing the tasks to have a common non-negative convex—not necessarily symmetrical—penalty function. Finally, we present a *zigzagging algorithm* extension to handle parallel machines.

3.1. Modified Algorithm GTW

In this subsection, we consider the optimal scheduling of a given sequence of tasks, under the TWET cost function. Let S_n be the partial solution computed for the first n tasks. We use the definition of a *block* as was stated in Section 2.2. We denote the tasks in a block by $\{T_{i_0}, \ldots, T_{i_1}\}$ and they obtain $s_i + p_i = s_{i+1}$ for $i_0 \leqslant i < i_1$, $s_{i_0-1} + p_{i_0-1} < s_{i_0}$ (or $i_0 = 1$) and $s_{i_1} + p_{i_1} < s_{i_1+1}$ (or $i_1 = N$). Assume that there are t blocks $B_1, \ldots, B_t$ in S_n. We define a partition of each block B_j into two subsets of tasks, Dec(j) and Inc(j), as follows: $\text{Dec}(j) \equiv \{T_i \in B_j | s_i > a_i\}$ and $\text{Inc}(j) \equiv \{T_i \in B_j | s_i \leqslant a_i\}$. The idea is that if $T_i \in \text{Dec}(j)$, reducing s_i (but not earlier than a_i) decreases the discrepancy of T_i, whereas if $T_i \in \text{Inc}(j)$, reducing s_i increases its discrepancy.

We define $I(j) = \sum_{T_i \in \text{Inc}(j)} \alpha_i$ and $D(j) = \sum_{T_i \in \text{Dec}(j)} \beta_i$. As a way of representing the blocks, we denote by first(j) and last(j) the smallest and largest indices, respectively, of the tasks in B_j.

Our initial solution S_1, simply schedules T_1 to start at a_1. In general, given S_n, we schedule T_{n+1} as follows: If $s_n + p_n \leqslant a_{n+1}$, we schedule T_{n+1} to start at a_{n+1}. Here, T_{n+1} has no discrepancy, and S_n and S_{n+1} have the same cost. If on the other hand $s_n + p_n > a_{n+1}$, we begin by scheduling T_{n+1} to start at $s_n + p_n$. Now T_{n+1} has a positive discrepancy and both, the last block B_t and its corresponding set Dec(t), have gained T_{n+1} as a member. A key property that the algorithm maintains is that, for each $B_j \in S_n$, either $D(j) < I(j)$ or $s_{\text{first}(j)} = 0$ (i.e., first(j) = 1). If $s_n + p_n > a_{n+1}$, depending upon the values of the E/T penalty weights, scheduling T_{n+1} to start at $s_n + p_n$ may result in $D(t) \geqslant I(t)$.

If $s_{\text{first}(t)} = 0$ or $D(t) < I(t)$, we take no further action; the current solution (schedule) is S_{n+1}. On the other hand, if $s_{\text{first}(t)} > 0$ and $D(t) \geqslant I(t)$, we can shift B_t earlier, without increasing the total E/T penalties of this solution. We may even *decrease* the total penalty had we had $D(t) > I(t)$ prior to the shift. The shift is performed, until one of the following three cases occur:

1. $s_{\text{first}(t)}$ becomes zero. In such a case, we *stop* the shift. Or,
2. for some $T_i \in B_t$, s_i becomes equal to a_i. In such a case, we recalculate $D(t)$ and $I(t)$. One of two cases occur:
   (a) $D(t) < I(t)$. In such a case, we *stop* the shift. Or,
   (b) $D(t) \geqslant I(t)$. In such a case, we continue the shift until one of the three cases occur (i.e., 1, 2 or 3). Or,
3. $s_{\text{first}(t)}$ becomes equal to $s_{\text{last}(t-1)} + p_{\text{last}(t-1)}$. In such a case, we recalculate the values of $D(t-1)$ and $I(t-1)$ of the unified block B_{t-1}. One of two cases occur:
   (a) $D(t-1) < I(t-1)$. In such a case, we *stop* the shift. Or,
   (b) $D(t-1) \geqslant I(t-1)$. In such a case, we continue the shift with B_{t-1} until one of the three cases occur (i.e., 1, 2 or 3).

The resulting solution is S_{n+1}.

Case (1) can only occur if $t = 1$; if it occurs, block B_t (or actually B_1) cannot be moved earlier because it now starts at time zero. In case (2), T_i is transferred from Dec(t) to Inc(t). If $D(t) < I(t)$, further shifting of B_t will only increase the cost of the solution. If on the other hand $D(t) \geqslant I(t)$, further shifting is desirable or at least, not harmful. In case (3), B_t is merged into B_{t-1}. As in case (2), the values of $D(t-1)$ and $I(t-1)$ in the merged block, depend upon the specific values of the E/T penalty weights. Therefore, by the properties mentioned earlier, if either $D(t-1) < I(t-1)$ or $s_{\text{first}(t-1)} = 0$, further shifting of B_{t-1} will either increase the cost of the solution or illegally start a task before time 0. If on the other hand $D(t-1) \geqslant I(t-1)$ and $s_{\text{first}(t-1)} > 0$, further shifting is desirable.

The modified algorithm GTW begins with S_1 and applies the above construction to form $S_2, S_3, \ldots, S_N$. The observations above suggest why the algorithm should work. When forming S_N, the algorithm terminates with an optimal solution.

3.2. Convex E/T penalties

Algorithm GTW can be further generalized to the case, where each T_i has a distinct, convex E/T penalty function P_i (not necessarily symmetrical) over $\mathbb{R}$, such that $P_i(0) = 0$ and $P_i(x) > 0\ \forall x \neq 0$ for $i = 1, \ldots, N$. We define the earliness penalty of T_i as $P_i(a_i - s_i)$ and the tardiness penalty as $P_i(s_i - a_i)$. We also define $I(j)$ and $D(j)$ as $I(j) = \sum_{T_i \in \text{Inc}(j)} P_i'(a_i - s_i)$ and $D(j) = \sum_{T_i \in \text{Dec}(j)} P_i'(s_i - a_i)$, where P_i' is the derivative of P_i. Thus, the decision of whether a shift will take place, is determined through the marginal values of the E/T penalties in Inc(j) and in Dec(j). Case (2) of the algorithm is no longer valid, since the desirability of a shift does not necessarily change in time points of target starting times but, in any point along the time scale for which the balance between $I(j)$ and $D(j)$ is changed. Note however, that the special case of common processing times cannot be solved with this modification of the algorithm and in order to solve the problem, we need to assume that $P_i = P$ for $i = 1, \ldots, N$ (a common, convex E/T penalty function).

3.3. Common processing times

Garey et al. [1] prove that in the $1|d_i, p|$MAD problem it is optimal to execute the tasks in a nondecreasing order of their target starting times. We now prove that this sequence is also optimal for the more general problem of $1|d_i, p|$CONVEX. We note that the $1|d_i, p|$TWET problem is polynomially solvable under the dominance condition on the E/T penalty weights, due to Verma and Dessouky [22]. Although we assume a common penalty function, which is a stricter demand than the dominance condition, we allow any convex function and the time complexity of our algorithm is smaller than that of Verma and Dessouky [22].

For the following properties, let P be a convex function over $\mathbb{R}$, where $P(0) = 0$ and $P(x) > 0$ $\forall x \neq 0$.

Property 1. $\forall x_1, x_2, x_3 \in \mathbb{R}$ such that $x_1 \leqslant x_2 \leqslant x_3$,

$$\frac{P(x_3) - P(x_2)}{x_3 - x_2} \geqslant \frac{P(x_2) - P(x_1)}{x_2 - x_1}.$$

Property 2. 1. $\forall x_1, x_2, x_3 \in \mathbb{R}$ such that $0 < x_1, x_2, x_3$,

$$P(x_2) + P(x_1 + x_2 + x_3) \geqslant P(x_1 + x_2) + P(x_2 + x_3).$$

2. $\forall x_1, x_2, x_3 \in \mathbb{R}$ such that $x_1, x_2, x_3 < 0$,

$$P(x_2) + P(x_1 + x_2 + x_3) \geqslant P(x_1 + x_2) + P(x_2 + x_3).$$

Property 3. 1. $\forall x_1, x_2 \in \mathbb{R}$ such that $0 < x_1, x_2$,

$$P(x_1 + x_2) \geqslant P(x_1) + P(x_2).$$

2. $\forall x_1, x_2 \in \mathbb{R}$ such that $x_1, x_2 < 0$,

$$P(x_1 + x_2) \geqslant P(x_1) + P(x_2).$$

Theorem 4. *Consider the* $1|d_i, p|$CONVEX *problem and assume that* $0 \leqslant a_1 \leqslant \cdots \leqslant a_N$. *Then, there exists a minimum cost solution in which* $s_i \leqslant s_{i+1}$ *for* $1 \leqslant i < N$.

Proof. Let S be a solution such that for some $i < j$, $a_i < a_j$ but $s_i > s_j$. We will show that T_i and T_j can be interchanged in S, without increasing its cost. The theorem then follows by induction on the number of interchanges needed to put the tasks in order of their indices (and their target starting times). Since T_i and T_j have equal lengths of processing time, interchanging their starting times results in a feasible schedule. Note that the starting times of all other tasks remain unchanged and thus, their corresponding E/T penalties remain unchanged. There are six cases to consider, depending on the relative ordering of a_i, a_j, s_i and s_j. For example, assume that $s_j < s_i \leqslant a_i < a_j$. We define for convenience $a = s_i - s_j$, $b = a_i - s_i$ and $c = a_j - a_i$. In S, the E/T penalty of T_i and T_j is $P(b) + P(a + b + c)$, whereas after the interchange, the penalty is $P(a + b) + P(b + c)$. Using Property 2 we have $P(b) + P(a + b + c) \geqslant P(a + b) + P(b + c)$ and thus, the E/T penalties incurred by T_i and T_j do not increase. Similarly, using Properties 2 and 3 we prove the other five possible relative orderings $a_i < a_j \leqslant s_j < s_i$, $s_j \leqslant a_i \leqslant s_i \leqslant a_j$, $s_j \leqslant a_i < a_j \leqslant s_i$, $a_i \leqslant s_j < s_i \leqslant a_j$ and $a_i \leqslant s_j \leqslant a_j \leqslant s_i$. In all six cases, the E/T penalties incurred by T_i and T_j do not increase. □

3.4. Parallel machines

In this subsection we will consider parallel machine problems. We start however, with some properties that characterize the single-machine solutions computed by Algorithm GTW. Denote by $\mathrm{GTW}(a_1,\dots,a_N)$ the solution (schedule) returned by Algorithm GTW given the input $a_1 \leqslant \cdots \leqslant a_N$.

Property 5. Let $S = \mathrm{GTW}(a_1,\dots,a_N)$ and $\tilde{S} = \mathrm{GTW}(\tilde{a}_1,\dots,\tilde{a}_N)$. Denote by s_i, the starting time of T_i in S and by $\tilde{s}_i$ the starting time of $\tilde{T}_i$ in $\tilde{S}$. Assume that the E/T penalty functions satisfy $P = \tilde{P}$. If $a_i \leqslant \tilde{a}_i$ for $i = 1,\dots,N$, then $s_i \leqslant \tilde{s}_i$ for $i = 1,\dots,N$.

Proof. Assume $a_i = \tilde{a}_i$ for every $i \neq j$ and $a_j < \tilde{a}_j$. Then, by executing Algorithm GTW it is evident that $s_i \leqslant \tilde{s}_i$ for $i = 1,\dots,N$. Now, Property 5 is achieved by applying this claim repeatedly, each time considering a different j for which $a_j < \tilde{a}_j$. □

Property 6. Let $S = \mathrm{GTW}(a_1,\dots,a_N)$ and $\tilde{S} = \mathrm{GTW}(a_0, a_1,\dots,a_N)$, where $0 \leqslant a_0 \leqslant a_1$. Denote by s_i the starting time of T_i in S and by $\tilde{s}_i$ the starting time of T_i in $\tilde{S}$, then $s_i \leqslant \tilde{s}_i$ for $i = 1,\dots,N$.

Proof. Solution S is obtained from the set $\{T_1,\dots,T_N\}$ of tasks. Assume a task T_0 with $a_0 = 0$. Let $S^* = \mathrm{GTW}(0, a_1,\dots,a_N)$. Clearly, if $p \leqslant s_1$, $s_i = s_i^*$ for $i = 1,\dots,N$. On the other hand, if $p > s_1$, $s_i \leqslant s_i^*$ for $i = 1,\dots,N$. Meanwhile, apply Property 5 to $S^* = \mathrm{GTW}(0, a_1,\dots,a_N)$ and $\tilde{S} = \mathrm{GTW}(a_0, a_1,\dots,a_N)$ where $0 \leqslant a_0$, which results in $s_i^* \leqslant \tilde{s}_i$ for $i = 0,\dots,N$. Combining the two results we get $s_i \leqslant \tilde{s}_i$ for $i = 1,\dots,N$. □

Property 7. Let $S = \mathrm{GTW}(a_1,\dots,a_{N-1})$, $\tilde{S} = \mathrm{GTW}(a_1,\dots,a_{N-1},a_N)$. Denote by s_i, the starting time of T_i in S and denote by $\tilde{s}_i$, the starting time of T_i in $\tilde{S}$. Then, $s_i \leqslant \tilde{s}_{i+1}$ for $i = 1,\dots,N-1$.

Proof. As with Property 6, compare solution S to solution $S^* = \mathrm{GTW}(0, a_1,\dots,a_{N-1})$ and compare solution S^* to solution $\tilde{S}$ and combine the two results to get $s_i \leqslant \tilde{s}_{i+1}$ for $i = 1,\dots,N-1$. □

Consider now the $P_2|d_i,p|\mathrm{CONVEX}$ problem, with machines denoted M_1 and M_2. We suggest the following *Zigzagging algorithm* with an $O(N \log N)$ time complexity.

Algorithm 8.

$P_2|d_i,p|\mathrm{CONVEX}$
input *A set* $\tilde{T} = \{T_1,\dots,T_N\}$ *of tasks.*
returns *A schedule of* $\tilde{T}$.
begin
1. *Order the tasks in a nondecreasing order of their target starting times, i.e.,* $a_1 \leqslant \cdots \leqslant a_N$.
2. $S_1 := \mathrm{GTW}(a_1, a_3,\dots,a_{2\lfloor (N-1)/2\rfloor} + 1)$.

3. $S_2 := \text{GTW}(a_2, a_4, \ldots, a_{2\lfloor N/2 \rfloor})$.
4. *Apply schedule S_i to machine M_i for $i := 1, 2$.*
return *The schedule of S_i on M_i.* [The solution.]
end $P_2|d_i, p|$CONVEX

Lemma 9. *The Zigzagging solution satisfies $s_i \leqslant s_{i+1}$ for $1 \leqslant i < N - 1$.*

Proof. Consider first an odd i, then s_i is obtained from $\text{GTW}(a_1, a_3, \ldots, a_i, \ldots)$ whereas s_{i+1} is obtained from $\text{GTW}(a_2, a_4, \ldots, a_{i+1}, \ldots)$. It follows from Property 5 that $s_i \leqslant s_{i+1}$ and in general, $s_j \leqslant s_{j+1}$ for any odd j, where $1 \leqslant j < N - 1$.

Consider now an even i, then s_i is obtained from $S = \text{GTW}(a_2, a_4, \ldots, a_i, \ldots)$ whereas s_{i+1} is obtained from $\tilde{S} = \text{GTW}(a_1, a_3, a_5, \ldots, a_{i+1}, \ldots)$. Let $S^* = \text{GTW}(a_3, a_5, \ldots, a_{i+1}, \ldots)$ and by Property 5, $s_i \leqslant s^*_{i+1}$ (we add a virtual T^*_{N+1} to have N tasks in S^*). Apply Property 6 to $\tilde{S}$ and S^* and thus, $s^*_{i+1} \leqslant \tilde{s}_{i+1}$. Combining the two results we get $s_i \leqslant s^*_{i+1} \leqslant \tilde{s}_{i+1}$ and thus, $s_i \leqslant s_{i+1}$ for any even i, where $1 \leqslant i < N - 1$. □

Theorem 10. *The Zigzagging solution is optimal for the $P_2|d_i, p|$CONVEX problem.*

Proof. Let $\tilde{S}$ be the solution generated by the zigzagging algorithm and let S be an arbitrary solution. Because of the optimality of Algorithm GTW, we assume that it was applied to the sequence of S, separately to each machine. According to Theorem 4, the assignment in S will maintain the property such that, the tasks are sequenced with nondecreasing target starting times. Thus, the only possible difference between S and $\tilde{S}$ is a different assignment of tasks to machines. Assume that T_{i-1} and T_i are assigned to the same machine, say M_1, in S and are the first such pair of consecutive tasks (on either machine). Thus, the assignment of tasks $\{T_1, \ldots, T_{i-1}\}$ is equivalent in S and in $\tilde{S}$. Denote by T_k, the first task assigned to M_2 in S, following T_{i-2}. Clearly, $k > i$ and thus $a_k \geqslant a_i$. Denote by s_i, the starting time of T_i in S.

1. Assume $s_k < s_i$. Interchanging T_k with T_i in S is feasible, and by the same observations given in Theorem 4, such an interchange will result in a schedule with less or equal cost. If following this interchange $S = \tilde{S}$, we stop the procedure, having a zigzagging sequence with no greater cost, which concludes our proof. Otherwise, the value of i is greater.
2. Assume $s_i \leqslant s_k$ and $s_{i-2} \leqslant s_{i-1}$. Clearly, $p \leqslant s_i - s_{i-1}$ and thus $p \leqslant s_k - s_{i-1}$ and $p \leqslant s_i - s_{i-2}$. We now interchange tasks $\{T_1, T_3, \ldots, T_{i-1}\}$ and tasks $\{T_2, T_4, \ldots, T_{i-2}\}$ maintaining the original starting times of each set. The interchange is feasible, the cost of the schedule remains the same and either $S = \tilde{S}$ or, the value of i is greater.
3. Assume $s_i \leqslant s_k$ and $s_{i-1} < s_{i-2}$. We interchange T_{i-2} and T_{i-1} maintaining the original starting times of the tasks. The interchange is feasible and results in a schedule with less or equal cost. We now observe s_{i-3} and s_{i-4}. If $s_{i-3} < s_{i-4}$, we create a similar interchange. Otherwise, we perform the feasible interchange of tasks $\{T_1, T_3, \ldots, T_{i-3}\}$ and tasks $\{T_2, T_4, \ldots, T_{i-4}\}$ maintaining the original starting times of each set. If the former instance occurs ($s_{i-3} < s_{i-4}$), we observe s_{i-5} and s_{i-6} and so on. Once completed, we maintain a schedule with less or equal cost and either $S = \tilde{S}$ or, the value of i is greater.

In all three instances, following the interchange, either $S = \tilde{S}$ or, the value of i is greater. We continue to the next two consecutive tasks until all tasks are assigned in a zigzagging sequence, similar to $\tilde{S}$. Since all interchanges do not increase the cost of S, the zigzagging sequence in $\tilde{S}$ is thus, an optimal assignment procedure. □

We now consider the $P_m|d_i, p|$CONVEX problem, with m machines denoted $M_1, \dots, M_m$, and suggest the following *Extended Zigzagging algorithm* with an $O(N \log N)$ time complexity.

Algorithm 11.

$P_m|d_i, p|$CONVEX
  input *A set* $\tilde{T} = \{T_1, \dots, T_N\}$ *of tasks.*
  returns *A schedule of* $\tilde{T}$.
  begin
  1. *Order the tasks in a nondecreasing order of their target starting times, i.e.,* $a_1 \leqslant \cdots \leqslant a_N$.
  for $i = 1, \dots, m$:
    1. $S_i :=$ GTW$(a_i, a_{m+i}, a_{2m+i}, \dots, a_{(\lfloor N/m \rfloor - 1)m+i})$.
    2. *Apply schedule* S_i *to machine* M_i.
    end for
  return *The schedule of* S_i *on* M_i.   [The solution.]
  end $P_m|d_i, p|$CONVEX

We may apply Lemma 9 and Theorem 10 on any two machines and thus, we conclude by the following lemma and theorem.

Lemma 12. *The Extended Zigzagging solution satisfies* $s_i \leqslant s_{i+1}$ *for* $1 \leqslant i < N - 1$.

Theorem 13. *The extended Zigzagging solution is optimal for the* $P_m|d_i, p|$CONVEX *problem.*

4. Non-execution penalty

In this section we remove the requirement that all tasks must be executed. Denote by γ_i the penalty incurred if T_i is not executed. For example, a modified TWET problem is to minimize $\sum_{i=1}^{N} [(1 - x_i)(\alpha_i e_i + \beta_i t_i) + x_i \gamma_i]$ where $x_i = 0$ if T_i is executed and $x_i = 1$, otherwise. We will consider different cost functions and present polynomial time algorithms.

4.1. $1|d_i, p, \gamma_i|$CONVEX

We present a pseudopolynomial time DP algorithm, which computes an optimal solution for the $1|d_i, p, \gamma_i|$CONVEX problem.

Lemma 14. *Consider the* $1|d_i, p, \gamma_i|\text{CONVEX}$ *problem, and denote by S the optimal solution. Also denote by* S_∞ *the optimal solution to the* $1|d_i, p, \gamma_i = \infty|\text{CONVEX}$ *problem (i.e., the problems are identical, except for their non-execution penalty weights). If* S_∞ *includes idle time between* T_k *and* T_{k+1}*, such an idle time also exists in S.*

Proof. According to Theorem 4, sequencing the tasks in a non-decreasing order of their target starting times, assuming all tasks are executed, is optimal. Without loss of generality, assume that $0 \leqslant a_1 \leqslant \cdots \leqslant a_N$, that there is an idle time in S_∞ between T_k and T_{k+1}, and that T_j is the only non-executed task in S. The case with more than a single non-executed task, is proved by applying the arguments inductively. Note that in S_∞ all tasks must be executed. Assume that $1 \leqslant j \leqslant k$. The proof for the case where $k+1 \leqslant j \leqslant N$, is analogous. Denote by $\{s_i\}$ and $\{s_{\infty_i}\}$, the starting times in S and in S_∞, respectively. Clearly, $s_i = s_{\infty_i}$ for $k+1 \leqslant i \leqslant N$ and specifically, $s_{k+1} = s_{\infty_{k+1}}$. Apply Property 5 to $\tilde{S} = \text{GTW}(0, a_1, \ldots, a_{j-1}, a_{j+1}, \ldots, a_k)$ and $\tilde{S}_\infty = \text{GTW}(a_1, \ldots, a_{j-1}, a_j, a_{j+1}, \ldots, a_k)$, which results in $\tilde{s}_i \leqslant \tilde{s}_{\infty_i}$, where $\{\tilde{s}_i\}$ and $\{\tilde{s}_{\infty_i}\}$ are the starting times in $\tilde{S}$ and in $\tilde{S}_\infty$, respectively. Specifically, $\tilde{s}_k \leqslant \tilde{s}_{\infty_k}$. Apply Property 6 to S and $\tilde{S}$, which results in $s_i \leqslant \tilde{s}_i$ for $i = 1, \ldots, j-1, j+1, \ldots, k$ and specifically, $s_k \leqslant \tilde{s}_k$. Combining the results, we achieve $s_k \leqslant \tilde{s}_{\infty_k} = s_{\infty_k}$. Thus, the idle time between T_k and T_{k+1} in S, is not smaller than the corresponding idle time in S_∞. □

Lemma 14 does not hold if tasks may have distinct processing times. Observe the following example. There are 3 tasks, T_1, T_2 and T_3. Target starting times are $a_1 = a_2 = 2$ and $a_3 = 4$. The E/T penalty weights are $\alpha_1 = \alpha_2 = \alpha_3 = \beta_1 = \beta_3 = 1$ and $\beta_2 = 2$. The processing times of the tasks are $p_1 = 2$ and $p_2 = p_3 = 1$. The optimal solution without allowing non-execution is $S_1 = (T_1, T_2, T_3)$, with $\text{cost}(S_1) = 2$ and starting times are $(0, 2, 4)$ respectively, with an idle time in the time interval [3,4]. Assume that non-execution is allowed and that $\gamma_2 = \varepsilon < 2$ and γ_1 and γ_3 are very large. Thus, the optimal solution is executing $S_2 = (T_1, T_3)$, with $cost(S_2) = \varepsilon < 2$ and starting times are $(2, 4)$ respectively. T_2 is not executed and S_2 does not have an idle time.

Corollary 15. *Consider the* $1|d_i, p, \gamma_i|\text{CONVEX}$ *problem and assume* $a_1 \leqslant \cdots \leqslant a_N$*. Assume that* S_∞ *includes idle time. Then, under any given set of values of* γ_i*, the problem can be divided into sub-problems, each sub-problem considers a different block of tasks from* S_∞*.*

Therefore, without loss of generality, we assume that S_∞ does not include an idle time. Thus, $s_{\infty_1} \leqslant a_1$, $a_N \leqslant s_{\infty_N}$, $s_{\infty_N} = s_{\infty_1} + (N-1)p$, and the time domain of the DP algorithm is the time interval $[s_{\infty_1}, s_{\infty_N}]$. Applying a common shift to the target starting times (and as a result, the starting times are shifted), we may assume without loss of generality that $s_{\infty_1} = 0$ and that the time domain of the starting times is $[0, (N-1)p]$.

Define $f_i(t)$ as the cost of an optimal solution for the sub-problem including tasks $\{T_i, T_{i+1}, \ldots, T_N\}$, where the machine is free for processing only from time t, in which $t \in [0, 1, \ldots, (N-1)p]$. The algorithm calculates the value of $f_i(t)$ for different values of t, where $t = (N-1)p, (N-1)p - 1, \ldots, 0$ and then proceeds to calculate the value of $f_{i-1}(t)$ for $t = (N-1)p, (N-1)p - 1, \ldots, 0$, etc. The result is the optimal solution $f_1(0)$. At each decision point, we may execute T_i and pay a potential E/T penalty or, we may prefer to not-execute T_i and pay a non-execution penalty γ_i. Denote by $P(t, a_i)$ the cost of scheduling T_i to start at t given that a_i is its target starting time.

Algorithm 16.

$1|d_i, p, \gamma_i|\text{CONVEX}$
input *A set* $\tilde{T} = \{T_1, \ldots, T_N\}$ *of tasks.*
returns *A schedule of the executed tasks.*
begin
1. *Order the tasks in a nondecreasing order of their target starting times, i.e.,* $a_1 \leqslant \cdots \leqslant a_N$.
2. $f_{N+1}(t) := 0$ for $t = 0, 1, 2, \ldots, (N-1)p$.
3. $f_i(t) := \infty$ for $i = 1, \ldots, N$ and $(i-1)p < t \leqslant (N-1)p$.
4. *Calculate* $f_i(t)$ for $i = N, \ldots, 1$ and $t = (N-1)p, \ldots, 0$, where

$$f_i(t) := \min \begin{cases} f_{i+1}(t) + \gamma_i & \text{if } T_i \text{ is not executed} \\ f_{i+1}(t+p) + P(t, a_i) & \text{if } T_i \text{ starts execution at } t \\ f_i(t+1) & \text{if } T_i \text{ starts execution after } t. \end{cases}$$

return $f_1(0)$ [The solution.]
end $1|d_i, p, \gamma_i|\text{CONVEX}$

The time complexity of Algorithm 16 is $O(N^2 p)$, due to $O(N)$ iterations and an $O(Np)$ time complexity of each iteration.

4.2. $1|d_i, p, \gamma_i|\text{TWET}$

We now show, that under the dominance condition of the E/T penalty weights described in Section 2.2, the $1|d_i, p, \gamma_i|\text{TWET}$ problem is polynomially solvable. We use the result of Verma and Dessouky [22] for the $1|d_i, p|\text{TWET}$ problem and perform necessary modifications to their algorithm. To enable easy reference, we use their terminology (see Problem 2.1 in [22]), assuming $p = 1$ and $\{d_i\}$ are not necessarily integers. We use the following notations:

- $x_{i,j} \in \{0, 1\}$ denotes whether T_i is executed at time j, where $x_{i,j} = 1$ if T_i is scheduled to start at j, and $x_{i,j} = 0$, otherwise.
- $c_{i,j}$ denotes the E/T penalty incurred as T_i is executed at time j, where $c_{i,j} = \alpha_i e_i + \beta_i t_i$ with $e_i = (a_i - j)^+$ and $t_i = (j - a_i)^+$.
- Q_i is the set of feasible starting times of T_i (i.e., if T_i is executed, its starting time must be a member of Q_i). Note that the set Q_i is calculated as a part of the solution, but its size $|Q_i|$, is polynomially bounded and thus the complexity of the algorithm remains polynomial.
- H_j is the set of feasible starting times of *all* tasks, which are not later than time j and not earlier than one time unit from j, i.e., $H_j \equiv \{j' \in \{Q_i\}_{i=1}^N | 0 \leqslant j - j' < 1\}$. Note that $|H_j|$ is polynomially bounded, since all $|Q_i|$ are polynomially bounded. Thus, the complexity of the algorithm remains polynomial.

The problem can be formulated as follows:

$$\begin{aligned} \min \quad & z = \sum_{i=1}^{N} \sum_{j \in Q_i} c_{i,j} x_{i,j} + \sum_{i=1}^{N} \gamma_i \left(1 - \sum_{j \in Q_i} x_{i,j}\right) \\ \text{s.t.} \quad & \sum_{j \in Q_i} x_{i,j} \leqslant 1 \qquad \text{for } 1 \leqslant i \leqslant N, \end{aligned} \tag{1}$$

$$\sum_{i=1}^{N} \sum_{j' \in H_j \cap Q_i} x_{i,j'} \leqslant 1 \quad \forall j, \tag{2}$$

$$x_{i,j} \in \{0,1\} \qquad \text{for } 1 \leqslant i \leqslant N \quad \text{and} \quad \forall j \in Q_i. \tag{3}$$

The only differences between the formulation of Verma and Dessouky [22] and ours, are the right term of the objective function (the penalty for non-execution) and that constraint (1) is an inequality instead of an equation (since we allow non-execution). All proofs given in Verma and Dessouky [22] except for their Theorem 3.1, do not consider the objective function and thus, hold also in our case. The proof of Theorem 3.1 is based upon a mutual decrease of strictly positive variables and an equal increase of other variables, which represent the same task. Thus, such a change in variable values, does not change the total sum of variables of each task and therefore, the penalty accrued by the non-execution penalty remains fixed. Thus, Theorem 3.1 also holds in this problem and it is polynomially solvable.

Note that the $Q_m|d_i, p = 1|$GENERAL problem is solved as an assignment problem. We now present special cases, where more efficient algorithms apply. In the algorithms, we use the concept of positional weights, which was introduced in Section 2.1. We will specifically describe the structure of the positional weights, in each problem. In all four cases, we consider a common due date.

4.3. $1|d, p, \gamma|$STWET

Assume that the tasks are ordered in a non-increasing order of their E/T penalty weights $w_1 \geqslant \cdots \geqslant w_N$. We will show that the set of executed tasks consists of those with the smallest E/T penalty weights. We define a strictly early set B and a tardy set A, as in Section 2.1 and assign half of the tasks to B and the other half to A in a "V-shape" sequence (see Section 2.1). Since the first task in A does not incur an E/T penalty, we assign T_1 to start at a. Thus, all odd indexed tasks are assigned to A and all even indexed tasks are assigned to B. The positional weight of $T_j \in B$ is $\sum_{i=j,j+2,\ldots} pw_i$ and the positional weight of $T_j \in A$ is $\sum_{i=j+2,j+4,\ldots} pw_i$. Thus, T_2 has the largest positional weight. If $\gamma < \sum_{i=2,4,\ldots} pw_i$, T_1 is not executed and the process is repeated with the set $\{T_2,\ldots,T_N\}$ of $N-1$ tasks. Now, T_2 is the task with the largest E/T penalty weight, and thus is assigned to start at a. Therefore, all even indexed tasks are now assigned to A and all odd indexed tasks (except for T_1, which is not executed) are now assigned to B. Now, T_3 has the largest positional weight $\sum_{i=3,5,\ldots} pw_i$ and we verify whether $\gamma < \sum_{i=3,5,\ldots} pw_i$. Generally, assume that we do not execute the task with the current smallest index, say T_j. Thus, the positional weight of T_{j+1} (becomes the first processed task in A), decreases by pw_{j+1}.

Property 17. There exists an optimal solution, which executes $\{T_{i^*},\ldots,T_N\}$ and does not execute $\{T_1,\ldots,T_{i^*-1}\}$, where i^* is the minimal index such that $\gamma \geqslant \sum_{i=i^*+1,i^*+3,\ldots} pw_i$.

Proof. First, we show that the set of executed tasks, denoted S, consists of those with the smallest E/T penalty weights, i.e., $S = \{T_{i^*},\ldots,T_N\}$. To the contrary, assume that $\exists T_k \notin S$ and $\exists T_j \in S$ such that $w_k \leqslant w_j$. If we execute T_k instead of T_j, the E/T penalties incurred by all other tasks in S do not change, and the saving Δ in the total cost due to the change is: $\Delta = pw_j - pw_k \geqslant 0$. Therefore, performing the change does not increase the total cost of the solution. The property then follows by induction on the number of interchanges needed, so that the set of tasks $\{T_{i^*},\ldots,T_N\}$ is executed.

We now show, that i^* is the minimal index such that, $\gamma \geqslant \sum_{i=i^*+1,i^*+3,\ldots} pw_i$. To the contrary, assume that i^* is smaller than the minimal possible such index. Thus, $\gamma < \sum_{i=i^*+1,i^*+3,\ldots} pw_i$ and not executing T_{i^*} as directed by the algorithm, results in a positive reduction in the total cost. Similarly, if i^* is larger than the minimal possible such index, $\gamma \geqslant \sum_{i=i^*,i^*+2,\ldots} pw_i$ and executing T_{i^*-1} is desirable, since it results in a non-negative cost reduction. □

Algorithm 18.

$1|d, p, \gamma|$STWET
input *A set* $\tilde{T} = \{T_1, \ldots, T_N\}$ *of tasks.*
returns *A schedule of the executed tasks.*
begin
1. *Order the tasks in a non-increasing order of their E/T penalty weights, i.e.,* $w_1 \geqslant \cdots \geqslant w_N$.
2. $b := p(w_2 + w_4 + w_6 + \cdots + w_{2\lfloor \frac{N}{2} \rfloor})$.
3. $a := p(w_3 + w_5 + w_7 + \cdots + w_{2\lfloor (N-1)/2 \rfloor + 1})$.
4. $i^* := 1$.
while $\gamma < \max\{a, b\}$ **do**
  $i^* := i^* + 1$.
  if i^* *is even*
    then
      $b := b - pw_{i^*}$.
    else
      $a := a - pw_{i^*}$.
    end if
end while
5. *Assign* $\{T_{i^*}, \ldots, T_N\}$ *to B and A such that,* $B := (\ldots, T_{i^*+5}, T_{i^*+3}, T_{i^*+1})$ *and* $A := (T_{i^*}, T_{i^*+2}, T_{i^*+4}, \ldots)$.
6. *Schedule set B to complete at a and schedule set A to start at a.*
return *The schedule of sets B and A.* [The solution.]
end $1|d, p, \gamma|$STWET

$O(N \log N)$ time is needed to order the tasks and to calculate the initial values of b and a. $O(N)$ time is needed to determine the value of i^* (the **while** loop) and for sequencing. Thus, the overall complexity of the algorithm is $O(N \log N)$.

Note that the solution is not unique, as the first scheduled task in A could be replaced by any non-executed task and the cost remains the same.

4.4. $1|d, p_i, \gamma_i|$WET

For arbitrary and distinct p_i and γ_i, the complexity of the problem is unknown. However, if the processing times and the non-execution penalty weights maintain the following condition, the problem is polynomially solvable.

Definition 19. Assume that tasks $T_1,\dots,T_N$ are ordered in a non-decreasing order of their processing times, i.e., $p_1 \leqslant \cdots \leqslant p_N$. We say that the tasks are *agreeably reversed*, if the non-execution penalty weights γ_i maintain $\gamma_1 \geqslant \cdots \geqslant \gamma_N$.

The case with agreeably reversed tasks, is solved in an $O(N \log N)$ time complexity. There are two special cases in which the tasks are agreeably reversed.

- **Common non-execution penalty weight.** In this case, we assume that $\gamma_i = \gamma$ for $i = 1,\dots,N$.
- **Common processing time.** In this case, we assume $p_i = p$ for $i = 1,\dots,N$ and a more efficient $O(N)$ time algorithm is presented. The problem is considered in Section 4.5.

Assume that the tasks are ordered in a non-decreasing order of their processing times $p_1 \leqslant \cdots \leqslant p_N$ and that the tasks are agreeably reversed. We will show, that the set of executed tasks consists of those with the smallest processing times (and the largest non-execution penalty weights). We define the sets B and A, as before. Assume that k tasks are executed and thus, we assign $\lfloor k\beta/(\alpha+\beta)\rfloor$ tasks to B and $\lceil k\alpha/(\alpha+\beta)\rceil$ tasks to A, in a "V-shape" sequence (see Section 2.1), where T_k is scheduled last. The E/T penalty of the first processed task is $\sum_{i\in B} \alpha p_i$ and the E/T penalty of T_k, is $\sum_{i\in A\setminus\{k\}} \beta p_i$. Therefore, if $\gamma_k < \max\{\sum_{i\in B} \alpha p_i, \sum_{i\in A\setminus\{k\}} \beta p_i\}$, T_k is not executed and the process is repeated with the set $\{T_1,\dots,T_{k-1}\}$ of $k-1$ remaining tasks. Otherwise, all k tasks are executed, and $\forall T_j \in \{T_1,\dots,T_k\}$, $\gamma_j \geqslant \max\{\sum_{i\in B} \alpha p_i, \sum_{i\in A\setminus\{k\}} \beta p_i\}$. We refer to this condition as the *execution property*. If $T_i \in B$, is the lth processed task, its positional weight is $l\alpha p_i$ and if $T_i \in A$, is the lth processed task in A, its positional weight is $(\lceil k\alpha/(\alpha+\beta)\rceil - l)\beta p_i$.

Property 20. Assume that the tasks are ordered in a non-decreasing order of their processing times $p_1 \leqslant \cdots \leqslant p_N$, and that the tasks are agreeably reversed. There exists an optimal solution, which executes the set of tasks $\{T_1,\dots,T_{i^*}\}$ and does not execute $\{T_{i^*+1},\dots,T_N\}$, where i^* is the maximal index such that the set $\{T_1,\dots,T_{i^*}\}$ maintains the execution property.

Proof. Assume to the contrary, that $\exists T_k \notin S$ and $\exists T_j \in S$ such that $p_k \leqslant p_j$ (and $\gamma_k \geqslant \gamma_j$), where S denotes the set of executed tasks. Assume that we execute T_k instead of T_j and that T_j (T_k after the change) is processed as the lth task in B. Thus, the saving Δ is: $\Delta = l\alpha p_j + \gamma_k - l\alpha p_k - \gamma_j = l\alpha(p_j - p_k) + (\gamma_k - \gamma_j) \geqslant 0$. The proof for the case where T_j (T_k after the change) is processed as the lth task in A, is analogous. The property then follows by induction on the number of interchanges needed, so that the set of tasks $\{T_1,\dots,T_{i^*}\}$ is executed.

If i^* is not the maximal index such that the set $\{T_1,\dots,T_{i^*}\}$ maintains the execution property, then either not executing T_{i^*} or executing T_{i^*+1} is desirable, which contradicts the assumption that i^* is the optimal index. □

Algorithm 21.

$1|d, p_i, \gamma_i|\mathrm{WET}$

input *A set* $\tilde{T} = \{T_1,\dots,T_N\}$ *of tasks.*
returns *A schedule of the executed tasks.*
begin
1. *Order the tasks in a non-decreasing order of their processing times, i.e.,* $p_1 \leqslant \cdots \leqslant p_N$.

```
2. N_L := 1                       [Lower border of index interval.]
3. N_H := N                       [Upper border of index interval.]
4. i* := ⌈(N_L + N_H)/2⌉          [The index of i*.]
while N_H > N_L do
    1. Call Procedure Assignment with i*.
    if γ_{i*} ⩾ max{∑_{i∈B} αp_i, ∑_{i∈A\{i*}} βp_i}
        then
            N_L := i*.
        else
            N_H := i* − 1.
        end if
    2. i* := ⌈(N_L + N_H)/2⌉.
    end while
5. Call Procedure Assignment with i*.
6. Schedule set B to complete at a and schedule set A to start at a.
return The schedule of {T_1, ..., T_{i*}}.     [The solution.]
end 1|d, p_i, γ_i|WET
```

```
Procedure Assignment
    input i*.      [The ordered set of tasks {T_1, ..., T_{i*}}.]
    returns Assignment of tasks {T_1, ..., T_{i*}} to sets B and A.
    begin
    A := ∅; B := ∅.
    for i = i*, ..., 1:
        if α(|B| + 1) > β|A|
            then
                A := (T_i, A)      [T_i is the current first processed task in A.]
            else
                B := (B, T_i)      [T_i is the current last processed task in B.]
            end if
        end for
    return Sets B and A.
    end Procedure Assignment
```

Ordering the tasks requires $O(N \log N)$ time. The **while** loop requires $O(N \log N)$ time, ($O(\log N)$ iterations are needed for the binary search of i^* and in each iteration, Procedure *Assignment* requires $O(N)$ time). Thus, the overall time complexity of the algorithm is $O(N \log N)$.

4.5. $1|d, p, \gamma_i|$WET

The problem is a special case of the $1|d, p_i, \gamma_i|$WET problem with agreeably reversed tasks, which was discussed in the previous section. The purpose of this subsection is to solve the problem more efficiently. Assume that $\gamma_1 \geqslant \cdots \geqslant \gamma_N$. We will show that the set of executed tasks consists of those with the largest non-execution penalty weights. We define the sets B and A, as before. Assume that

k tasks are executed and thus, we assign $\lfloor k\beta/(\alpha+\beta)\rfloor$ tasks to B and $\lceil k\alpha/(\alpha+\beta)\rceil$ tasks to A. The earliness penalty of the first processed task is $p\alpha\lfloor k\beta/(\alpha+\beta)\rfloor$ and the tardiness penalty of the last processed task is $p\beta(\lceil k\alpha/(\alpha+\beta)\rceil-1)$. Denote $\gamma_{\min}\equiv\min_{i=1,\dots,k}\{\gamma_i\}$. If $\gamma_{\min}<\max\{p\alpha\lfloor k\beta/(\alpha+\beta)\rfloor, p\beta(\lceil k\alpha/(\alpha+\beta)\rceil-1)\}$, $T_{\gamma_{\min}}$ is not executed and the process is repeated with the set of $k-1$ remaining tasks. Otherwise, all k tasks are executed. Each executed T_i maintains the *execution property*, i.e., $\gamma_i\geqslant\max\{p\alpha\lfloor k\beta/(\alpha+\beta)\rfloor, p\beta(\lceil k\alpha/(\alpha+\beta)\rceil-1)\}$.

Property 22. There exists an optimal solution, which executes the set of tasks $\{T_1,\dots,T_{i^*}\}$ and does not execute $\{T_{i^*+1},\dots,T_N\}$, where i^* is the maximal index such that the set $\{T_1,\dots,T_{i^*}\}$ maintains the execution property.

Proof. Assume to the contrary, that $\exists T_k\notin S$ and $\exists T_j\in S$ such that $\gamma_k\geqslant\gamma_j$, where S denotes the set of executed tasks. Assume that we execute T_k instead of T_j. Thus, the saving Δ is: $\Delta=\gamma_k-\gamma_j\geqslant 0$. The property then follows by induction on the number of interchanges needed, so that the set of tasks $\{T_1,\dots,T_{i^*}\}$ is executed.

If i^* is not the maximal index such that the set $\{T_1,\dots,T_{i^*}\}$ maintains the execution property, then either not executing T_{i^*} or executing T_{i^*+1} is desirable, which contradicts the assumption that i^* is the optimal index. □

Algorithm 23.

$1|d,p,\gamma_i|$WET
  input *A set* $\tilde{T}=\{T_1,\dots,T_N\}$ *of tasks.*
  returns *A schedule of the executed tasks.*
  begin
  1. $N_L:=1$ [Lower border of index interval.]
  2. $N_H:=N$ [Upper border of index interval.]
  3. $i^*:=\lceil(N_L+N_H)/2\rceil$ [The index of i^*.]
  4. $T:=\tilde{T}$ [The set of tasks considered.]
  5. $M:=i^*-N_L+1$ [The median index of the set of tasks considered.]
  while $N_H>N_L$ **do**
    1. *Find* $\gamma_{\{M\}}$ *from the tasks in* T.
    if $\gamma_{\{M\}}\geqslant\max\{p\alpha\lfloor i^*\beta/(\alpha+\beta)\rfloor, p\beta(\lceil i^*\alpha/(\alpha+\beta)\rceil-1)\}$
      then
        $T:=\{T_j\in T|\gamma_j\leqslant\gamma_{\{M\}}\}$.
        $N_L:=i^*$.
      else
        $T:=\{T_j\in T\setminus T_{\{M\}}|\gamma_j\geqslant\gamma_{\{M\}}\}$.
        $N_H:=i^*-1$.
      end if
    2. $i^*:=\lceil(N_L+N_H)/2\rceil$.
    3. $M:=i^*-N_L+1$.
  end while

6. $T := \{T_j \in \tilde{T} | \gamma_j \geqslant \gamma_{\{i^*\}}\}$.
7. *Assign* $\lfloor i^*\beta/(\alpha+\beta)\rfloor$ *tasks from* T *as set* B *and assign the remaining* $\lceil i^*\alpha/(\alpha+\beta)\rceil$ *tasks, as set* A.
8. *Schedule set* B *to complete at* a *and schedule set* A *to start at* a.
return *The schedule of* T. [The solution.]
end $1|d,p,\gamma_i|\mathrm{WET}$

In Algorithm 23, we denote by $\gamma_{\{i\}}$ the ith largest non-execution penalty in a given set of tasks. Without loss of generality, assume that the non-execution penalty weights are distinct to all tasks. The assumption is necessary, as the median of the non-execution penalty weights in the binary search, has to be distinct.

$O(N)$ time is needed to find the median value of N values. Each iteration of the **while** loop, decreases the size of T by half and thus it takes $O(N+\frac{N}{2}+\frac{N}{4}+\frac{N}{8}+\cdots)=O(N)$ time to find i^*. Then, another $O(N)$ time is needed to determine the set of tasks to be executed and to schedule them. Thus, the overall complexity of the algorithm is $O(N)$.

4.6. $1|d,\frac{p_i}{\gamma_i}=r|\mathrm{WET}$

Assume that the tasks are ordered in a non-decreasing order of their processing times $p_1 \leqslant \cdots \leqslant p_N$. We will show that the set of executed tasks consists of those with the largest processing times. We define the sets B and A, as before. We assign $\lfloor N\beta/(\alpha+\beta)\rfloor$ tasks to B and $\lceil N\alpha/(\alpha+\beta)\rceil$ tasks to A, in a "V-shape" sequence. If $T_1 \in B$, its positional weight is $\lfloor N\beta/(\alpha+\beta)\rfloor \alpha p_1$ and if $T_1 \in A$, its positional weight is $(\lceil N\alpha/(\alpha+\beta)\rceil - 1)\beta p_1$. Thus, it is desirable to not execute T_1, if $\gamma_1 < \max\{\lfloor N\beta/(\alpha+\beta)\rfloor \alpha p_1, (\lceil N\alpha/(\alpha+\beta)\rceil - 1)\beta p_1\}$ and then, the process is repeated with the set $\{T_2,\ldots,T_N\}$ of $N-1$ remaining tasks. Otherwise, all N tasks are executed.

Property 24. Assume that the tasks are ordered in a non-decreasing order of their processing times, i.e., $p_1 \leqslant \cdots \leqslant p_N$. There exists an optimal solution, which executes the set of tasks $\{T_{i^*+1},\ldots,T_N\}$ and does not execute $\{T_1,\ldots,T_{i^*}\}$, where i^* is the maximal index such that, $\gamma_{i^*} < \max\left\{\left\lfloor \frac{(N-i^*+1)\beta}{\alpha+\beta}\right\rfloor \alpha p_{i^*}, \left(\left\lceil \frac{(N-i^*+1)\alpha}{\alpha+\beta}\right\rceil - 1\right)\beta p_{i^*}\right\}$.

Proof. Assume to the contrary, that $\exists T_k \notin S$ and $\exists T_j \in S$ such that $p_k \geqslant p_j$, where S denotes the set of executed tasks. Assume that we execute T_k instead of T_j and that T_j (T_k after the change) is processed as the lth task in B. Thus, its positional weight is $l\alpha p_j$ ($l\alpha p_k$). For the executed T_j (T_k after the change), the non-execution penalty is not smaller than its positional weight, i.e., $l\alpha p_j \leqslant \gamma_j$ or $l\alpha \leqslant \frac{1}{r}$. Thus, the saving Δ due to the change is: $\Delta = l\alpha p_j + \gamma_k - l\alpha p_k - \gamma_j = \left(l\alpha - \frac{1}{r}\right)(p_j - p_k) \geqslant 0$. The proof for the case where T_j (T_k after the change) is processed as the lth task in A, is analogous. The property then follows by induction on the number of interchanges needed, so that the set of tasks $\{T_{i^*+1},\ldots,T_N\}$ is executed.

If i^* is not the maximal index such that γ_{i^*} is strictly smaller than the maximal positional weights, i.e., $\gamma_{i^*} < \max\{\lfloor (N-i^*+1)\beta/(\alpha+\beta)\rfloor \alpha p_{i^*}, (\lceil (N-i^*+1)\alpha/(\alpha+\beta)\rceil - 1)\beta p_{i^*}\}$, then either not executing T_{i^*+1} or executing T_{i^*} is desirable, which contradicts the assumption that i^* is the optimal index. □

Algorithm 25.

$1|d, \frac{p_i}{\gamma_i} = r|\text{WET}$
  input *A set* $\tilde{T} = \{T_1, \ldots, T_N\}$ *of tasks.*
  returns *A schedule of the executed tasks.*
  begin
  1. *Order the tasks in a non-decreasing order of their processing times, i.e.,* $p_1 \leqslant \cdots \leqslant p_N$.
  2. $A := \emptyset$; $B := \emptyset$.
  for $i = N, \ldots, 1$: [Assigning the ordered tasks to sets B and A.]
    if $\alpha(|B| + 1) > \beta|A|$
      then
        $A := (T_i, A)$ [T_i is the current first processed task in A.]
      else
        $B := (B, T_i)$ [T_i is the current last processed task in B.]
      end if
    end for
  3. *Schedule set* B *to complete at* a *and schedule set* A *to start at* a.
  4. $i^* := 1$.
  5. $T := \tilde{T}$.
  while $\gamma_{i^*} < \max\{\lfloor (N - i^* + 1)\beta/(\alpha + \beta)\rfloor \alpha p_{i^*}, (\lceil (N - i^* + 1)\alpha/(\alpha + \beta)\rceil - 1)\beta p_{i^*}\}$ **do**
    1. $T := T \setminus T_{i^*}$. [Do not execute T_{i^*}.]
    if $T_{i^*} \in B$
      then
        $s_j := s_j + p_{i^*} \quad \forall T_j \in B$ [Shift set B, p_{i^*} time units later.]
      else
        $s_j := s_j - p_{i^*} \quad \forall T_j \in A$ [Shift set A, p_{i^*} time units earlier.]
      end if
    2. $i^* := i^* + 1$.
    end while
  return *The schedule of* T. [The solution.]
  end $1|d, p_i/\gamma_i = r|\text{WET}$

Ordering the tasks requires $O(N \log N)$ time. The **for** loop requires $O(N)$ time, and $O(N)$ iterations are needed to determine the value of i^*. Thus, the overall complexity of Algorithm 25 is $O(N \log N)$.

References

[1] Garey MR, Tarjan RE, Wilfong GT. One-processor scheduling with symmetric earliness and tardiness penalties. Mathematics of Operations Research 1988;13(2):330–48.

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

[3] Baker KR, Scudder GD. Sequencing with earliness and tardiness penalties: a review. Operations Research 1990;38(1):22–36.

[4] Kanet JJ. Minimizing the average deviation of job completion times about a common due date. Naval Research Logistics Quarterly 1981;28(4):643–51.
[5] Hall NG. Single- and multiple-processor models for minimizing completion time variance. Naval Research Logistics Quarterly 1986;33(1):49–54.
[6] Bagchi U, Sullivan RS, Chang YL. Minimizing mean absolute deviation of completion times about a common due date. Naval Research Logistics Quarterly 1986;33(2):227–40.
[7] Bagchi U, Chang Y-L, Sullivan RS. Minimizing absolute and squared deviations of completion times with different earliness and tardiness penalties and a common due date. Naval Research Logistics 1987;34(5):739–51.
[8] Hall NG, Posner ME. Earliness-tardiness scheduling problems, I: weighted deviation of completion times about a common due date. Operations Research 1991;39(5):836–46.
[9] De P, Ghosh JB, Wells CE. On the minimization of completion time variance with a bi-criteria extension. Operations Research 1992;40:1148–55.
[10] Jurisch B, Kubiak W, Józefowska J. Algorithms for minclique scheduling problems. Discrete Applied Mathematics 1997;72:115–39.
[11] Kovalyov MY, Kubiak W. A fully polynomial approximation scheme for the weighted earliness-tardiness problem. Operations Research 1999;47(5):757–61.
[12] Szwarc W. The weighted common due date single machine scheduling problem revisited. Computers & Operations Research 1996;23(3):255–62.
[13] Hoogeveen JA, van de Velde SL. Earliness-tardiness scheduling around almost equal due dates. INFORMS Journal on Computing 1997;9(1):92–9.
[14] James RJW, Buchanan JT. A neighborhood scheme with a compressed solution space for the early/tardy scheduling problem. European Journal of Operational Research 1997;102(3):513–27.
[15] Davis JS, Kanet JJ. Single-machine scheduling with early and tardy completion costs. Naval Research Logistics 1993;40:85–101.
[16] Szwarc W, Mukhopadhyay SK. Optimal timing schedules in earliness-tardiness single machine sequencing. Naval Research Logistics 1995;42:1109–14.
[17] Fry TD, Armstrong RD, Blackstone JH. Minimizing weighted absolute deviation in single machine scheduling. IIE Transactions 1987;19(4):445–50.
[18] Fry TD, Armstrong RD, Rosen LD. Single machine scheduling to minimize mean absolute lateness: a heuristic solution. Computers & Operations Research 1990;17(1):105–12.
[19] Kim Y-D, Yano CA. Minimizing mean tardiness and earliness in single-machine scheduling problems with unequal due dates. Naval Research Logistics 1994;41:913–33.
[20] Ow PS, Morton TE. The single machine early/tardy problem. Management Science 1989;35(2):177–91.
[21] Lee CY, Choi JY. A genetic algorithm for job sequencing problems with distinct due dates and general early-tardy penalty weights. Computers & Operations Research 1995;22(8):857–69.
[22] Verma S, Dessouky M. Single-machine scheduling of unit-time jobs with earliness and tardiness penalties. Mathematics of Operations Research 1998;23(4):930–43.
[23] Kanet JJ, Sridharan V. Scheduling with inserted idle time: problem taxonomy and literature review. Operations Research 2000;48(1):99–110.
[24] Sundararaghavan PS, Ahmed MU. Minimizing the sum of absolute lateness in single-machine and multi-machine scheduling. Naval Research Logistics Quarterly 1984;31(2):325–33.
[25] Emmons H. Scheduling to a common due date on parallel uniform processors. Naval Research Logistics 1987;34(6):803–10.
[26] Kubiak W, Lou S, Sethi S. Equivalence of mean flow time problems and mean absolute deviation problems. Operations Research Letters 1990;9(6):371–4.
[27] Webster ST. The complexity of scheduling job families about a common due date. Operations Research Letters 1997;20(2):65–74.
[28] Chen Z-L, Powell WB. A column generation based decomposition algorithm for a parallel machine just-in-time scheduling problem. European Journal of Operational Research 1999;116(1):220–33.
[29] Lageweg BJ, Lenstra JK, Lawler EL, Rinnooy Kan AHG. Computer-aided complexity classification of combinatorial problems. Communications of the ACM 1982;25(11):817–22.

[30] Lawler EL, Lenstra JK, Rinnooy Kan AHG, Shmoys DB. Sequencing and scheduling: algorithms and complexity. In: Graves SC, Rinnooy Kan AHG, Zipkin PH editors. Logistics of production & inventory; handbooks in operations research and management science, vol. 4. Amsterdam: North-Holland; 1993. p. 445–522.
[31] Hall NG, Kubiak W, Sethi SP. Earliness-tardiness scheduling problems, II: deviation of completion times about a restrictive common due date. Operations Research 1991;39(5):847–56.
[32] Raghavachari M. A V-Shape property of optimal schedule of jobs about a common due date. European Journal of Operational Research 1986;23(3):401–2.
[33] Kubiak W. Completion time variance minimization on a single machine is difficult. Operations Research Letters 1993;14:49–59.
[34] Szwarc W. Single-machine scheduling to minimize absolute deviation of completion times from a common due date. Naval Research Logistics 1989;36(5):663–73.
[35] Hoogeveen JA, Oosterhout H, van de Velde SL. New lower and upper bounds for scheduling around a small common due date. Operations Research 1994;42(1):102–10.
[36] Federgruen A, Mosheiov G. Greedy heuristics for single-machine scheduling problems with general earliness and tardiness costs. Operations Research Letters 1994;16:199–208.
[37] Cai X, Lum VYS, Chan JMT. Scheduling about a common due date with job-dependent asymmetric earliness and tardiness penalties. European Journal of Operational Research 1997;98(1):154–68.