Migration from Sequence to Schedule in Total Earliness and Tardiness Scheduling Problem

Mohammad Namakshenas and Mohammad Mahdavi Mazdeh*

School of Industrial Engineering, Iran University of Science and Technology, Tehran, Iran
E-mail: m_namakshenas@ind.iust.ac.ir [M. Namakshenas]; mazdeh@iust.ac.ir [M.M. Mazdeh]

Abstract

Services must be delivered with high punctuality to be competitive. The classical scheduling theory offers to minimize the total earliness and tardiness of jobs to deliver punctual services. In this study, we developed a fully polynomial-time optimal algorithm to transform a given sequence, the permutation of jobs, into its corresponding minimum cost schedule, the timing of jobs. We provide the necessary and sufficient properties for the optimal scheduling of the sub-sequences of jobs, called clusters. The algorithm first decomposes a sequence into several clusters, and then it applies a recursive scheme to join the clusters and to generate their minimum cost schedules. The complexity status sheds light on our algorithm's competitiveness compared with other state-of-the-art algorithms in the literature.

Keywords: scheduling; single machine; total earliness and tardiness; exact algorithm

1. Introduction

The total tardiness and earliness scheduling problem has been receiving significant attention in the literature. One reason is that a large number of products and services, or jobs in general terms, should be produced or delivered at the right time. Therefore, it is undesirable that jobs finish too late as well as too early.

The total tardiness and earliness scheduling problem is essentially motivated by the Just-In-Time (JIT) production system in which all jobs ideally complete at their due dates. The formal description of the problem is as follows. Consider the permutaion set $J = \langle 1, \dots, n \rangle$ of independent jobs, $\forall j \in J$, with the processing times p_j and the due dates d_j that have to be processed on a single machine. All jobs are ready at time zero. The preemption is not allowed, i.e., a job cannot be interrupted once it has started. The objective is to decide the start times s_j that minimize the total earliness and tardiness criteria or equivalently the total absolute deviation of $s_j + p_j$ from d_j. According to the 3-field notation (Graham et al.,

*Author to whom all correspondence should be addressed (e-mail: mazdeh@iust.ac.ir).

1979), we identify the problem as $1 \mid\mid \sum_{j \in J} E_j + \sum_{j \in J} T_j$ in which $E_j = \max\{0, d_j - (s_j + p_j)\}$ and $T_j = \max\{0, (s_j + p_j) - d_j\}$. For the notational simplicity, we abbreviate the objective function $\sum_{j \in J} E_j + \sum_{j \in J} T_j$ as `ET`. The problem with equal processing times, $p_j = p,\ \forall j \in J$, is also characterized as $1 \mid p_j = p \mid$ `ET`.

The problem with distinct processing times $1 \mid\mid$ `ET` is hard to solve; Wan and Yuan (2013) proved that problem $1 \mid\mid$ `ET` is NP-hard in the strong sense by a unary pseudo-polynomial reduction from the `3-Partition` problem. In contrast to the `regular` objective functions, non-decreasing in s_j, the `non-regular` objective `ET` has an additional layer of complexity, i.e., finding the optimal position of idle times among jobs. Conway et al. (2003) showed that there exists an optimal schedule without having idle times among jobs for any scheduling problem in the form $1 \mid r_j = 0 \mid$ `regular`. In the theory of scheduling, the term *sequencing* is often used to define the *permutation* of jobs while the term *scheduling* refers to the *timing* which is usually described by the start times of jobs. Hence, an optimal schedule generates its optimal sequence, but not necessarily the other way around. In this study, we investigate the properties to polynomially transform an arbitrary sequence to its minimum cost schedule for $1 \mid\mid$ `ET`.

We refer to the work of Garey et al. (1988) as a serious attempt to develop a polynomial algorithm. The algorithm transforms a given sequence into a minimum cost schedule in $O(n \log(n))$ for $1 \mid\mid$ `ET`. Although the algorithm is robust and fast; however, its expressiveness based on the traditional tree-based heap data structure (Cormen et al., 2009) leads to a verbose algorithm. Davis and Kanet (1993) developed a time-tabling scheme and embedded it into an enumerative search algorithm to detect *semi-active* schedules and to prune the permutation space. In a semi-active schedule, no shift applied to any job can improve the objective value. They proved that any semi-active schedule dominates the set of all schedules for $1 \mid\mid$ `ET`, thereby produces a minimum cost schedule. This property was essentially helpful in reducing the permutation space as much as possible.

Szwarc and Mukhopadhyay (1995) developed an algorithm by dividing a given sequence into several sub-sequences, called *clusters*. Their algorithm's main feature is that the durations of idle times between the detected clusters remain unchanged. Nevertheless, the algorithm has a downside; all start times should be pushed forward in each iteration to produce a semi-active schedule. Our proposed algorithm only updates the necessary changes in the current cluster. The worst-case running time of Szwarc and Mukhopadhyay (1995) algorithm is the polynomial function of the number of clusters and jobs, which practically outperforms the algorithm of Garey et al. (1988) in most computational cases.

It is noteworthy to review a few other works which have developed lower bounds and greedy rules. For example, Kim and Yano (1994) proved several dominance properties for $1 \mid\mid$ `ET` in which two overlapping jobs finished at their distinct due dates. They developed a *conflict resolution* procedure for overlapping jobs to obtain tight lower bounds. The problem with equal processing times was investigated by Verma and Dessouky (1998); they formulated it as a *time-indexed* integer problem. They also tightened the model by propagating certain dominance properties to each job's start time; hence, they reduced the permutation space.

A body of researches has avoided considering idle times either by restricting problems to a non-idle schedule or a common due date for all jobs (see for example Gordon et al. (2002), Mason et al. (2005), Sourd (2005), Valente and Alves (2005), Tanaka et al. (2009), and Wodecki (2009)). Mason et al. (2005) used a pair-wise interchange procedure to improve their initial heuristic solutions, and Tanaka et al. (2009) developed a dynamic programming procedure to efficiently schedule a set of jobs in a reasonable amount of time. The complexity analyses of both works were not reported.

The following lines provide a summary of contributions in this study. We develop a new compact algorithm whose time complexity only depends on the polynomial function of the number of jobs and clusters to transform an input sequence into its minimum cost schedule for $1 \mid\mid \texttt{ET}$. Our proposed algorithm is compact since we manipulate all properties to a single well-structured algorithm. The algorithm only treats the schedule of one cluster regardless of the schedules of other clusters; hence, it does not push the entire schedule like in the algorithm of Szwarc and Mukhopadhyay (1995).

2. Algorithm Design

In what follows, we assume that a fixed sequence of jobs $\langle 1, 2, \ldots, j, \ldots, n\rangle$ is given as input, and any changes in the sequence are not permitted. Let $\pi_c = \langle 1, \ldots, n_c\rangle$ denote the sequence of cluster c, n_c the number of jobs in cluster c, $C = \{1, \ldots, \phi\}$ the set of clusters, and ϕ the number of clusters such that $\phi \leq n$. Hence, the chain of clusters $\pi_1, \ldots, \pi_\phi$ forms a complete sequence. We also denote $s_{j\in\pi_c}$ as the start time of job j of cluster π_c and $\texttt{ET}(\pi_c)$ as the objective evaluated for π_c. We first discuss the properties of the problem regarding distinct processing times. Next, we prove additional rules for the case with equal processing times to develop a more efficient algorithm.

2.1. Case 1. Distinct Processing Times

Lemma 1. *Given a fixed sequence $\pi = \langle 1, 2, \ldots, j, j+1, \ldots, n\rangle$, there exists an optimal schedule in which no idle time is placed between jobs j and $j+1$ if $d_{j+1} - d_j \leq p_{j+1},\ \forall j \in \pi$.*

Proof. Refer to Szwarc and Mukhopadhyay (1995). $\square$

Definition 1 (First-order cluster)**.** A partial sequence π is called a *First-order Cluster* if $d_{j+1} - d_j \leq p_{j+1},\ \forall j \in \pi$.

Definition 2 (Semi-active schedule)**.** A schedule is said to be *semi-active* if no job shift provides a lower cost schedule without changing any job sequence.

Lemma 2. *There exists a semi-active schedule for cluster π_c if*

$$s_{1\in\pi_c} \leftarrow \max\left\{0, \operatorname{med}\left\{d_j - \sum_{j'=1}^{j} p_{j'},\ j \in \pi_c\right\}\right\},$$

where $\operatorname{med}\{A\}$ is the median of set A.

Proof. From the definition of a cluster, no idle time between any consecutive jobs in π_c are allowed. Hence, $\texttt{ET}(\pi_c)$ can be redefined based on any job in π_c, e.g., job 1, and it follows that $s_j \leftarrow s_1 + \sum_{j'=1}^{j-1} p_{j'},\ \forall j \in \pi_c \setminus \{1\}$. Moreover, we obtain $\texttt{ET}(\pi_c) = \sum_{j\in\pi_c} \left| s_1 - (d_j - \sum_{j'=1}^{j} p_{j'})\right|$. It directly follows that $\texttt{ET}(\pi_c)$ is a 1-norm function with single variable s_1, and the median of set $\left\{d_j - \sum_{j'=1}^{j} p_{j'},\ j \in \pi_c\right\}$ *necessarily* minimizes $\texttt{ET}(\pi_c)$ Schwertman et al. (1990) (i). For checking the *sufficiency*, let $S = \left\{ s_1 - (d_j - \sum_{j'=1}^{j} p_{j'}),\ j \in \pi_c \right\}$ and the set function $f : 2^{|S|} \mapsto \mathbb{R}_+$ such that $f(\emptyset) = 0$. We deduce that $f = \sum_{\xi \in S} |\xi|$ over set S holds the following property,

$$f(M_1) + f(M_2) \geq f(M_1 \cap M_2) + f(M_1 \cup M_2),\ \forall M_1, M_2 \subseteq S,$$

since the left and the right hand sides contain the independent and non-separable modules. Hence, f is *sub-modular* (ii). From (i) and (ii), the results follow directly. $\square$

According to the definition of a cluster and Lemma 2, the start time of other jobs in π_c is immediately computed as $s_{j \in \pi_c} \leftarrow s_{1 \in \pi_c} + \sum_{j'}^{j-1} p_{j'},\ \forall j \in \pi_c \setminus \langle 1 \rangle$. Algorithm 1 reproduces the core result of Lemma 2 in function `fix_start`(π_c).

Algorithm 1 `fix_start`(π_c): Applying Lemma 2

```
1: Input: $\pi_c$
2: $s_{1 \in \pi_c} \leftarrow \max\left\{0, \mathrm{med}\left\{d_j - \sum_{j'=1}^{j} p_{j'},\ j \in \pi_c\right\}\right\}$
3: $s_{n \in \pi_c} \leftarrow s_{1 \in \pi_c} + \sum_{j \in \pi_c \setminus \langle n_c \rangle} p_j$
4: return $s_{1 \in \pi_c}, s_{n \in \pi_c}$
```

Lemma 3 (Second-order cluster)**.** *Consider two clusters π_c and π_{c+1} to which Lemma 2 is applied separately and at least a job of π_c overlaps with at least a job of π_{c+1}, i.e., $s_{n_c \in \pi_c} + p_{n_c \in \pi_c} \geq s_{1 \in \pi_{c+1}}$. There exists a semi-active schedule if two clusters form a new cluster $\pi' = \langle \pi_c, \pi_{c+1} \rangle$ with the start times $s_{1 \in \pi'} \leftarrow \max\left\{0, \mathrm{med}\left\{d_j - \sum_{j'=1}^{j} p_{j'},\ j \in \pi'\right\}\right\}$. We shall call π' a Second-order cluster.*

Proof. The final partial schedule should process cluster π_c immediately followed by cluster π_{c+1} in order to preserve the order of the input sequence. It directly follows that any idle time between two clusters does not reduce ET. Hence, it suffices to join two clusters as $\pi' \leftarrow \langle \pi_c, \pi_{c+1} \rangle$ and to apply Lemma 2 to π'. The later operations yield a semi-active schedule. $\square$

Lemma 3 helps to distinguish the types of clusters; let ϕ_1 and ϕ_2 denote the number of the first-order and the second-order clusters. Lemma 3 immediately ensures that all partial schedules of clusters remain semi-active in the following recursive scheme. In what follows, we construct an algorithm with the least ET based on the results obtained from Lemmas 1 through 3. Algorithm 2 consists of two primary routines, i.e., the clustering routine (CLR) and the schedule-correcting routine (SCR). The operations in lines 2 through 13 of the algorithm represents the CLR and detect the first-order clusters.

The SCR, which starts from the `while` loop, first checks whether the currently corrected cluster overlaps with the previous or the next cluster. If such a case happens, it joins two clusters, according to Lemma 3, and the product is a new second-order cluster. The next step checks checks any infeasibility caused by the overlapping jobs inside the new cluster and then produces its semi-active schedule. Hence, the algorithm runs recursively in the backward or the forward direction until the condition $c \leq \phi$ is not satisfied in the `while` loop. Algorithm 2 does not distinguish the first and second-order clusters. Function `update_list`(π, ϕ) refers to performing push operations on the items of a list. For example, if $\phi = 5$, and the SCR forces $\pi_c \leftarrow \langle \pi_c, \pi_{c+1} \rangle$ and $C \leftarrow \{1, \ldots, \phi - 1\}$ operations, then `update_list`(π, ϕ) re-index the clusters since π_{c+1} is redundant. This operation is efficiently done in $O(1)$.

Algorithm 2 Recursive scheme to generate minimum cost schedule for $1 \mid\mid$ ET

```
1: Input: $J \leftarrow \langle 1, 2, \ldots, n \rangle$        ▷ a fixed sequence with the numerically increasing order
2: for $c \in J$ do
3:     $\pi_c \leftarrow \langle \emptyset \rangle$
4: $c \leftarrow 1$
5: for $j \in J$ do
6:     if $\pi_c := \langle \emptyset \rangle$ and $d_{j+1} - d_j \leq p_{j+1}$ then
7:         $\pi_c \leftarrow \langle \pi_c, j, j+1 \rangle$
8:     else if $d_{j+1} - d_j \leq p_{j+1}$ then
9:         $\pi_c \leftarrow \langle \pi_c, j+1 \rangle$
10:    else
11:        $n_c \leftarrow |\pi_c|$
12:        $c \leftarrow c + 1$
13: $\phi \leftarrow c$ ; $c \leftarrow 1$
14: for $c \leq \phi$ do
15:     $s_{1\in\pi_c}, s_{n\in\pi_c} \leftarrow$ fix_start$(\pi_c)$
16: while $c \leq \phi$ do
17:     if $c < \phi$ and $s_{1\in\pi_{c+1}} - s_{n\in\pi_c} \leq \sum_{j\in\pi_c} p_j$ then
18:         $\pi_c \leftarrow \langle \pi_c, \pi_{c+1} \rangle$
19:         $\phi \leftarrow \phi - 1$
20:         update_list$(\langle \pi_1, \ldots, \pi_\phi \rangle, \phi)$
21:         if $s_{n_c\in\pi_c} - s_{1\in\pi_c} < \sum_{j\in\pi_c\setminus\langle n_c\rangle} p_j$ then
22:             $s_{1\in\pi_c}, s_{n\in\pi_c} \leftarrow$ fix_start$(\pi_c)$
23:     else if $c > 1$ and $s_{1\in\pi_c} - s_{n\in\pi_{c-1}} \leq \sum_{j\in\pi_{c-1}} p_j$ then
24:         $\pi_{c-1} \leftarrow \langle \pi_{c-1}, \pi_c \rangle$
25:         $c \leftarrow c - 1$ ; $\phi \leftarrow \phi - 1$
26:         update_list$(\langle \pi_1, \ldots, \pi_\phi \rangle, \phi)$
27:         if $s_{n_c\in\pi_c} - s_{1\in\pi_c} < \sum_{j\in\pi_c\setminus\langle n_c\rangle} p_j$ then
28:             $s_{1\in\pi_c}, s_{n\in\pi_c} \leftarrow$ fix_start$(\pi_c)$
29:     else
30:         $c \leftarrow c + 1$
31: $C \leftarrow \{1, \ldots, \phi\}$
32: return $\{s_{1\in\pi_c},\ \forall c \in C\}$
```

Lemma 4. *Every semi-active schedule is optimal for* $1 \mid\mid$ *ET.*

Proof. Refer to Davis and Kanet (1993). □

Corollary 1. *Given an arbitrary fixed sequence, Algorithm 2 yields the minimum cost schedule for* $1 \mid\mid$ *ET.*

Proof. The outputs of the algorithm are the number of clusters and the corresponding schedule to each cluster. Recall that the number of possible clusters can be 1, n, and $1 < \phi < n$. If we consider the simplest case, $\phi = n$, then the only way that the algorithm produces n clusters is $d_j - d_i > p,\ \forall i, j \in J$. If such a case happens, the algorithm outputs $s_j^* \leftarrow d_j - p,\ j \in J$ without entering into the SCR. For case $\phi < n$, according to Lemma 2 and 3, the SCR produces the final clusters both feasible and semi-active. According to Lemma 4, such a schedule is also globally optimal for $1 \mid\mid \texttt{ET}$. □

Corollary 2. *Algorithm 2 generates the minimum cost schedule for* $1 \mid\mid \texttt{ET}$ *in worst-case running time of* $O(\phi_1 n)$.

Proof. It is immediate the CLR is performed in a separate $O(n)$ iteration. The schedule correcting routine escalates to the worst-case scenario when the maximum number of the median function in $O(n)$ is evaluated. In the worst-case, the algorithm computes the median function $2\phi_1 - 1$ times to generate the final semi-active schedule. The results follow directly. □

2.2. Case 2. Equal Processing Times

In this section, we first investigate an optimal sequencing rule for $1 \mid p_j = p \mid \texttt{ET}$. Then, we revise the previous subsection results to develop a faster algorithm than Algorithm 2 for $1 \mid p_j = p \mid \texttt{ET}$.

Lemma 5. *There exists an optimal schedule for* $1 \mid p_j = p \mid \texttt{ET}$*, in which the jobs are ordered according to the Earliest Due Date (*`EDD`*) rule in which job j follows job i if* $d_i \leq d_j$.

Proof. Proof by contradiction. Suppose schedule π, which is not ordered w.r.t. EDD, is optimal. There must be two adjacent jobs $\langle j, i\rangle \in \pi$, say job j followed by job i, hence $s_i > s_j$, such that $d_i \leq d_j$. We perform a standard pairwise interchange on jobs j and i and call the new schedule π^r in which $\langle i, j\rangle \in \pi^r$. Immediately, π^r is feasible since jobs i and j have equal processing times and do not overlap under this interchange. Exactly six different interchange cases are constructed based on the relative ordering of jobs i and j among d_i and d_j. It can be easily verified that $\texttt{ET}(\pi) = \texttt{ET}(\pi^r)$ for cases $s_j < s_i \leq d_i < d_j$ and $d_i < d_j \leq s_j < s_i$; moreover, $\texttt{ET}(\pi) > \texttt{ET}(\pi^r)$ for cases $d_i \leq s_j < s_i \leq d_j$, $d_i \leq s_j < d_j \leq s_i$, $s_j < d_i \leq s_i \leq d_j$, and $s_j \leq d_i < d_j \leq s_i$. This contradicts the optimality of π and completes the proof. □

Without loss of generality, we re-index jobs of a first-order cluster w.r.t. `EDD` rule in the numerically increasing order.

Corollary 3. *Given a first-order cluster* π_c *ordered by* `EDD`*, there exists a semi-active schedule for* π_c *if*

$$\begin{aligned} &s_{1\in\pi_c} \leftarrow \max\{0, (\gamma_1 + \gamma_2)/2\}, \\ &\gamma_1 \leftarrow d_{\lfloor (n_c+1)/2\rfloor} - \lfloor (n_c+1)/2\rfloor p, \\ &\gamma_2 \leftarrow d_{\lceil (n_c+1)/2\rceil} - \lceil (n_c+1)/2\rceil p. \end{aligned} \tag{1}$$

Proof. From Lemma 2, we have

$$s_{1\in\pi_c} \leftarrow \max\{0, \text{med}\{d_j - jp,\ j \in \pi_c\}\},$$

and it directly follows that set $\{d_j - jp,\ j \in \pi_c\}$ is non-increasing since π_c preserves both the EDD rule and Lemma 1. $\square$

Algorithm 3 is practically more efficient than Algorithm 2 since it distinguishes the first-order clusters by plugging Corollary 3 in $O(1)$ iteration into the CLR. In other words, it fixes the start times of jobs in the first-order clusters before entering into the SCR.

Algorithm 3 Recursive scheme to generate minimum cost schedule for $1 \mid p_j = p \mid$ ET

```
 1: Input: list of sorted $\{s_j \leftarrow d_j,\ \forall j \in J\}$ based on EDD
 2: for $c \in J$ do
 3:     $\pi_c \leftarrow \langle \emptyset \rangle$
 4: $c \leftarrow 1$
 5: for $j \in J$ do
 6:     if $\pi_c := \langle \emptyset \rangle$ and $d_{j+1} - d_j \leq p$ then
 7:         $\pi_c \leftarrow \langle \pi_c, j, j+1 \rangle$
 8:     else if $d_{j+1} - d_j \leq p$ then
 9:         $\pi_c \leftarrow \langle \pi_c, j+1 \rangle$
10:     else
11:         $n_c \leftarrow |\pi_c|$
12:         $s_{1\in\pi_c} \leftarrow \max\{0, (\gamma_1 + \gamma_2)/2\}$        ▷ $\gamma_1$ and $\gamma_2$ are computed w.r.t. Corollary 3.
13:         $c \leftarrow c + 1$
14: $\phi_1 \leftarrow c$ ; $c \leftarrow 1$
15: while $c \leq \phi_1$ do
16:     if $s_{n_c\in\pi_c} - s_{1\in\pi_c} < (n_c - 1)p$ then
17:         $s_{1\in\pi_c} \leftarrow \max\{0, \text{med}\{d_j - jp,\ j \in \pi_c\}\}$
18:         $s_{n_c\in\pi_c} \leftarrow s_{1\in\pi_c} + (n_c - 1)p$
19:         if $c < \phi_1$ and $s_{1\in\pi_{c+1}} - s_{n_c\in\pi_c} \leq n_c p$ then
20:             $\pi_c \leftarrow \langle \pi_c, \pi_{c+1} \rangle$
21:             $\phi_1 \leftarrow \phi_1 - 1$
22:             update_list($\langle \pi_1, \ldots, \pi_{\phi_1} \rangle, \phi_1$)
23:         else if $c > 1$ and $s_{1\in\pi_c} - s_{n_{c-1}\in\pi_c} \leq n_{c-1} p$ then
24:             $\pi_{c-1} \leftarrow \langle \pi_{c-1}, \pi_c \rangle$
25:             $c \leftarrow c - 1$ ; $\phi_1 \leftarrow \phi_1 - 1$
26:             update_list($\langle \pi_1, \ldots, \pi_{\phi_1} \rangle, \phi_1$)
27:         else
28:             $c \leftarrow c + 1$
29: $C \leftarrow \{1, \ldots, \phi_1\}$
30: return $\{s_{1\in\pi_c},\ \forall c \in C\}$
```

Corollary 4. *Given a sequence sorted by* EDD*, Algorithm 3 yields an optimal schedule for* $1 \mid p_j = p \mid$ *ET.*

Proof. Algorithm 3 preserves the input sequence, which is ordered by EDD rule, to the end. The rest of the proof is trivial from Corollary 1. □

Corollary 5. *Algorithm 2 generates the minimum cost schedule for problem* $1 \mid\mid ET$ *in the worst-case running time of* $O(\phi_1 n - \phi_1^2/2 + \phi_1/2)$.

Proof. Like Algorithm 2, this algorithm handles the CLR in a separate $O(n)$ iteration. Again, the worst-case scenario for the SCR happens when the maximum number of the median function is evaluated in the $O(n)$ iteration. Recall that the CLR partitions all jobs into ϕ_1 clusters of n_c jobs. Without loss of generality, consider a case in which the first and the second first-order clusters are overlapping. After correction, it overlaps with the third first-order cluster. This continues to the ϕ_1th cluster. Hence, the algorithm needs to evaluate the median of $(n_1 + n_2)$, $(n_1 + n_2 + n_3)$, $\ldots$, and $\sum_{c=1}^{\phi} n_c$ jobs in the SCR. Hence, the worst-case running time is computed as $O(\phi_1 n - \phi_1^2/2 + \phi_1/2)$. □

References

Conway, R.W., Maxwell, W.L., Miller, L.W., 2003. *Theory of scheduling*. Courier Corporation.

Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C., 2009. *Introduction to algorithms*. MIT press.

Davis, J.S., Kanet, J.J., 1993. Single-machine scheduling with early and tardy completion costs. *Naval Research Logistics (NRL)* 40, 1, 85–101.

Emmons, H., Vairaktarakis, G., 2012. *Flow shop scheduling: theoretical results, algorithms, and applications*, Vol. 182. Springer Science & Business Media.

Garey, M.R., Tarjan, R.E., Wilfong, G.T., 1988. One-processor scheduling with symmetric earliness and tardiness penalties. *Mathematics of Operations Research* 13, 2, 330–348.

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

Graham, R.L., Lawler, E.L., Lenstra, J.K., Kan, A.R., 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of discrete mathematics*, Vol. 5. Elsevier, pp. 287–326.

Kim, Y.D., Yano, C.A., 1994. Minimizing mean tardiness and earliness in single-machine scheduling problems with unequal due dates. *Naval Research Logistics (NRL)* 41, 7, 913–933.

Mason, S.J., Jin, S., Jampani, J., 2005. A moving block heuristic for minimizing earliness and tardiness on a single machine with unrestrictive common due dates. *Journal of manufacturing systems* 24, 4, 328–338.

Schwertman, N.C., Gilks, A., Cameron, J., 1990. A simple noncalculus proof that the median minimizes the sum of the absolute deviations. *The American Statistician* 44, 1, 38–39.

Sourd, F., 2005. Punctuality and idleness in just-in-time scheduling. *European Journal of Operational Research* 167, 3, 739–751.

Szwarc, W., Mukhopadhyay, S.K., 1995. Optimal timing schedules in earliness-tardiness single machine sequencing. *Naval Research Logistics (NRL)* 42, 7, 1109–1114.

Tanaka, S., Fujikuma, S., Araki, M., 2009. An exact algorithm for single-machine scheduling without machine idle time. *Journal of Scheduling* 12, 6, 575–593.

Valente, J.M., Alves, R.A., 2005. Improved heuristics for the early/tardy scheduling problem with no idle time. *Computers & Operations Research* 32, 3, 557–569.

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.

Wan, L., Yuan, J., 2013. Single-machine scheduling to minimize the total earliness and tardiness is strongly np-hard. *Operations Research Letters* 41, 4, 363–365.

Wodecki, M., 2009. A block approach to earliness-tardiness scheduling problems. *The International Journal of Advanced Manufacturing Technology* 40, 7-8, 797–807.