Single-Machine Common Due Date Total Earliness/Tardiness Scheduling with Machine Unavailability

Kerem Bülbül
Sabancı University, Industrial Engineering, Orhanlı-Tuzla, 34956, Istanbul, Turkey
bulbul@sabanciuniv.edu

Safia Kedad-Sidhoum
Sorbonne Universités, Université Pierre et Marie Curie, Laboratoire LIP6 UMR 7606, 4 place Jussieu 75005 Paris, France
safia.kedad-sidhoum@lip6.fr

Halil Şen
Inria Bordeaux - Sud-Ouest Team ReAlOpt and Institut de Mathématiques de Bordeaux UMR 5251
351 cours de la Libération 33405 Talence, France
halil.sen@inria.fr

Abstract: Research on non-regular performance measures is at best scarce in the deterministic machine scheduling literature with machine unavailability constraints. Moreover, almost all existing works in this area assume either that processing on jobs interrupted by an interval of machine unavailability may be resumed without any additional setup/processing or that all prior processing is lost. In this work, we intend to partially fill these gaps by studying the problem of scheduling a single machine as to minimize the total deviation of the job completion times from an unrestrictive common due date when one or several fixed intervals of unavailability are present in the planning horizon. We also put a serious effort into investigating models with semi-resumable jobs so that processing on a job interrupted by an interval of machine unavailability may later be resumed at the expense of some extra processing time. The conventional assumptions regarding resumability are also taken into account. Several interesting cases are identified and explored, depending on the resumability scheme and the location of the interval of machine unavailability with respect to the common due date. The focus of analysis is on structural properties and drawing the boundary between polynomially solvable and $\mathcal{NP}$−complete cases. Pseudo-polynomial dynamic programming algorithms are devised for $\mathcal{NP}$−complete variants in the ordinary sense.

Keywords: single-machine; earliness/tardiness; common due date; unrestrictive; machine unavailability; maintenance; resumable; semi-resumable; non-resumable; $\mathcal{NP}$−complete; dynamic programming.

1. Introduction The continuous availability of resources is a dominant assumption in the machine scheduling literature. The overwhelming majority of scheduling research ignores the impact of events such as machine breakdowns, scheduled and preventive maintenance, etc., on the shop floor. If a machine in operation requires the uninterrupted attention of a worker, lunch, rest, and weekend breaks are further complicating factors for operations scheduling. Examples pointing to the diligence required in scheduling activities in the presence of machine unavailabilities are several in the literature. Benmansour et al. (2014) motivate their model, which integrates job scheduling decisions with periodic and flexible preventive maintenance activities, by arguing that preventive planned maintenance is an effective strategy for reducing the risk of breakdowns and the operating costs in production systems subject to random failures. This rationale is further supported by Garg and Deshmukh (2006) who contend that maintenance costs can comprise the largest part of an operational budget along with energy costs. Another common setting with maintenance activities incorporated into the schedule is due to the machine tool wear as cited by Low et al. (2010) in the context of the micro-drilling processes in PCB manufacturing. In general for the semiconductor industry, Graves and Lee (1999) state that "..., it is not uncommon to observe an operational machine in an idle state waiting for maintenance while jobs are waiting to be processed. This is due to lack of coordination between operators (or production planning personnel) and maintenance personnel." In a somewhat different setting from the chemical industry discussed by Rapine et al. (2012), jobs require intervention by an operator at their start and termination, and the machine may become unavailable as a consequence of operator unavailability.

As evident from the previous paragraph, scheduling problems involving machine unavailabilities arise in various physical manufacturing environments. Moreover, machine unavailabilities may also result from the tactical and operational level scheduling schemes (Schmidt, 2000). For instance, a prevalent scheduling practice in dynamic environments is to construct schedules in a rolling planning horizon framework. The natural overlap of two consecutive planning intervals translates into machine unavailabilities in the latter planning interval because resources may have already been committed based on earlier scheduling decisions. An analogous problem setting occurs in the context of real-time operating systems, where programs with low priority have to be scheduled on the processor(s) around those with higher priority, or multi-user computer system applications, where new jobs have to be executed in addition to those already scheduled. In both cases, a scheduling model captures periods assigned to tasks of higher priority / earlier arrival time as intervals of unavailability.

Scheduling problems with machine unavailability constraints have received considerable attention from researchers in the last two decades motivated by abundant practical examples as discussed above. A rich set of features and characteristics have been considered, and a brief taxonomy is in order. The first differentiating dimension is the information available about the occurrence and length of the unavailabilities. Studies focusing on unpredictable machine breakdowns/repairs and maintenance required due to a random drift toward unacceptable product quality are stochastic in nature and deemed out of scope here. We refer the interested reader to Federgruen and Mosheiov (1997) and Liu and Sanlaville (1997) – two widely cited works in this area. The remaining properties pertain to the deterministic scheduling problems with machine unavailabilities, and next in the list is the structure of the scheduling objective: regular versus non-regular. In the scheduling literature, it is well-established that regular objective functions, which are non-decreasing in the job completion times, are generally less challenging compared to non-regular objectives from theoretical and/or practical viewpoints. The third feature describes the level of control on scheduling the unavailability intervals, and there are two main streams of research here. In one stream, the timing of the machine unavailabilities is an external input; that is, the associated start and completion times are fixed. At times, an additional periodicity requirement may be imposed. The durations may be identical for all intervals of unavailability or may be allowed to change. The other stream targets the integration of the job and maintenance scheduling decisions and treats the start time of an interval of unavailability as a variable. There is often an upper bound on the time elapsed between two consecutive unavailabilities, and such settings are frequently referred to as problems with flexible and/or periodic maintenance. A further defining characteristic is the number of intervals of unavailability in the planning horizon: single versus several. Finally, any scheduling problem with unavailability constraints must specify how the remaining processing of a job interrupted by an interval of unavailability is to be handled. If all prior processing is lost, and processing must be re-started from scratch after the machine becomes available again, then we have a *non-resumable* problem. Alternatively, a problem setting is referred to as *resumable* if the processing of an interrupted job resumes without any additional processing and/or setup following the interruption. In between these two extremes, *semi-resumability* – initially introduced by Lee (1999) – implies that an interrupted job may continue its execution at the expense of extra processing time and/or setup. Some papers refer to the resumable and non-resumable cases as *preemptive* and *non-preemptive*, respectively, but we adopt the earlier terminology. We also use the term *break* for an interval of machine unavailability in the rest of the paper. Based on this classification of the literature, we tackle a deterministic single-machine problem with machine unavailability constraints, in which all information about the jobs and the breaks is known with certainty at the time of

planning. The objective is non-regular and minimizes the total absolute deviation of the job completion times from an *unrestrictive* common due date as defined precisely in Section 2. We study several variants, and if there are multiple breaks, their lengths may be non-identical. There is no periodicity assumption. All three different cases regarding resumability are analyzed. In the sequel, we provide pointers to the existing studies in an effort to position our work with respect to the literature by sticking to the taxonomy laid out above. The focus is on the single-machine environment as it creates the context for the current study, and in our coverage of the literature with regular scheduling objectives we do not delve into the specifics of the solution methods, but instead focus on the attributes of the problems attacked so far. For an in-depth analysis of the literature – including the various complexity results, polynomial and enumerative optimal methods, heuristics and the associated approximation bounds, the interested reader is referred to the comprehensive surveys by Schmidt (2000) and Ma et al. (2010). We ultimately conclude this section by summarizing our contributions.

Virtually all scheduling research with machine unavailabilities ignores non-regular objective functions. One of the earliest examples of research on regular objective functions in the single-machine literature is by Adiri et al. (1989), who establish that the single-machine total completion time problem with a single break is $\mathcal{NP}-$complete. The timing and the length of the break are known a priori, and the jobs are non-resumable. An exact branch-and-bound (B&B) algorithm is developed by Leon and Wu (1992) for minimizing the maximum lateness on a single-machine under the same constraints, except that the planning horizon may include several breaks and jobs are released at different times. Lee (1996) continues in the same vein of research by characterizing and developing algorithms for one fixed break per machine in the single- and parallel-machine environments under both the resumability and non-resumability assumptions for several regular scheduling criteria: makespan, maximum lateness, total (weighted) completion time, and total number of tardy jobs. The state-of-the art for the single-machine total weighted completion time problem with a single fixed break and non-resumable jobs is defined by Kacem et al. (2008) and Kacem and Chu (2008), who devise exact algorithms which scale up to 3000 and 6000 jobs, respectively. Wang et al. (2005) attack the resumable version of the single-machine total weighted completion time problem with multiple fixed breaks. They prove that the problem is $\mathcal{NP}-$hard in the strong sense and provide approximation results for two special cases. Laalaoui and M'Hallah (2016) take on the objective of maximizing the weighted number of scheduled non-resumable jobs on a single machine over a planning horizon which incorporates a predefined number of fixed breaks of possibly different durations. Another recent piece of research with one fixed maintenance activity in the planning horizon is contributed by Yin et al. (2016b). These authors develop two pseudo-polynomial time dynamic programming algorithms for a set of non-resumable jobs on a single machine with the goal of minimizing the total amount of late work, where the length of processing performed on a job past its due date is labeled as late. Approximation results are also provided. In the realm of periodic and/or flexible maintenance with regular objective functions on a single machine, Ji et al. (2007), Low et al. (2010), and Cui and Lu (2017) are concerned with the integrated scheduling of non-resumable jobs and several periodic maintenance activities as to minimize the makespan. Chen (2009), Lee and Kim (2012), and Liu et al. (2016) consider the identical setting under the performance measure of minimizing the number of tardy jobs. In these six papers, two consecutive maintenance breaks are separated exactly by a fixed pre-defined duration, and the breaks are all of equal length, except in Low et al. (2010) and Cui and Lu (2017), who allow for flexibility in the start time of a break. The recent work by Drozdowski et al. (2017) has a fresh perspective on flexible maintenance activities. These authors observe that in practice maintenance activities are also often triggered by the number of jobs performed since the completion of the most recent maintenance. Under this

setting, the authors explore various problem variants with the objective of minimizing the makespan or the maximum lateness. The work by Graves and Lee (1999) is an exception to the body of work discussed so far, because these authors take semi-resumable jobs into account. More specifically, a job interrupted by a break may be carried on after the break following a job-dependent fixed setup time. The objective is either to minimize the total weighted completion time or the maximum lateness on a single machine, and the length of the planning horizon justifies just a single break or a maximum of two. In both cases, a flexible break must be performed within a pre-determined fixed period of time. Detienne (2014) adopts the exact same semi-resumability scheme and develops computationally effective mixed-integer programming formulations for minimizing the weighted number of late jobs with several fixed breaks in the planning horizon and no periodicity requirement. The conventional resumable and non-resumable cases are also considered. To the best of our knowledge, these are the only two pieces of research in the single-machine literature, which handle the case of semi-resumable jobs. For an overview of shop scheduling problems with machine unavailability constraints, the reader is referred to the survey papers by Schmidt (2000) and Ma et al. (2010), and the recent papers by Yoo and Lee (2016), Yin et al. (2016a), Yin et al. (2017), and Huo (2017).

In contrast to a fairly rich literature on machine scheduling problems with unavailability constraints under regular performance measures, papers attacking non-regular objectives under similar constraints are quite rare. A first example is set by Mannur and Addagatla (1993). Similar to our work, these authors address the problem of minimizing the total absolute deviation of the job completion times from a common due date on a single machine with several fixed breaks in the planning horizon. However, the attention is restricted to non-resumable jobs, and two heuristics are proposed based on the decomposition of the planning horizon into several independent processing intervals by the breaks. This early piece of work was only followed up in the last few years starting with Benmansour et al. (2011), who set up a mixed-integer programming formulation for minimizing the total weighted earliness/tardiness (E/T) with a common due date. Jobs are non-resumable, and a single machine is unavailable periodically for a fixed maintenance duration. In this stream of research, Low et al. (2015) incorporate a single fixed planned maintenance period with non-resumable jobs into their problem of minimizing the sum of the absolute deviations of the job completion times from a common due date on a single machine. A mixed-integer programming formulation of the problem at hand is followed by the development of an ant colony heuristic for large-scale instances. In addition, the authors also tackle a special case under an unrestrictiveness assumption, where the due date falls into the break. This setting is identical to that in Section 5.1 of our paper. However, we provide a substantially more concise, streamlined, and easy-to-follow analysis by exposing a certain discrete convexity property, which is then exploited algorithmically. Molaee et al. (2011) and Benmansour et al. (2014) take a different path from these three papers focusing on additive E/T criteria and incorporate the maximum earliness and/or the maximum tardiness into their objectives. More specifically, Molaee et al. (2011) first focus on minimizing the maximum earliness and then shift their attention to the bi-objective problem of identifying the Pareto frontier for minimizing the maximum earliness and the number of tardy jobs on a single machine simultaneously. There is a single fixed break in the planning horizon under the non-resumability assumption. For either type of problem, the authors first derive some structural properties, lower bounds, and dominance rules, and then leverage these for devising a heuristic and a B&B method. Benmansour et al. (2014) are concerned with the single-machine scheduling problem of minimizing the weighted sum of the maximum earliness and the maximum tardiness costs, where the jobs share a restrictive common due date. The machine is required to undergo periodic and flexible maintenance of fixed length. An upper limit on the time elapsed between

two consecutive maintenance breaks is present, and the jobs are non-resumable. These two features lend the problem a bin packing structure. A heuristic relying on this structure is proposed following a mathematical programming formulation of the problem.

The taxonomy and review of the literature reveals a clear void regarding E/T problems with machine unavailability constraints, and we intend to partially fill this gap in this paper. As pointed out previously in this section, our focus is on minimizing the total absolute deviation of the job completion times from an unrestrictive common due date on a single machine with one or several non-periodic fixed breaks in the planning horizon, and we cover all three cases with respect to resumability. In the E/T literature with additive objectives, only the single-machine unrestrictive common due date problems with job-independent unit E/T penalties are polynomially solvable – see Baker and Scudder (1990) and Kanet and Sridharan (2000) for the early results in this field. Given this fact and the lack of a rigorous understanding of the structural properties of E/T problems with machine unavailability constraints in the literature, we consider it a worthy research question to investigate how the structural properties of an originally simple E/T problem are affected by the presence of machine unavailability constraints. We elaborate more on this at the end of Section 2, following a formal introduction of our problem.

Our primary technical contribution in this paper is that for a given problem variant we consider, we either present a polynomial-time optimal algorithm or prove its $\mathcal{NP}$−completeness. For $\mathcal{NP}$−complete variants in the ordinary sense, such a result is accompanied by a dynamic programming algorithm of pseudo-polynomial complexity as appropriate. Ultimately, we provide a fairly complete characterization of the single-machine unrestrictive common due date total E/T problem with machine unavailability constraints in our setting and generally succeed in drawing the boundary between polynomially solvable and $\mathcal{NP}$−complete problems for the variants we explore. From a modeling perspective, we have other contributions on top of that directly to the E/T literature. In contrast to the overwhelming majority of the literature, we take both a single and several fixed – not necessarily periodic – breaks into account. It turns out that these two types of problems are quite different in nature. Finally, a major contribution of this paper is that all three assumptions regarding resumability are analyzed in detail. In their conclusions, Ma et al. (2010) point out that only a handful of papers are available on semi-resumability and emphasize semi-resumability as a future research direction based on its prevalence in the industry. We make a serious effort to examine this case in our models.

In Section 2, a formal definition of our problem is presented, and we then proceed to establish the strong $\mathcal{NP}$−completeness of our problem with several breaks in the following section. A set of preliminaries is discussed in Section 4, and Sections 5-6 are dedicated to the analysis of a single break. Several interesting cases are identified and investigated, depending on the location of the break with respect to the common due date and the assumptions regarding resumability. We conclude with future research prospects in Section 7.

2. Problem Statement In the most general statement of the single-machine E/T scheduling problem with machine availability restrictions, a total of n jobs are to be processed non-preemptively on a single machine. If the processing of a job spans a break, then we label this job as an *interrupted* job. Each job i has a nominal processing time $p_i > 0$ and incurs a break penalty (extra amount of processing) if its execution window intersects with a break. In the rest of the paper, we assume that the nominal processing times are in the Longest Processing Time (LPT) order, i.e., $p_1 \geq p_2 \geq \cdots \geq p_n$, unless specified explicitly otherwise. The vector of processing times is denoted by $\mathbf{p}$. The actual processing time $\bar{p}_i(s_i)$ of job i depends on its start time s_i, and its exact form is specified in the sequel. In addition, a due date d_i, a unit earliness cost α_i and a unit tardiness cost β_i are associated with job i. All jobs are ready for processing at time zero. There is a total of K breaks in

the planning horizon $[0, T]$, where break k is given by the time interval $\left[B_s^k, B_f^k\right]$ with a length of $b_k = B_f^k - B_s^k$ time units. $B_f^k < B_s^{k+1}$ holds for all $k = 1, \ldots, K-1$. All processing times, the due dates, and the break start and finish times are assumed to be integral. This general problem formulation with distinct job-dependent due dates and multiple breaks is strongly $\mathcal{NP}$−hard because it subsumes the strongly $\mathcal{NP}$−hard single-machine scheduling problem of minimizing the total weighted tardiness with job-dependent penalties and distinct job-dependent due dates. A time-indexed binary integer programming formulation for this general problem statement is provided in the appendix. Finally, we compute the actual processing time $\bar{p}_i(s_i)$ of job i as

$$\bar{p}_i(s_i) = \begin{cases} p_i + b_k + l_i\left(s_i, B_s^k\right) & \text{if } s_i < B_s^k < s_i + p_i \text{ for some } k \in \{1 \ldots, K\}, \\ p_i, & \text{otherwise,} \end{cases} \tag{1}$$

where the break penalty $l_i(s_i, B_s^k)$ of job i is calculated as follows:

$$l_i(s_i, B_s^k) = \left\lceil \left(B_s^k - s_i\right)\Theta \right\rceil. \tag{2}$$

Note that the break penalty represents the fraction of work completed before the break which needs to be repeated after the break. The structure of (2) captures all three cases of *resumable*, *non-resumable*, and *semi-resumable* jobs with $\Theta = 0$, $\Theta = 1$, and $0 < \Theta < 1$, respectively. The values $\Theta > 1$ are not relevant because such values imply that delaying the start of a currently interrupted job decreases its completion time. Figure 1 presents the length of the processing of an interrupted job after the break with respect to that before the break for various values of Θ. The computations for $\Theta = 0$ and $\Theta = 1$ are straightforward because the former implies that no work is repeated following an interruption, and all prior processing is lost in the latter case. To illustrate the calculations for the intermediate values of Θ, assume that job j with $p_j = 10$ is interrupted by break k after receiving five units of processing. According to (2), the break penalty evaluates to $\left\lceil \left(B_s^k - s_j\right)\Theta \right\rceil = \lceil 5\Theta \rceil$. Thus, the total amount of work performed after the break is $(10 - 5) + \lceil 5 * 0.3 \rceil = 7$ and $(10 - 5) + \lceil 5 * 0.7 \rceil = 9$ for $\Theta = 0.3$ and $\Theta = 0.7$, respectively. It should also not go unnoticed that the computation of the actual processing time specified in (1) implicitly assumes that a job will not be interrupted more than once. For reasons that will become evident at the end of Section 3, this is the prevalent case in this paper, and this formula is sufficient for our purposes. However, the logic underlying (1) can easily be generalized to compute the actual processing time of a job interrupted several times in succession. In any case, we stress that our general problem statement and the complexity proof in the next section do not depend on an assumption that a job is interrupted at most once.

The computational complexity of the general problem definition discussed up to this point and the scarcity of papers taking on E/T objectives with machine unavailability restrictions prompt us to identify the special cases which are amenable to optimal solution methods of polynomial or pseudo-polynomial complexity. To this end, we follow suit with the E/T literature at large, which grew out from the study of common due date problems. For this class of problems, the literature branches out into two main paths. In the case of a restrictive common due date, the imminence of the due date has an impact on the optimal schedule and adds an additional layer of complexity. Problems with an unrestrictive common due date $d \geq \sum_{i=1}^{n} p_i$ such that $d_i = d, i = 1, \ldots, n$, are in general theoretically and/or practically easier compared to their restrictive counterparts and have more structure. For this reason, unrestrictive common due date problems are typically tackled first in the literature, and the outcomes are then possibly leveraged in the design of optimal or heuristic algorithms for the corresponding restrictive common due date problems in subsequent research. We take a similar approach in this paper by restricting our attention to an unrestrictive common due date with the hope that our study will pave the way for follow-up research on various possible extensions. For our problem,

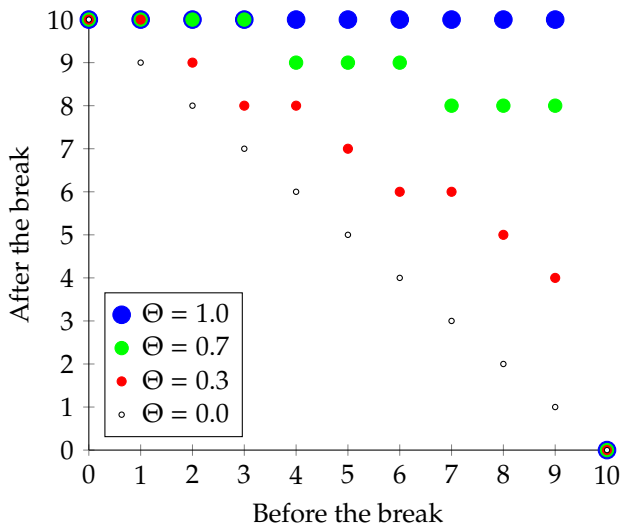


Figure 1 Amount of processing an interrupted job j with $p_j = 10$ receives upon resuming as a function of the length of its processing before the break.

a sufficient condition for the unrestrictiveness of the common due date is given as $\sum_{i=1}^{n} p_i \leq \min\{d, B_s^1\}$. Furthermore, as underlined toward the end of Section 1, even minimizing the total weighted E/T with an unrestrictive common due date remains $\mathcal{NP}$−complete unless the unit E/T weights are job-independent, and we assume that $\alpha_i = \beta_i = 1, i = 1, \ldots, n$, in the rest of the paper. In the next section, we settle the complexity of the unrestrictive common due date problem with semi-resumable and non-resumable jobs when multiple breaks are present in the planning horizon before we proceed with our formal analysis of the unrestrictive common due date problem with a single break in the rest of the paper.

3. The Non-resumable & Semi-resumable Unrestrictive Common Due Date Total E/T Problem with Multiple Breaks is Strongly $\mathcal{NP}$−Hard The decision version of the single-machine unrestrictive common due date total E/T scheduling problem with multiple breaks – referred to as **ET–MB** – and $0 < \Theta \leq 1$ requires a *yes/no* answer to the following question: Does there exist a feasible schedule S with a total cost $f(S)$ no larger than some integer y_0?

The proof proceeds by a reduction from the `3-PARTITION` problem defined as follows: Given an integer $b > 0$ and a set of $3t$ positive integers $X = \{x_1, x_2, \ldots, x_{3t}\}$ with $\frac{b}{4} < x_i < \frac{b}{2}$, $i = 1, \ldots, 3t$, and $\sum_{i=1}^{3t} x_i = tb$, is it possible to partition X into t mutually disjoint three element subsets $X_k \subset X, k = 1, \ldots, t$, such that $\sum_{i \in X_k} x_i = b$ for $k = 1 \ldots, t$? Without loss of generality, we also assume that $x_{i-1} \leq x_i$ for $i = 2, \ldots, 3t$. In the sequel, we prove that `3-PARTITION` has a *yes* answer if and only if the particular instance **I1** of the decision version of **ET–MB** described in the following is a *yes*-instance as well. The construction of **I1** is clearly polynomial in the size of the `3-PARTITION` instance.

In the instance **I1**, the common due date is set to $d = 2y_0 + tb + 1$, where $y_0 = \sum_{k=1}^{t} ((k-1)(3b+3) + 3b)$. The value of Θ may be chosen arbitrarily from the interval $(0, 1]$. **I1** includes $3t$ "partition" jobs J_i with $p_i = x_i$ for $i = 1, \ldots, 3t$, an additional dummy job J_0 with $p_0 = y_0$, and $t + 1$ breaks such that $B_s^0 = d - y_0 - 1$, $B_f^0 = d - (\lceil (p_0 - 1)\Theta \rceil + 1)$, $B_s^k = B_f^k - 1$, $B_f^k = d + k(b+1)$ for $k = 1, \ldots, t-1$, and $B_s^t = d + t(b+1) - 1$, $B_f^t = d + y_0 + 1$. Observe that the partition jobs $J_i, i = 1, \ldots, 3t$, are in the Shortest Processing Time (SPT) order, and the common due date d satisfies the sufficiency condition $\sum_{i=0}^{3t} p_i \leq \min\left\{B_s^0, d\right\}$ for unrestrictiveness stipulated at the end of Section 2 because $\sum_{i=0}^{3t} p_i = y_0 + tb \leq \min\left\{B_s^0, d\right\} = B_s^0 = d - y_0 - 1 = (2y_0 + tb + 1) - y_0 - 1 = y_0 + tb$.

Lemma 3.1 *If the partitioning of X into mutually disjoint three element subsets $X_1, X_2, \ldots, X_t$, corresponds to a solution of* `3-PARTITION`*, then there exists a feasible schedule S_0 for* ***I1*** *with a total E/T cost of at most y_0.*

Proof. Assume that $X_1, X_2, \ldots, X_t$ yield a solution for `3-PARTITION`. Consider the feasible schedule S_0 illustrated in Figure 2, in which the jobs in sets X_k, $k = 1, \ldots, t$, are scheduled in increasing order of their indices. Note that for brevity of notation, we employ $X_k, k = 1, \ldots, t$, also as index sets for jobs scheduled in specific intervals in S_0.

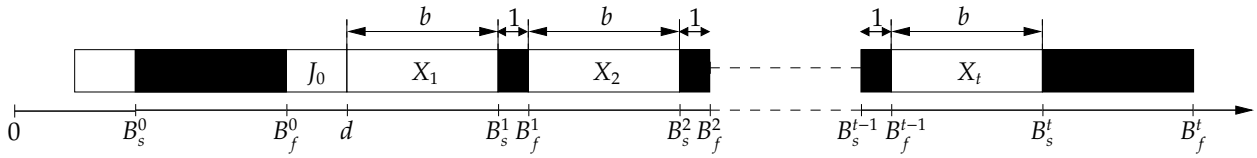


Figure 2 Schedule S_0.

The cost of the schedule S_0 is:

$$f(S_0) = 0 + \sum_{k=1}^{t} \left(3(k-1)(b+1) + \sum_{i \in X_k} \sum_{\substack{j \in X_k \\ j \le i}} p_j \right) \tag{3}$$

$$< \sum_{k=1}^{t} \left(3(k-1)(b+1) + \frac{3b}{2} + \frac{2b}{2} + \frac{b}{2} \right) = \sum_{k=1}^{t} (3(k-1)(b+1) + 3b) = y_0. \tag{4}$$

In (3), the expression $(k-1)(b+1)$ is the delay of the start time of the first of the three jobs in X_k with respect to d, and adding $\sum_{\substack{j \in X_k \\ j \le i}} p_j$ to this quantity yields the tardiness of job $i \in X_k$. The strict inequality in the transition from (3) to (4) follows from the fact that there are exactly three jobs in each X_k, $k = 1, \ldots, t$, and that the processing times of all jobs are less than $\frac{b}{2}$ by definition. (3)-(4) certify that there exists a feasible schedule S_0 for **I1** with a total cost of $f(S_0) \le y_0$ if $X_1, X_2, \ldots, X_t$ constitute a solution for `3-PARTITION`. □

Conversely, suppose that there exists a feasible schedule S for **I1** such that $f(S) \le y_0$.

Lemma 3.2 *The following properties must hold for a feasible schedule S of* ***I1*** *if $f(S) \le y_0$:*

i. No job completes before B_s^0 or after B_f^t.

ii. The dummy job J_0 is scheduled at the first position.

iii. The dummy job J_0 completes at the common due date d.

Proof.

i. This is due to the choice of the lengths and positions of the first and last breaks – i.e., $d - B_s^0 > y_0$ and $B_f^t - d > y_0$. So, a job which completes before the first break or after the last break incurs a cost larger than y_0.

ii. By contradiction. Assume that J_0 with $p_0 = y_0$ is not the first job in S with a total cost of $f(S) \le y_0$ and note that **I1** includes at least three jobs in addition to J_0 because $t \ge 1$. In order to calculate a lower bound on $f(S)$, we consider the corresponding instance of the unrestrictive common due date total E/T problem – a special case of **ET–MB** with no breaks. The discussion immediately following Property 4.1 in the next section reveals that the total E/T cost in the absence of any breaks cannot be less than $y_0 + 3$, unless J_0 occupies the initial position. Incorporating breaks cannot decrease the cost and $f(S) \ge y_0 + 3$ must hold in this case as well, contradicting the upper bound y_0 assumed on $f(S)$.

iii. If $f(S) \leq y_0$, Properties i-ii imply that no job can finish its processing before J_0 and the completion time of J_0 will be later than B_f^0. If J_0 is interrupted just by break 0, the amount of processing J_0 receives following the break is given by $\left\lceil \left(B_s^0 - s_0\right)\Theta \right\rceil + \left(p_0 - \left(B_s^0 - s_0\right)\right)$ and is a non-increasing function of the work $\left(B_s^0 - s_0\right)$ completed before the break. This result follows directly from $\left\lceil \left(B_s^0 - s_0 + 1\right)\Theta \right\rceil + \left(p_0 - \left(B_s^0 - s_0 + 1\right)\right) - \left\lceil \left(B_s^0 - s_0\right)\Theta \right\rceil - \left(p_0 - \left(B_s^0 - s_0\right)\right) = \left\lceil \left(B_s^0 - s_0\right)\Theta + \Theta \right\rceil - \left\lceil \left(B_s^0 - s_0\right)\Theta \right\rceil - 1 \leq 0$ because $0 < \Theta \leq 1$. Therefore, if J_0 is interrupted by break 0, then it will need to stay on the machine for a minimum of $\lceil (p_0 - 1)\Theta \rceil + 1$ time units after the break, and it will terminate no earlier than at time $B_f^0 + \lceil (p_0 - 1)\Theta \rceil + 1 = d - (\lceil (p_0 - 1)\Theta \rceil + 1) + (\lceil (p_0 - 1)\Theta \rceil + 1) = d$. If J_0 is not interrupted, its minimum possible completion time is $B_f^0 + p_0 = d - (\lceil (p_0 - 1)\Theta \rceil + 1) + p_0 \geq d$.

Furthermore, if $f(S) \leq y_0$, Properties i-ii require that all tb units of work on jobs $J_1, \ldots, J_{3t}$ must fit between the completion time of J_0 and B_s^t. This is only attainable if the completion time of J_0 is no larger than d because the total availability of the machine in the time interval $[d, B_s^t]$ is exactly tb time units. Combined with the arguments in the previous paragraph, which establish that J_0 cannot be finished before d if $f(S) \leq y_0$, we conclude that the completion time of J_0 is d if $f(S) \leq y_0$.

□

Lemma 3.3 *If there exists a feasible schedule S of **I1** with $f(S) \leq y_0$, then the underlying instance of the* `3-PARTITION` *problem is a yes-instance.*

Proof. From Lemma 3.2, job J_0 must be scheduled first and completes at time d, and there must be no job completions before B_s^0 and after B_f^t if $f(S) \leq y_0$. This leaves t separate blocks of time, each of length exactly b, for the remaining jobs J_i, $i = 1, \ldots, 3t$, between d and B_s^t. This is just enough time to accommodate these jobs only if no job is interrupted because the total nominal processing time of these $3t$ jobs is tb time units and $\Theta > 0$. Consequently, each block is filled entirely and a job is processed completely in a single block; that is, the jobs $J_1, \ldots, J_{3t}$ are partitioned into three-job subsets $X_1, X_2, \ldots, X_t$ such that $\sum_{i \in X_k} p_i = b$ for all $k = 1, \ldots, t$, providing us with a solution of the associated `3-PARTITION` instance. □

Theorem 3.4 *The decision version of **ET–MB** is $\mathcal{NP}$–complete in the strong sense for $0 < \Theta \leq 1$.*

Proof. The decision version of **ET–MB** is clearly in $\mathcal{NP}$. Furthermore, the construction of **I1** is polynomial in the size of the underlying `3-PARTITION` instance, and **I1** is a *yes*-instance of **ET–MB** if the associated `3-PARTITION` instance is a *yes*-instance (Lemma 3.1). The converse follows from Lemma 3.3. These two lemmas complete the polynomial transformation from `3-PARTITION` to **ET–MB** and yield the desired result because `3-PARTITION` is $\mathcal{NP}$–complete in the strong sense. □

The careful reader may have noticed that the complexity analysis fails to go through if the jobs are resumable, i.e., if $\Theta = 0$, because we can no longer claim that each job is performed exclusively in a single block in the proof of Lemma 3.3. The complexity of **ET–MB** with $\Theta = 0$ remains open.

Graves and Lee (1999) state that in their experience "there are very few (one or two) maintenance periods during a planning horizon," and more than two maintenance periods scheduled during a planning horizon are rare in practice. Motivated by this claim and having established the difficulty of minimizing the total E/T under an unrestrictive common due date and multiple breaks, we shift our focus to the special case of **ET–MB** with a single break – referred to as **ET–SB** – in the rest of the paper. For brevity of notation, the superscripts are omitted from B_s^1 and B_f^1.

4. Preliminary Insights The problem of minimizing the absolute deviation from an unrestrictive common due date on a single machine (**UCDD-ADev**) is one of the easiest of all earliness/tardiness scheduling problems. For the rest of the paper, we tackle various generalizations of this problem with a single break. Therefore, in this section we first list the fundamental structural properties exhibited by **UCDD-ADev**, and then proceed to illustrate that these properties do not necessarily hold for **ET–SB**. The lack of these properties renders solving **ET–SB** to optimality a substantially harder endeavor in general. In the following sections, we analyze various variants of **ET–SB** differentiated by the position of the break with respect to the due date and the value of Θ. We start with the simplest setting in Section 5.1 and work our way up toward tougher problem types.

It is fairly straightforward to show that there exists an optimal solution for **UCDD-ADev** with the following properties (Kanet, 1981):

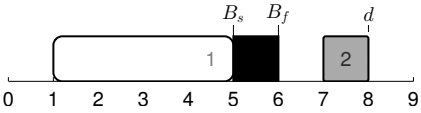
Property 4.1

a. *There is no inserted idle time in the schedule.*
b. *One job completes precisely at the due date.*
c. *The optimal schedule is V-shaped. That is, the jobs which complete before or at the due date are in the LPT order, while the remaining jobs are sequenced in the SPT order.*

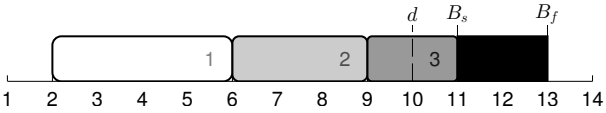
These properties were initially exploited in a seminal paper by Kanet (1981) to design a polynomial time optimal algorithm of complexity $O(n^2)$ for **UCDD-ADev**. However, researchers subsequently realized that this complexity can be dropped to $O(n \log n)$ (Bagchi et al., 1986, Hall, 1986). The key observation is that the processing time of an early/on-time job contributes to the earliness of every preceding job, and the processing time of a tardy job is counted toward the tardiness of every job completed later, including its own. Based on this rationale, solving **UCDD-ADev** to optimality boils down to matching the processing times to the set of n positional weights $c = \{0, 1, 1, 2, 2, 3, 3, \ldots\}$ sorted in non-decreasing order, and the optimal objective function value is then given by $\sum_{j=1}^{n} c_j p_j$ because the jobs are labeled in the LPT order. The basic idea of this algorithm is generalized in Algorithm 1 in Section 5.1 for solving **ET–SB** with a *straddling break* – that is, the break contains the due date – and non-resumable jobs optimally. This is only possible because this variant of **ET–SB** preserves the Properties 4.1a and 4.1c. It turns out that the second property is not essential for the sequencing decisions, but only helps determine the completion times given the job processing sequence.

Figure 3 attests to the fact that each of the Properties 4.1a - 4.1c may be violated at optimality for an instance of **ET–SB**. In particular, the machine is left idle for one time unit following the break in the unique optimal schedule of the instance in Figure 3a – not complying with Property 4.1a. Similarly, the unique optimal solution in Figure 3b exhibits a *straddling job* – a job that starts before the due date and completes tardy – in contradiction to Property 4.1b. Finally, in any one of the (symmetric) optimal solutions of the instance in Figure 3c, the break is preceded by a job of duration of five time units, but the length of the job following the break is shorter. Both jobs start and complete after the due date and break the SPT order in violation of Property 4.1c. The first two cases are relatively common and illustrative examples are simple to construct. However, the absence of the V-shaped property is more subtle and occurs less frequently.

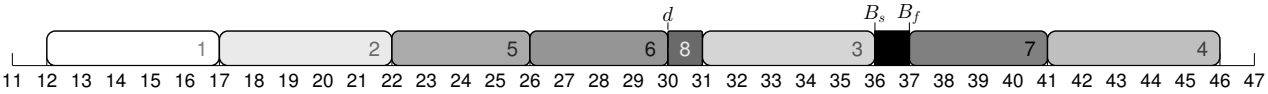
The careful reader may question whether the presence of interrupted and straddling jobs in an optimal schedule are mutually exclusive, which arguably would help reduce the search space in an enumerative algorithm. However, the examples in Figure 3b and Figure 4 demonstrate the lack of such a structural property. In Figures 3b and 4a, there only exist a single straddling and a single interrupted job in the optimal



(a) No optimal schedule without inserted idle time ($\Theta = 1$).



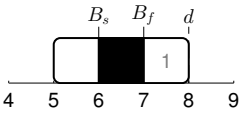
(b) No optimal schedule without a straddling job ($\Theta = 0$).



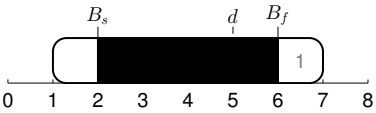
(c) No optimal schedule obeys the V-shaped property ($\Theta = 1$).

Figure 3 Properties 4.1a - 4.1c are not necessarily satisfied in **ET–SB**.

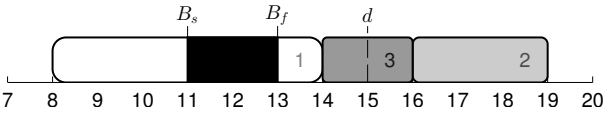
schedule, respectively. In Figure 4b, job 1 is both interrupted and straddling, while two separate interrupted and straddling jobs are present in Figure 4c.



(a) There only exists an interrupted job in the optimal schedule ($\Theta = 0$).



(b) The same job is both straddling and interrupted in the optimal schedule ($\Theta = 0$).



(c) Two distinct straddling and interrupted jobs in the optimal schedule ($\Theta = 0$).

Figure 4 Interrupted and straddling jobs in an optimal schedule of **ET–SB**.

The next section presents our analysis of the simpler case with a single straddling break, and we then proceed to the characterization and solution of **ET–SB** with a non-straddling break in Section 6.

5. Single Straddling Break

5.1 Non-resumable Jobs This variant of our problem is referred to as **ET–SStB-NonRes** and preserves the V-shaped property described in Property 4.1c. That is, all jobs preceding and succeeding the break are sequenced in the LPT and SPT orders, respectively. However, the position of the due date within the break has an impact on the structure of the optimal solution. In particular, the number of early and tardy jobs n_e and $(n - n_e)$, respectively, are affected by the relative magnitudes of the expressions involving $B_E = d - B_s$ and $B_T = B_f - d$ in the objective function. Consequently, unlike **UCDD-ADev**, the optimal value of n_e cannot be identified on the fly during the course of the algorithm when the processing times are matched to the objective function coefficients. Ultimately, we characterize the optimal objective function value of **ET–SStB-NonRes** with the additional restriction that exactly n_e jobs are placed before the break. We show that this value $\omega(n_e)$ is discrete convex over $n_e = 0, \ldots, n$, which results in an effective optimal algorithm for **ET–SStB-NonRes**.

The optimal objective function value of **ET–SStB-NonRes** with n_e early jobs is expressed as

$$\omega(n_e) = n_e B_E + (n - n_e)B_T + \sum_{j=1}^{n} c_j(n_e)p_j = n_e B_E + (n - n_e)B_T + f(n_e), \tag{5}$$

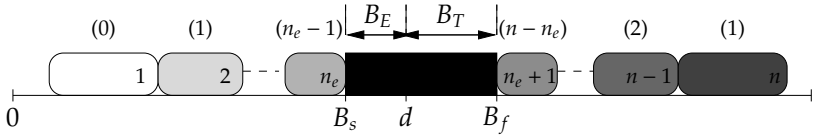


Figure 5 The cost coefficients applied to the processing times are indicated in parentheses above the jobs.

where the entries in the set of objective function coefficients

$$c(n_e) = \{0, 1, \ldots, n_e - 1\} \cup \{1, 2, \ldots, n - n_e\} \tag{6}$$

are sorted in non-decreasing order to minimize the objective function (Baker and Scudder, 1990, Emmons, 1987). The last term $f(n_e)$ in (5) is easily recognized as the optimal objective function value of **UCDD-ADev** with the additional constraint of exactly n_e early/on-time jobs in the schedule. In our problem, this term must be augmented by the first two terms in (5) which denote the *fixed* earliness and tardiness costs resulting from the break. Figure 5 illustrates the objective function coefficients $c(n_e)$.

Lemma 5.1 *The optimal objective function $\omega(n_e)$ defined over $n_e = 0, \ldots, n$ is discrete convex.*

Proof. A function $g : \mathbb{N}_0 \mapsto \mathbb{R}$ is discrete convex if and only if the differences $t \mapsto g(t+1) - g(t)$ are non-decreasing. Therefore, we only need to show that the difference $\omega(n_e + 1) - \omega(n_e)$ is non-decreasing for $n_e = 0, \ldots, n-1$. To this end, we compute

$$\omega(n_e+1) - \omega(n_e) = \begin{cases} B_E - B_T - \sum_{j=2n_e+1}^{n} p_j, & 0 \le n_e < \frac{n}{2}, \\ B_E - B_T, & n_e = \frac{n}{2}, \\ B_E - B_T + \sum_{j=2(n-n_e)+1}^{n} p_j, & \frac{n}{2} < n_e \le n-1. \end{cases} \tag{7}$$

In the interest of space, we only illustrate the calculations for $n_e = \frac{n}{2}$ explicitly, which is only relevant for even n. In this case, we obtain $c(n_e) = c(n_e+1) = \{0, 1, 1, 2, 2, \ldots, n_e - 1, n_e - 1, n_e\}$ from (6). Thus, $\omega(n_e+1) - \omega(n_e) = (n_e+1)B_E - n_eB_E + (n - n_e - 1)B_T - (n - n_e)B_T = B_E - B_T$. The remaining two cases can be derived by constructing $c(n_e)$ and $c(n_e+1)$ in a similar way. In particular, for $0 \le n_e < \frac{n}{2}$ we obtain $\omega(n_e+1) - \omega(n_e) = B_E - B_T + \sigma_{n_e}$, where σ_{n_e} is defined, explained, and computed in Algorithm 2 and the related discussion below.

The term $B_E - B_T$ is common to all three cases in (7). Thus, the proof is completed by arguing that $-\sum_{j=2n_e+1}^{n} p_j$ is strictly negative and strictly increasing for $0 \le n_e < \frac{n}{2}$, and $\sum_{j=2(n-n_e)+1}^{n} p_j$ is strictly positive and strictly increasing for $\frac{n}{2} < n_e \le n-1$. □

A direct consequence of Lemma 5.1 is that $n_e^\star = \min\{n_e \in \mathbb{N}_0 \mid \omega(n_e+1) - \omega(n_e) \ge 0\}$ – the minimizer of $\omega(n_e)$ – can be identified by a standard binary search over the range $0, \ldots, n$, as stated in Algorithm 1, which is invoked with $\underline{n}_e = 0$ for solving **ET–SStB-NonRes**. The argument $\underline{n}_e$ supplied to *Comp_StradB_NonRes_Sched* is for generality and is required in Section 5.2.

The complexity of *Comp_StradB_NonRes_Sched* depends on the function evaluations $f(n_e)$ on line 2 of Algorithm 1. It turns out that all function evaluations $f(n_e)$, $n_e = 0, \ldots, n$, can be performed in $O(n)$ time on line 1 by a careful analysis of the relationship of the set of objective coefficients $c(n_e)$ and $c(n_e+1)$ and the corresponding difference between $f(n_e)$ and $f(n_e+1)$. If all jobs are tardy, then $f(0) = \sum_{j=1}^{n} jp_j$ as calculated in the first `for`-loop in Algorithm 2. The difference $f(n_e+1) - f(n_e)$ is denoted by σ_{n_e}, and we

Algorithm 1: *Comp_StradB_NonRes_Sched*

input : n, $\mathbf{p}$, d, B_s, B_f, $\underline{n}_e$: $\mathbf{p}$ is in LPT order, $\underline{n}_e$ is the minimum number of jobs before the break.
output: $\omega^\star, S^\star, n_e^\star$: optimal objective function value, optimal schedule, number of jobs before the break in $S^\star$.

1 f = *Compute_***UCDD-ADev***_OFV*$(n, \mathbf{p})$;
`// Calculate the values` $f(n_e)$ `for` $n_e = 0, \ldots, n$. `See Algorithm 2.`
2 Search for the minimizer $n_e^\star$ of $\omega(n_e) = n_e B_E + (n - n_e)B_T + f(n_e)$ over $[\underline{n}_e, n]$ via binary search in $O(\log n)$ time;
`//` $\omega(n_e)$ `is discrete convex -- see Lemma 5.1.`
3 Construct the optimal schedule $S^\star$ from $n_e^\star$, $\omega^\star = \omega(n_e^\star)$;

observe that $\sigma_0 = f(1) - f(0) = -\sum_{j=1}^{n} p_j$ – as calculated in the first `for`-loop – because $c(0) = \{1, 2, \ldots, n\}$ and $c(1) = \{0, 1, 2, \ldots, n-1\}$. In general, the effect of increasing the number of early jobs n_e by 1 is to insert the element n_e into and delete the element $n - n_e$ from the set of objective coefficients. That is,

$$\begin{aligned} c(n_e + 1) &= c(n_e) \cup \{n_e\} \setminus \{n - n_e\} \\ &= \{0, 1, 1, 2, 2, \ldots, n_e - 1, n_e - 1, n_e, n_e + 1, \ldots, n - n_e\} \cup \{n_e\} \setminus \{n - n_e\} \quad (8) \\ &= \{0, 1, 1, 2, 2, \ldots, n_e - 1, n_e - 1, n_e, n_e, n_e + 1, \ldots, n - n_e - 1\}. \quad (9) \end{aligned}$$

This presentation assumes that $0 \le n_e \le \frac{n}{2} - 1$ or $0 \le n_e \le \frac{n+1}{2} - 1$ depending on whether n is even or odd, respectively, as taken into account in the bounds of the second `for`-loop in Algorithm 2. The changes in $c(n_e + 1)$ over $c(n_e)$ in (8)-(9) reveal that $\sigma_{n_e} = -\sum_{j=2n_e+1}^{n} p_j$. Similarly, $\sigma_{n_e+1} = -\sum_{j=2(n_e+1)+1}^{n} p_j$, and $\sigma_{n_e+1} - \sigma_{n_e} = p_{2n_e+1} + p_{2n_e+2}$ as employed on line 10 for updating the variable σ. Finally, it is straightforward to figure out that $c(n - n_e) = c(n_e + 1)$ which justifies line 9 and completes the algorithm. Obviously, Algorithm 2 runs in $O(n)$ time and leads to an overall complexity of $O(n + \log n) = O(n)$ for Algorithm 1 without including the cost of sorting the processing times in the LPT order.

Algorithm 2: *Compute_***UCDD-ADev***_OFV*.

input : n, $\mathbf{p}$: $\mathbf{p}$ is in LPT order.
output: $f(n_e)$, $n_e = 0, \ldots, n$: optimal objective function value of **UCDD-ADev** with exactly $n - n_e$ tardy jobs.

1 **if** *n is even* **then** $\bar{n}_e = \frac{n}{2}$ **else** $\bar{n}_e = \frac{n+1}{2}$;
2 $f(0) = 0, \sigma = 0$;
3 **for** $j = 1$ *to* n **do**
4 $\quad \sigma = \sigma - p_j$;
5 $\quad f(0) = f(0) + jp_j$;
6 **end**
7 **for** $k = 0$ *to* $\bar{n}_e - 1$ **do**
8 $\quad f(k+1) = f(k) + \sigma$;
9 $\quad f(n-k) = f(k+1)$;
10 $\quad \sigma = \sigma + p_{2k+1} + p_{2k+2}$;
11 **end**

5.2 Resumable and Semi-resumable Jobs In this variant of our problem with $0 \le \Theta < 1$ – referred to as **ET–SStB-SemiRes**, the jobs are allowed to be interrupted by the straddling break at integer points in time. In this section, we first establish several structural properties of the problem, and then leverage these for our algorithmic design. The key result is a polynomial time algorithm of complexity $O(n^2)$ which computes the optimal solution of **ET–SStB-SemiRes** if $\frac{1}{1-\Theta}$ is integral. Otherwise, a minor modification applied to this algorithm provides the optimal solution at the expense of the pseudo-polynomial complexity $O(n \sum_j p_j)$. In the following development, $n_t = n - n_e$.

Lemma 5.2 *For an instance with a straddling break and for any value of* $0 \le \Theta < 1$, *there exists an optimal schedule without an interrupted job if* $d - B_s \le B_f - d$.

Proof. Suppose to the contrary that there exists an interrupted job – say job j – in the optimal schedule $S^\star$ of an instance with a straddling break, where $d - B_s \le B_f - d$. Let e be the amount of processing job j receives preceding the break before the due date. Consequently, when processing resumes for job j following the break, the length of the remaining processing time is $p_j - e + \lceil e\Theta \rceil$.

If $n_e > n_t$, then $S^\star$ cannot be optimal because the total cost is decreased by delaying the start time of the schedule by e units – that is, by starting job j after the break:

$$n_t \left(p_j - \left(p_j - e + \lceil e\Theta \rceil\right)\right) - n_e e = n_t \left(e - \lceil e\Theta \rceil\right) - n_e e \le \left(n_t - n_e\right) e < 0. \tag{10}$$

The first term in (10) represents the additional cost incurred by the tardy jobs in $S^\star$, including job j. This extra cost is more than offset by the reduction $n_e e$ in the total earliness.

In the complementary case with $n_e \le n_t$, we can construct another schedule with no higher objective function value by shifting the start time of the schedule earlier by $(p_j - e)$ time units. As a result, job j is processed entirely before the break. In the analysis of the change in the total cost in (11), $(d - B_s)$ is the earliness cost incurred by job j after the shift, and $n_e \left(p_j - e\right)$ is the further earliness incurred by the early jobs in $S^\star$. These costs are compensated for by the last two terms on the left hand side of the inequality in (11), which express the reduction in the total tardiness of the tardy jobs in $S^\star$. Note that the tardiness cost of job j in $S^\star$ is $(B_f - d) + (p_j - e + \lceil e\Theta \rceil)$.

$$(d - B_s) + n_e \left(p_j - e\right) - \left(B_f - d\right) - n_t \left(p_j - e + \lceil e\Theta \rceil\right) \le \left(n_e - n_t\right) \left(p_j - e\right) \le 0. \tag{11}$$

□

Lemma 5.2 establishes a simple condition for the dominance of schedules without an interrupted job even when $0 \le \Theta < 1$. This is formalized in Corollary 5.3 and also taken into account in the design of Algorithm 3 which solves **ET–SStB-SemiRes** to optimality.

Corollary 5.3 *If* $d - B_s \le B_f - d$ *then Algorithm 1 solves* ***ET–SStB-SemiRes*** *optimally for* $0 \le \Theta < 1$.

The pillar of the algorithms developed in this section is to fix the location of the interrupted job around the straddling break and then rely on Algorithm 1 to compute the optimal objective function value under this setting. Therefore, Lemma 5.4 presented next helps rule out some jobs from assuming the role of the interrupted job and reduces the computational effort.

Lemma 5.4 *If* $\left\lceil \left(p_j - 1\right)\Theta \right\rceil + 1 = p_j$, *then there exists an optimal schedule, in which job* j *is not interrupted.*

Proof. If job j is interrupted, then the length of its remaining processing time after the break is given by $l_j(s_j, B_s) + (p_j - (B_s - s_j)) = \left\lceil \left(B_s - s_j\right)\Theta \right\rceil + (p_j - (B_s - s_j))$. It is easy to observe that this expression is non-decreasing in s_j for $B_s - p_j + 1 \le s_j \le B_s - 1$ and attains its minimum at $B_s - p_j + 1$ with the corresponding value $\left\lceil \left(p_j - 1\right)\Theta \right\rceil + 1$. If this quantity is equal to p_j and job j is interrupted, the amount of the remaining processing time of job j after the break is then never less than p_j. Consequently, there always exists another feasible schedule with no larger cost in which job j is scheduled completely after the break. □

Next, we turn our attention to the behavior of an interrupted job around a straddling break in an optimal schedule if schedules containing an interrupted job cannot be completely excluded from consideration by Lemma 5.2 and Lemma 5.4. To illustrate the need for this exploration, observe that job j in Figure 1 is always

processed for nine time units after the break if it starts at time $t = B_s - 4$, $B_s - 5$, or $B_s - 6$ and $\Theta = 0.7$. Thus, if this job completes at time $B_f + 9$ in an optimal schedule, then it only makes sense to start it at time $B_s - 4$ in order to avoid introducing additional earliness cost. In order to formalize this observation, we define t_j as the amount of processing performed on job j after the straddling break in an optimal schedule. Then, the corresponding processing time preceding the break is denoted by $e_j^*\left(t_j\right)$ and given in Definition 5.5. In Figure 1, we have $e_j^*\left(t_j\right) = 4$.

Definition 5.5 *If an interrupted job j is processed for $\left\lceil \left(p_j - 1\right)\Theta \right\rceil + 1 \leq t_j \leq p_j - 1$ time units after the break in an optimal schedule with $n_e > 0$, then $e_j^*\left(t_j\right) = \min\left\{e \in \mathbb{N}_0 \mid \left(p_j - e\right) + \lceil e\Theta \rceil = t_j\right\}$.*

The next two technical results describe how $e_j^*\left(t_j\right)$ changes as t_j is varied and are only required for the proofs of Lemma 5.8 and Proposition 5.9. Therefore, the proofs of Lemma 5.6 and Lemma 5.7 are relegated to the appendix.

Lemma 5.6 *The difference between $e_j^*\left(t_j\right)$ and $e_j^*\left(t_j + 1\right)$ equals to either $\left\lceil \frac{1}{1-\Theta} \right\rceil$ or $\left\lfloor \frac{1}{1-\Theta} \right\rfloor$ for $\left\lceil \left(p_j - 1\right)\Theta \right\rceil + 1 \leq t_j < p_j - 1$, and it is equal to $\left\lceil \frac{1}{1-\Theta} \right\rceil$ for $t_j = p_j - 1$.*

Lemma 5.7 *The inequality $e_j^*\left(p_j - i\right) - e_j^*\left(p_j\right) \geq \frac{i}{1-\Theta}$ holds for $1 \leq i \leq p_j - \left(\left\lceil \left(p_j - 1\right)\Theta \right\rceil + 1\right)$.*

Lemma 5.8 *If there is an interrupted job in the optimal schedule, then $\frac{1}{1-\Theta} \leq \frac{n_t}{n_e}$.*

Proof. Suppose to the contrary that $\frac{1}{1-\Theta} > \frac{n_t}{n_e}$ and there is an interrupted job j in the optimal schedule with the completion time $B_f + t_j$. We prove that the cost of this schedule decreases strictly if the entire schedule is shifted to the right by $p_j - t_j$ time units so that job j starts at time B_f. The difference of the extra tardiness cost resulting from the shift and the gain in the earliness cost is computed as:

$$n_t\left(p_j - t_j\right) - n_e\left(e_j^*\left(t_j\right) - e_j^*\left(p_j\right)\right) = n_t\left(p_j - t_j\right) - n_e\left(e_j^*\left(p_j - \left(p_j - t_j\right)\right) - e_j^*\left(p_j\right)\right) \tag{12}$$

$$\leq n_t\left(p_j - t_j\right) - n_e\left(\frac{p_j - t_j}{1-\Theta}\right) = \left(p_j - t_j\right)\left(n_t - n_e\left(\frac{1}{1-\Theta}\right)\right) \tag{13}$$

$$< \left(p_j - t_j\right)\left(n_t - n_e\left(\frac{n_t}{n_e}\right)\right) = 0. \tag{14}$$

The transition from (12) to (13) is due to Lemma 5.7. The strict decrease in the total cost under the assumption that $\frac{1}{1-\Theta} > \frac{n_t}{n_e}$ contradicts the optimality of the original schedule. Therefore, $\frac{1}{1-\Theta}$ must be no larger than $\frac{n_t}{n_e}$ for the presence of an interrupted job in the optimal solution. □

Proposition 5.9 *If job j is interrupted in the optimal schedule and the ratio $\frac{1}{1-\Theta}$ is integral, then there exists an optimal schedule where job j completes at time $B_f + \left\lceil \left(p_j - 1\right)\Theta \right\rceil + 1$. That is, $t_j = \left\lceil \left(p_j - 1\right)\Theta \right\rceil + 1$.*

Proof. If the tardy part of job j in the optimal schedule is $t_j + i$ units where $1 \leq i < p_j - t_j$, then we can shift the entire schedule to the left without increasing the total cost so that the processing of the interrupted job terminates at $B_f + t_j$. The decrease in the total cost associated with this shift is calculated as:

$$n_t\left(t_j + i - t_j\right) - n_e\left(e_j^*\left(t_j\right) - e_j^*\left(t_j + i\right)\right) \tag{15}$$

$$= n_t\left(i\right) - n_e\sum_{l=1}^{i}\left(e_j^*\left(t_j + \left(l-1\right)\right) - e_j^*\left(t_j + l\right)\right) = \sum_{l=1}^{i}\left(n_t - n_e\left(e_j^*\left(t_j + \left(l-1\right)\right) - e_j^*\left(t_j + l\right)\right)\right) \tag{16}$$

$$= \sum_{l=1}^{i}\left(n_t - n_e\left(\frac{1}{1-\Theta}\right)\right) \tag{17}$$

$$\geq \sum_{l=1}^{i} \left(n_t - n_e \frac{n_t}{n_e} \right) = 0. \tag{18}$$

The transition from (16) to (17) stems from Lemma 5.6 and the integrality of $\frac{1}{1-\Theta}$, which yields $\frac{1}{1-\Theta} = \left\lceil \frac{1}{1-\Theta} \right\rceil = \left\lfloor \frac{1}{1-\Theta} \right\rfloor$. The transition from (17) to (18) is due to Lemma 5.8. The schedule obtained via the shift does not lead to a higher cost over the original optimal schedule and must also be optimal. Furthermore, the interrupted job completes at $B_f + t_j$ in this schedule as desired. □

Proposition 5.9 provides us with a fundamental result for designing an effective algorithm for **ET–SStB-SemiRes** if $\frac{1}{1-\Theta}$ is integral by fixing the position of the interrupted job with respect to the break. This allows us to consider the interrupted job as "part of the break." In other words, for each choice of the interrupted job – say job j – we construct an *artificial* break with the start and end points $B_s' = B_s - (p_j - 1)$ and $B_f' = B_f + \left\lceil (p_j - 1)\Theta \right\rceil + 1$, respectively, and call our optimal algorithm *Comp_StradB_NonRes_Sched* for the non-resumable case iteratively. The tardiness cost of job j is then added to the value retrieved from *Comp_StradB_NonRes_Sched* in order to arrive at the optimal cost ω^j given that job j is interrupted. The algorithm *Comp_StradB_SemiRes_Sched* provided in Algorithm 3 solves **ET–SStB-SemiRes** for any value of Θ in $[0, 1)$ such that $\frac{1}{1-\Theta}$ is integral with a complexity of $O(n^2)$ because Algorithm 1 with a complexity of $O(n)$ is invoked at most $n + 1$ times. One significant issue deserves further attention in the design of Algorithm 3. It turns out that the minimizer n_e^{j+1} of $\omega^{j+1}(n_e)$ is no smaller than that of $\omega^j(n_e)$, where $\omega^j(n_e)$ is defined based on (5) with respect to the straddling break running from B_s' to B_f' and the set of $n - 1$ jobs $\{1, \ldots j-1, j+1, \ldots, n\}$. This result – formally proven in Lemma 5.10 – is an important computational enhancement even if it does not affect the theoretical complexity and allows us to restrict the search for n_e^{j+1} to the interval $[n_e^j, n]$. See line 9 in Algorithm 3 below.

Lemma 5.10 *The minimizer n_e^{j+1} of $\omega^{j+1}(n_e)$ is no smaller than that of $\omega^j(n_e)$; i.e., $n_e^{j+1} \geq n_e^j$.*

Proof. By Lemma 5.1, both $\omega^j(n_e)$ and $\omega^{j+1}(n_e)$ are discrete convex. That is, the differences $\omega^j(n_e+1) - \omega^j(n_e)$ and $\omega^{j+1}(n_e + 1) - \omega^{j+1}(n_e)$ are both non-decreasing for $n_e = 0, \ldots, n-2$. Therefore, showing that

$$\omega^j(n_e + 1) - \omega^j(n_e) \geq \omega^{j+1}(n_e + 1) - \omega^{j+1}(n_e), \qquad n_e = 0, \ldots, n-2, \tag{19}$$

holds is sufficient to establish $n_e^{j+1} \geq n_e^j$. Our strategy for the proof is to demonstrate the validity of (19) for each of the three cases in (7). Observe that $\omega^j(n_e)$ and $\omega^{j+1}(n_e)$ are both defined with respect to a total of $n - 1$ jobs, and the indices in the presentation below account for this fact whenever required.

We start by analyzing the common expression $B_E - B_T$ in (7), where we augment the notation B_E and B_T with the index of the interrupted job:

$$\begin{aligned} B_E^j - B_T^j &= d - (B_s - (p_j - 1)) - (B_f + \left\lceil (p_j - 1)\Theta \right\rceil + 1 - d) \\ &= 2d - B_s - B_f + (p_j - \left\lceil (p_j - 1)\Theta \right\rceil) - 2, \text{ and} \\ B_E^{j+1} - B_T^{j+1} &= 2d - B_s - B_f + (p_{j+1} - \left\lceil (p_{j+1} - 1)\Theta \right\rceil) - 2. \end{aligned}$$

Thus, (19) is satisfied for $n_e = \frac{n-1}{2}$ if $B_E^j - B_T^j \geq B_E^{j+1} - B_T^{j+1}$ stands correct. Equivalently, we must argue that $p_j - \left\lceil (p_j - 1)\Theta \right\rceil \geq p_{j+1} - \left\lceil (p_{j+1} - 1)\Theta \right\rceil$. This is accomplished by recalling that $p_j \geq p_{j+1}$ and employing Lemma I.1 in the appendix, which states that $g(p) = p - \lceil (p-1)\Theta \rceil$ is non-decreasing over $p = 1, 2, \ldots$, for $0 \leq \Theta \leq 1$.

In the analysis of the remaining two cases, p_i^j denotes the processing time of the ith job in the LPT sequence if job j is the interrupted job. Obviously, we have $p_j^j = p_{j+1}$, $p_j^{j+1} = p_j$, and $p_i^j = p_i^{j+1}$ for $i = 1, \ldots, j-1, j+1, \ldots, n-1$. Consequently, for $0 \leq n_e < \frac{n-1}{2}$,

$$\omega^j(n_e + 1) - \omega^j(n_e) \geq \omega^{j+1}(n_e + 1) - \omega^{j+1}(n_e)$$

Algorithm 3: *Comp_StradB_SemiRes_Sched*

input : n, $\mathbf{p}$, d, B_s, B_f, Θ: $\mathbf{p}$ is in LPT order.
output: $\omega^\star, S^\star, i^\star, n_e^\star$: optimal objective function value, optimal schedule, index of the interrupted job in $S^\star$, number of jobs before the break in $S^\star$.

```
[ω*, S*, n_e*] = Comp_StradB_NonRes_Sched(n, p, d, B_s, B_f, 0);   // Find the optimal solution with no interrupted job
ω^0 = ω*, S^0 = S*, i* = 0, n_e^0 = n_e*;
if B_f − d < d − B_s then // Else, no need to check schedules with an interrupted job -- see Lemma 5.2
    for j = 1 to n do                                              // Make job j interrupted
        if ⌈(p_j − 1)Θ⌉ + 1 ≤ p_j − 1 then   // Else, no need to make job j interrupted -- see Lemma 5.4
            B'_s = B_s − (p_j − 1);                  // Optimal start time of job j -- see Proposition 5.9
            B'_f = B_f + ⌈(p_j − 1)Θ⌉ + 1 ;   // Optimal completion time of job j -- see Proposition 5.9
            p' = [p_1, ..., p_{j−1}, p_{j+1}, ..., p_n];      // The processing times of all jobs except for job j
            if j = 1 then n_e = 0 else n_e = n_e^{j−1};                    // Based on Lemma 5.10
            // Need to make provisions on the previous line if some jobs are ruled out for
               interruption based on Lemma 5.4
            [ω^j, S^j, n_e^j] = Comp_StradB_NonRes_Sched(n − 1, p', d, B'_s, B'_f, n_e);
            ω^j = ω^j + (B'_f − d) ;                  // Add the tardiness of the interrupted job
            if ω^j < ω* then i* = j, ω* = ω^{i*};      // Keep track of the current best solution
        end
    end
end
S* = S^{i*},   n_e* = n_e^{i*}
```

$$\iff B_E^j - B_T^j - \sum_{i=2n_e+1}^{n-1} p_i^j \geq B_E^{j+1} - B_T^{j+1} - \sum_{i=2n_e+1}^{n-1} p_i^{j+1}$$

$$\iff (B_E^j - B_T^j) - (B_E^{j+1} - B_T^{j+1}) + \sum_{i=2n_e+1}^{n-1} (p_i^{j+1} - p_i^j) \geq 0.$$

$(B_E^j - B_T^j) - (B_E^{j+1} - B_T^{j+1}) \geq 0$ obtained from the previous case and $p_i^{j+1} \geq p_i^j$, $i = 1, \ldots, n-1$, yield the validity of the last inequality. Analogously,

$$\omega^j(n_e+1) - \omega^j(n_e) \geq \omega^{j+1}(n_e+1) - \omega^{j+1}(n_e) \iff (B_E^j - B_T^j) - (B_E^{j+1} - B_T^{j+1}) + \sum_{i=2(n-1-n_e)+1}^{n-1} (p_i^{j+1} - p_i^j) \geq 0$$

is a correct statement for $\frac{n-1}{2} < n_e \leq n-2$. This completes the proof. □

If $\frac{1}{1-\Theta}$ is not integral, then the structural property in Proposition 5.9 is destroyed. Consider the following instance: $n = 19$, $p_j = 29$, for all j, $d = 899$, $B_s = 747$, $B_f = 900$, $\Theta = 0.41$. In any optimal solution, the start time of the interrupted job is 730. This leads to a completion time of $900 + \lceil (747-730)0.41 \rceil + (29 - (747-730)) = 919$. That is, the amount of processing carried out for the interrupted job following the break is 19 time units, which is strictly larger than $\lceil (29-1)0.41 \rceil + 1 = 13$. Our inability to fix the position of the interrupted job around the break precludes us from attaining a polynomial time optimal algorithm for **ET–SStB-SemiRes** with non-integral $\frac{1}{1-\Theta}$ values. However, it is a simple matter to make provisions in Algorithm 3 for the lack of this property. Adding a second (inner) `for`-loop which traverses over all possible completion times $B_f + \lceil (p_j - 1)\Theta \rceil + 1, \ldots, B_f + p_j - 1$ of the interrupted job j – along with the other associated obvious modifications – is sufficient to ensure the optimality of *Comp_StradB_SemiRes_Sched* for **ET–SStB-SemiRes** with any Θ in $[0, 1)$ at the expense of the pseudo-polynomial complexity $O(n \sum_j p_j)$.

6. Non-Straddling Break Earlier in Section 4, we surmised that the position of the break with respect to the due date factors into the complexity of solving **ET–SB**, and the analysis of and the algorithmic design for **ET–SStB-NonRes** and **ET–SStB-SemiRes** in Section 5 attest to the correctness of this claim. Ultimately, **ET–SB** with a single straddling break remains polynomially solvable, unless $\frac{1}{1-\Theta}$ is fractional. However, the story is quite different if the break is not straddling as we delve into in this section. It turns out that **ET–SNStB**, which stands for **ET–SB** with a single non-straddling break, is $\mathcal{NP}-$complete regardless of the value of Θ. The proof is carried out by a reduction from the `EVEN-ODD PARTITION` problem and establishes that **ET–SNStB** is at least weakly $\mathcal{NP}-$hard. As in Section 5, we first investigate the case with non-resumable jobs and devise a pseudo-polynomial time exact algorithm in Section 6.2. We subsequently proceed to **ET–SNStB** with resumable and semi-resumable jobs in Section 6.3 and outline how the pseudo-polynomial time algorithm for non-resumable jobs can be leveraged to solve these cases in much the same spirit that the algorithm *Comp_StradB_SemiRes_Sched* in Section 5.2 invokes *Comp_StradB_NonRes_Sched* as a subroutine. The pseudo-polynomial time complexity remains intact and leads to the conclusion that **ET–SNStB** is not strongly $\mathcal{NP}-$hard. Our results in this section settle the complexity of **ET–SNStB** precisely with no gap for any Θ.

6.1 ET–SNStB is at least Weakly $\mathcal{NP}-$Hard for General Θ The decision version of **ET–SNStB** with $0 \leq \Theta \leq 1$ requires a *yes*/*no* answer to the following question: Does there exist a feasible schedule S with a total cost $f(S)$ no larger than some integer y_0?

The proof borrows constructs and steps from that by Hoogeveen and Van de Velde (1991) for the single-machine *restrictive* common due date total E/T scheduling problem. The similarity between this problem and **ET–SNStB** is best observed if we assume that a long break starts shortly after the due date so that all jobs have to be scheduled in the time interval $[0, B_s]$ in any optimal schedule. In this case, the "restrictiveness" in the tardy part of the schedule – instead of that in the early part – must be accounted for. Nevertheless, a direct reduction from the single-machine restrictive common due date total E/T scheduling problem to **ET–SNStB** remains elusive, and the proof proceeds by a reduction from the `EVEN-ODD PARTITION` problem (Garey et al., 1988) defined as follows: Given a set of $2t$ positive integers $X = \{x_1, x_2, \ldots, x_{2t}\}$ with $x_i > x_{i+1}$ for $1 \leq i < 2t$, does there exist a partition of X into subsets X_1 and X_2 such that $\sum_{x_i \in X_1} x_i = \sum_{x_i \in X_2} x_i = \frac{1}{2}\sum_{x_i \in X} x_i = A$ and X_1 (and hence X_2) contains exactly one of $\{x_{2i-1}, x_{2i}\}$ for each $1 \leq i \leq t$? In the sequel, we prove that `EVEN-ODD PARTITION` has a *yes* answer if and only if the particular instance **I1** of the decision version of **ET–SNStB** described in the following is a *yes*-instance as well. The construction of **I1** is clearly polynomial in the size of the `EVEN-ODD PARTITION` instance.

The instance **I1** includes $2t$ "partition" jobs J_i with $p_i = x_i + tA$ for $i = 1, \ldots, 2t$, in addition to three dummy jobs J_0', J_0'', J_0''' with the processing times $p_0' = p_0'' = p_0''' = 3\left(t^2+1\right)A$, respectively. The common due date is set to $d = 11\left(t^2+1\right)A$, and a single non-straddling break spans the time interval from $B_s = d + \left(t^2+1\right)A = 12\left(t^2+1\right)A$ until $B_f = B_s + y_0$, where $y_0 = \sum_{i=1}^{t} i\,(p_{2i-1} + p_{2i}) + d$ is the upper bound on the total scheduling cost. The value of Θ may be chosen arbitrarily from the interval $[0, 1]$. Observe that the partition jobs J_i, $i = 1, \ldots, 2t$, are in the LPT order, and the common due date d is unrestrictive because $p_0' + p_0'' + p_0''' + \sum_{i=1}^{2t} p_i = 9(t^2+1)A + \sum_{i=1}^{2t}(x_i + tA) = 9(t^2+1)A + 2t^2A + \sum_{i=1}^{2t} x_i = 11(t^2+1)A = d \leq \min\{B_s, d\} = d$ – see the end of Section 2.

Now, consider a partitioning of the set of partition jobs $\{J_1, \ldots, J_{2t}\}$ into two subsets $X_1 = \left\{J_1^{(1)}, \ldots, J_t^{(1)}\right\}$ and $X_2 = \left\{J_1^{(2)}, \ldots, J_t^{(2)}\right\}$ such that both $J_i^{(1)}, J_i^{(2)} \in \{J_{2i-1}, J_{2i}\}$ and $J_i^{(1)} \neq J_i^{(2)}$ for $1 \leq i \leq t$. The processing time associated with $J_i^{(k)}$ is represented by $p_i^{(k)}$ for $k = 1, 2$, and $i = 1, \ldots, t$.

LEMMA 6.1 *If X_1 and X_2 correspond to a solution of* `EVEN-ODD PARTITION`*, then there exists a feasible schedule S_0 for* ***I1*** *with a total E/T cost of at most y_0.*

PROOF. Consider the feasible schedule S_0 illustrated in Figure 6, in which the jobs in X_1 and X_2 are scheduled in LPT and SPT orders, respectively. The cost of the schedule S_0 is:

$$f(S_0) = p_0'' + 2p_0''' + 3\sum_{i=1}^{t} p_i^{(1)} + \sum_{i=1}^{t}(i-1)p_i^{(1)} + \sum_{i=1}^{t} i p_i^{(2)} \tag{20}$$

$$= 12(t^2+1)A + \sum_{i=1}^{t} i\left(p_i^{(1)} + p_i^{(2)}\right) - (t^2+1)A = \sum_{i=1}^{t} i\,(p_{2i-1} + p_{2i}) + d = y_0. \tag{21}$$

In (20), the first three terms stand for the total earliness cost incurred by J_0', J_0'', J_0''', and the last two terms account for the total E/T cost of the jobs in X_1 and X_2. The transition from (20) to (21) follows from the fact that $\sum_{i=1}^{t} p_i^{(1)} = \sum_{x_i \in X_1} x_i + t^2A = \sum_{x_i \in X_2} x_i + t^2A = \sum_{i=1}^{t} p_i^{(2)} = (t^2+1)A$ because X_1 and X_2 correspond to a solution of `EVEN-ODD PARTITION`. (20)-(21) certify that there exists a feasible schedule S_0 for **I1** with a total cost of $f(S_0) \leq y_0$ if X_1 and X_2 constitute a solution for `EVEN-ODD PARTITION`.

□

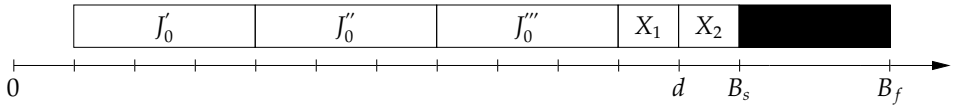


Figure 6 Schedule S_0.

Conversely, suppose that there exists a feasible schedule S for **I1** such that $f(S) \leq y_0$. We next prove that S must have the same structure as S_0, and that the subsets X_1 and X_2 must correspond to a solution of `EVEN-ODD PARTITION`. To this end, we first show a set of useful properties in Lemma 6.2 and then establish that S_0 and S must be identical in Lemma 6.3. The proofs of Lemmas 6.2-6.3 are similar to those of the corresponding results in Hoogeveen and Van de Velde (1991) and therefore presented in the appendix. However, we note that the extra provisions required are much less than straightforward in some cases.

LEMMA 6.2 *The following properties must hold for a feasible schedule S of* ***I1*** *if $f(S) \leq y_0$:*

i. No job completes after the break.
ii. At most t jobs can be started at or after d.
iii. The last job must be completed at time B_s.
iv. The dummy jobs J_0', J_0'', and J_0''' must be scheduled at the first three positions.
v. At least $t-1$ partition jobs must be started at or after d.

LEMMA 6.3 *If there exists a feasible schedule S of* ***I1*** *with $f(S) \leq y_0$, then the underlying instance of the* `EVEN-ODD PARTITION` *problem is a yes-instance.*

THEOREM 6.4 *The decision version of* ***ET–SNStB*** *is $\mathcal{NP}$–complete in the ordinary sense for $0 \leq \Theta \leq 1$.*

PROOF. The decision version of **ET–SNStB** is clearly in $\mathcal{NP}$. Furthermore, the construction of **I1** is polynomial in the size of the underlying `EVEN-ODD PARTITION` instance, and Lemma 6.1 assures that **I1** is a *yes*-instance of **ET–SNStB** if the associated `EVEN-ODD PARTITION` instance is a *yes*-instance. The converse follows from Lemma 6.3. These two lemmas complete the polynomial transformation from `EVEN-ODD PARTITION` to **ET–SNStB** and yield the desired result because `EVEN-ODD PARTITION` is $\mathcal{NP}$–complete in the ordinary sense.

□

6.2 Non-resumable Jobs In this section, we devise a pseudo-polynomial time exact dynamic programming (DP) algorithm for the special case of **ET–SNStB** with non-resumable jobs – referred to as **ET–SNStB-NonRes**. As evident from Figure 3a and the related discussion in Section 4, inserted idle time may be a must in the optimal schedule. For clarity, two separate DP recursions are developed, depending on whether the single non-straddling break completes prior to the commencement of the break such that $B_f < d$ or starts after the due date with $B_s > d$. However, we underscore that both cases bear structural similarities. In particular, the underlying pillar of both DP algorithms is the *weakly V-shaped* property. The essence of this property is that a string of jobs in LPT order is followed by a string of jobs in SPT order in any optimal schedule that minimizes the total E/T around a restrictive common due date (Hall et al., 1991). That is, the processing time of the straddling job – if it exists – is either no longer than that of the final job that completes prior to the due date or no longer than that of the first job that starts after the due date. Thus, if the jobs are indexed in the SPT order – as assumed in the following presentation – and inserted into the schedule one by one, the next longest job is either appended to the head or the tail of the job processing sequence. The start time of the first job along with the job processing sequence is then sufficient to describe a schedule. This is exploited in an exact pseudo-polynomial DP algorithm by Ventura and Weng (1995) for solving the total E/T problem with a restrictive common due date. Obviously, the weakly V-shaped property must also hold for any optimal schedule of **ET–SNStB-NonRes** following or preceding the break, depending on whether $B_f < d$ or $B_s > d$, respectively. Therefore, our optimal DP algorithm for **ET–SNStB-NonRes** follows suit with that in Ventura and Weng (1995), except that it also accounts for the possibility that jobs can be assigned for processing in the LPT order before the break if $B_f < d$ or in the SPT order upon the completion of the break if $B_s > d$.

If $B_f < d$, idle time between two consecutive jobs may only be present in an optimal schedule between B_f and the start time of the first job performed following the break. In the forward DP recursion below, $h_k(s,e)$ stands for the minimum cost of scheduling jobs $1,2,\ldots,k$ given that the first job after the break starts at time s and the total amount of processing before the break is e time units. The recursive relation for stage $k \geq 1$ considers different cases defined by the relative positions of s and d and the length of the processing sequence starting at time s:

$$h_k(s,e) = \begin{cases} \min\left\{h_k'(s,e), h_k''(s,e), h_k'''(s,e)\right\}, & \text{if } B_f \leq s < d,\ s + \sum_{i=1}^{k} p_i - e > d, \text{ and } e \geq 0, \\ \min\left\{h_k'(s,e), h_k'''(s,e)\right\}, & \text{if } B_f \leq s < d,\ s + \sum_{i=1}^{k} p_i - e \leq d, \text{ and } e \geq 0, \\ \min\left\{h_k''(s,e), h_k'''(s,e)\right\}, & \text{if } d \leq s \leq d + p_n - 1 \text{ and } e \geq 0, \\ \infty, & \text{otherwise,} \end{cases} \tag{22}$$

where $h_k'(s,e) = \left|d - (s + p_k)\right| + h_{k-1}(s + p_k, e)$ captures the optimal total cost of scheduling jobs $1,2,\ldots,k$ given that job k starts at time s. This case is omitted from consideration if $s \geq d$ because it would violate the SPT order for the tardy jobs. The second option is to schedule job k at the very end of the processing sequence with a completion time of $\left(s + \sum_{i=1}^{k} p_i - e\right)$, and the associated optimal cost of scheduling jobs $1,2,\ldots,k$ is then computed as $h_k''(s,e) = \left(s + \sum_{i=1}^{k} p_i - e\right) - d + h_{k-1}(s,e)$. This case is only relevant if job k terminates strictly after the due date; otherwise, the LPT order would not be observed by the jobs executed between B_f and d. Finally, job k may also be performed in the initial position prior to the break to complete at $(B_s - (e - p_k))$. The resulting optimal cost for jobs $1,2,\ldots,k$ is provided by $h_k'''(s,e) = d - (B_s - (e - p_k)) + h_{k-1}(s, e - p_k)$.

The boundary conditions for the recursion in (22) are defined as:

$$h_0(s,e) = \begin{cases} 0, & \text{for } e = 0 \text{ and } B_f \le s \le d + p_n - 1, \\ \infty, & \text{otherwise,} \end{cases} \tag{23}$$

and the optimal cost is given by $h_n(s^*, e^*) = \min_{B_f \le s \le d, 0 \le e \le P} h_n(s,e)$, where $P = \sum_{i=1}^{n} p_i$.

Note that the value of the first state variable is increased to or beyond d at some stage k only if $s < d$ currently holds, job k is started at time s, and $s + p_k \ge d$. In this case, no job k' is scheduled at the time instant denoted by the value of the first state variable at any stage $1 \le k' < k$ in order to conform with the SPT ordering of the jobs following the due date – as explained above. These observations lead us to conclude that $s \le d - 1 + p_n$ is fulfilled at any stage $1 \le k \le n$ as asserted on the third line of (22) and in (23), and that the total number of states is $O(nP(d + p_n - B_f))$. Each state is evaluated in constant time, and thus, if $B_f < d$ **ET–SNStB-NonRes** is solved to optimality by our DP algorithm with an overall pseudo-polynomial time complexity of $O(nP(d + p_n - B_f))$.

If $B_s > d$, unforced idle time between two consecutive jobs may only exist in an optimal schedule upon the completion of the final job before the break and B_s. In the forward DP recursion below, $h_k(c,t)$ represents the minimum cost of scheduling jobs $1, 2, \ldots, k$ given that the final job before the break terminates at time c and the total amount of processing after the break is t time units. As with the previous DP, the recursive relation for stage $k \ge 1$ considers different cases defined by the relative positions of c and d and the length of the processing sequence ending at time c:

$$h_k(c,t) = \begin{cases} \min\left\{h_k'(c,t), h_k''(c,t), h_k'''(c,t)\right\}, & \text{if } d < c \le B_s,\ c - \sum_{i=1}^{k} p_i + t < d, \text{ and } t \ge 0, \\ \min\left\{h_k'(c,t), h_k'''(c,t)\right\}, & \text{if } d < c \le B_s,\ c - \sum_{i=1}^{k} p_i + t \ge d, \text{ and } t \ge 0, \\ \min\left\{h_k''(c,t), h_k'''(c,t)\right\}, & \text{if } d - p_n + 1 \le c \le d \text{ and } t \ge 0, \\ \infty, & \text{otherwise,} \end{cases} \tag{24}$$

where $h_k'(c,t) = (c - d) + h_{k-1}(c - p_k, t)$ is the optimal total cost of scheduling jobs $1, 2, \ldots, k$ if job k is appended to the end of the string of jobs before the break and finishes at time c. This case is ignored if $c \le d$ in order to avoid breaking the LPT order for the early and on-time jobs. Otherwise, job k may be also be performed at the very start of the schedule and completes processing at time $c - \left(\sum_{i=1}^{k-1} p_i - t\right)$, which corresponds to $h_k''(c,t) = d - \left(c - \left(\sum_{i=1}^{k-1} p_i - t\right)\right) + h_{k-1}(c,t)$ as the associated optimal cost of scheduling jobs $1, 2, \ldots, k$. This case is disregarded if the start time $\left(c - \left(\sum_{i=1}^{k} p_i - t\right)\right)$ of job k is not prior to the due date; otherwise, the SPT order would be violated by the jobs executed between d and B_s. Finally, $h_k'''(c,t) = \left(B_f + t\right) - d + h_{k-1}(c, t - p_k)$ reflects the optimal cost of scheduling jobs $1, 2, \ldots, k$ if job k is put to the last position after the break.

The boundary conditions for the recursion in (24) are given as:

$$h_0(c,t) = \begin{cases} 0, & \text{for } t = 0 \text{ and } d - p_n + 1 \le c \le B_s, \\ \infty, & \text{otherwise,} \end{cases} \tag{25}$$

and $h_n(c^*, t^*) = \min_{d \le c \le B_s, 0 \le t \le P} h_n(c,t)$ provides us with the optimal cost of scheduling all jobs $1, \ldots, n$.

By a similar reasoning to that in the case with $B_f < d$, we determine that the number of states in the DP recursion is $O(nP(B_s - d + p_n))$, where each state is evaluated in constant time. This lends an overall pseudo-polynomial time complexity of $O(nP(B_s - d + p_n))$ to solving **ET–SNStB-NonRes** exactly if $B_s > d$.

6.3 Resumable and Semi-resumable Jobs For this variant of our problem with $0 \le \Theta < 1$ – referred to as **ET–SNStB-SemiRes**, we do not expect to be able to devise a polynomial time algorithm due to Theorem

6.4. However, we can rely on a strategy similar to that we applied for non-integral $\frac{1}{1-\Theta}$ at the end of Section 5.2 to design an exact algorithm. That is, we create an *artificial* break for every possible completion time of a candidate interrupted job and then invoke one of the algorithms in Section 5.2 or Section 6.2 to calculate the optimal cost for the remaining $n-1$ jobs, depending on the relative location of the due date with respect to the artificial break and the value of Θ. Since the complexity of the DP algorithms in Section 6.2 are pseudo-polynomial, we do not expect a better overall worst-case complexity; however, a result similar to that in Proposition 5.9 would prove useful from a computational point of view by reducing the set of possible completion times for a candidate interrupted job to a singleton. Unfortunately, such a result remains out of our reach even when $\frac{1}{1-\Theta}$ is integral, and the structure of the optimal solution may be different than that prescribed in Proposition 5.9 if the break is not straddling. This is illustrated in the example of Figure 7, where the interrupted job terminates at $B_f + 2$ in all six symmetric optimal solutions of this instance. If the break were straddling, we would be assured of the existence of an optimal solution with a completion time of $B_f + 1$ for the interrupted job.

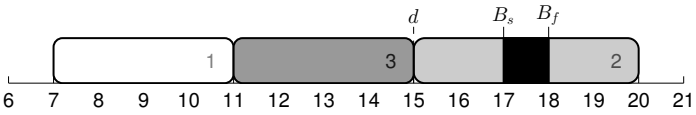


Figure 7 The interrupted job completes at time $B_f + 2$ in all optimal solutions ($\Theta = 0$).

The proposed algorithm for solving **ET–SNStB-SemiRes** traverses over all possible completion times $B_f + \left\lceil \left(p_j - 1\right)\Theta \right\rceil + 1, \ldots, B_f + p_j - 1$ of a candidate interrupted job j. If job j receives t_j units of processing following B_f, the corresponding artificial break runs from $B_s - e_j^*\left(t_j\right)$ to $B_f + t_j$. We then invoke one of the algorithms in Section 5.2 appropriate for the value of Θ with $n-1$ jobs by excluding the interrupted job if the artificial break happens to contain d. Otherwise, if d falls outside the artificial break, we rely on one of DP recursions in Section 6.2 depending on whether $B_f + t_j < d$ or $B_s - e_j^*\left(t_j\right) > d$. In all cases, the E/T cost of the interrupted job is added to the objective function value with $n-1$ jobs retrieved from the subroutine to arrive at the correct objective function value for all n jobs, given a candidate interrupted job and an associated fixed completion time. The minimum cost over $O(P)$ iterations – one iteration for each possible position of a candidate interrupted job $j = 1, \ldots, n$ – yields the minimum cost of **ET–SNStB-SemiRes** under the assumption that there exists an interrupted job. In order to identify the optimal schedule of **ET–SNStB-SemiRes**, this figure is then compared to the cost of the optimal schedule with no interrupted job provided by one of the DP algorithms in Section 6.2 for the entire set of jobs and the original break. The overall time complexity of solving **ET–SNStB-SemiRes** exactly is pseudo-polynomial because we either call a polynomial or pseudo-polynomial time algorithm for a total of $O(P)$ iterations.

7. Conclusions & Future Research In this paper, we offer a rigorous analysis of the single-machine total earliness/tardiness scheduling problem around an unrestrictive common due date with machine unavailability constraints. We cover a wide range of problem variants differentiated by the number of breaks, the job resumability scheme, and the position of the due date with respect to the break if there is just a single break in the planning horizon. In all cases, we derive structural properties and draw the line between polynomially solvable and $\mathcal{NP}$–complete variants. The analysis of the variants characterized as $\mathcal{NP}$–complete in the ordinary sense is complemented by exact algorithms of pseudo-polynomial complexity, leaving no ambiguity in their complexity status. In particular, this research establishes that no problem variant with a single break is

$\mathcal{NP}-$complete in the strong sense.

Certain open research questions remain. The proof of Theorem 3.4 is restricted to $0 < \theta \leq 1$, and the complexity of **ET–MB** still needs to be settled for resumable jobs. Moreover, for the case of a single straddling break with semi-resumable jobs and a non-integral value for $\frac{1}{1-\Theta}$, we have a pseudo-polynomial algorithm. We conjecture that this variant is $\mathcal{NP}-$complete in the ordinary sense, but have no formal proof at this point.

A primary strategy of the paper has been to initially gain insights into the nature of the problem variants with non-resumable jobs and then leverage these for corresponding variants with resumable and semi-resumable jobs. For non-resumable jobs, the problem complexity goes from being polynomial in the presence of a single straddling break to $\mathcal{NP}-$complete in the ordinary sense if the break is not straddling, and finally to $\mathcal{NP}-$complete in the strong sense if there are several breaks. This elegant sequence of results justifies solving **ET–MB** via an integer programming formulation – as is done in Section 2. However, a worthy future research goal is to develop a custom, fast, and scalable exact algorithm for **ET–MB** with $\Theta = 1$ that can possibly be further modified for or called upon as a subroutine in an exact approach to solve **ET–MB** with resumable and semi-resumable jobs.

Other possible extensions of this paper include considering different penalty schemes for the job interruptions. For instance, the parameter Θ may be job-dependent, or each interruption may be followed by a job-dependent fixed setup time independent of the amount of processing already received before the break (Graves and Lee, 1999). Another research question is whether the structural properties, complexity results, and algorithms presented in this paper can be generalized if the unit E/T penalties are job-dependent.

Acknowledgment. We thank the anonymous referees and the associate editor for their comments which helped us improve the paper. The second author has been partially supported by a Google Award. The third author has been supported by Fondation Mathématique Jacques Hadamard under the Gaspard Monge Program for optimisation and operations research.

Appendix H. A Time-Indexed Formulation for the Single-Machine E/T Scheduling Problem with Machine Availability Restrictions and Distinct Due Dates In the time-indexed formulation corresponding to the general problem statement given in Section 2, period t covers the time interval $[t, t+1)$. The set of all time periods during which the machine is available in the planning horizon is represented by $\mathcal{T}$, and the set $\mathcal{T}_i$ is defined such that job i must start in one of the time periods in $\mathcal{T}_i$ so that it finishes no later than past the end of the planning horizon T. The time-indexed formulation **(TI)** of the general problem is then stated as:

$$\textbf{(TI)} \qquad \text{minimize} \quad \sum_{i=1}^{n} \sum_{t \in \mathcal{T}_i} c_{it} x_{it} \tag{26}$$

$$\text{subject to} \quad \sum_{t \in \mathcal{T}_i} x_{it} = 1, \qquad i = 1, \ldots, n, \tag{27}$$

$$\sum_{i=1}^{n} \sum_{s \in S_i(t)} x_{is} \leq 1, \qquad t \in \mathcal{T}, \tag{28}$$

$$x_{it} \in \{0, 1\}, \qquad i = 1, \ldots, n,\ t \in \mathcal{T}_i, \tag{29}$$

where the binary variable x_{it} is set to 1, if job i starts its processing at time t with an associated cost of $c_{it} = \alpha_i(d_i - (t + \bar{p}_i(t)))^+ + \beta_i(t + \bar{p}_i(t) - d_i)^+$. The notation $(z)^+$ stands for $\max(z, 0)$. The constraints (27) mandate that each job starts its processing exactly once in the planning horizon, and the machine capacity constraints (28) prescribe that no more than one job is active at any time instant in the planning horizon. In the modeling of the capacity constraints, the set $S_i(t) = \{s_i |\ s_i \leq t \text{ and } s_i + \bar{p}_i(s_i) \geq t + 1\}$ denotes the set of all possible start

times of job i so that job i is in process in time period t. Thus, $\sum_{i=1}^{n}\sum_{s\in S_i(t)} x_{is}$ yields the total number of jobs being executed on the machine at time t, and the capacity is imposed by restricting this expression not to exceed 1 in any time period. For any given instance, all sets $\mathcal{T}_i$, $i = 1,\ldots,n$, and $S_i(t)$, $i = 1,\ldots,n, t \in \mathcal{T}$, can be constructed from the nominal processing times, the locations of the breaks in the planning horizon, and the value of Θ as part of the data to be fed into the formulation **(TI)**.

Appendix I. Technical Results for Section 5.2

I.1 Proof of Lemma 5.6

Proof. For $\lceil (p_j-1)\Theta \rceil + 1 \le t_j < p_j - 1$, the difference is computed as follows:

$$\begin{aligned}
e_j^*(t_j) - e_j^*(t_j+1) &= \min\{e\in\mathbb{N}_0 \mid p_j - e + \lceil e\Theta\rceil = t_j\} - \min\{e\in\mathbb{N}_0 \mid p_j - e + \lceil e\Theta\rceil = t_j+1\} \\
&= \min\{e\in\mathbb{N}_0 \mid \lceil e(\Theta-1)\rceil = t_j - p_j\} - \min\{e\in\mathbb{N}_0 \mid \lceil e(\Theta-1)\rceil = t_j - p_j + 1\} \\
&= \min\{e\in\mathbb{N}_0 \mid \lfloor e(1-\Theta)\rfloor = p_j - t_j\} - \min\{e\in\mathbb{N}_0 \mid \lfloor e(1-\Theta)\rfloor = p_j - t_j - 1\} \\
&= \min\{e\in\mathbb{N}_0 \mid e(1-\Theta) \ge p_j - t_j\} - \min\{e\in\mathbb{N}_0 \mid e(1-\Theta) \ge p_j - t_j - 1\} \\
&= \left\lceil \frac{p_j - t_j}{1-\Theta} \right\rceil - \left\lceil \frac{p_j - t_j - 1}{1-\Theta} \right\rceil.
\end{aligned}$$

Note that $\left\lfloor \frac{1}{1-\Theta} \right\rfloor \le \left\lceil \frac{p_j-t_j}{1-\Theta} \right\rceil - \left\lceil \frac{p_j-t_j-1}{1-\Theta} \right\rceil \le \left\lceil \frac{1}{1-\Theta} \right\rceil$ and $0 \le \left\lceil \frac{1}{1-\Theta} \right\rceil - \left\lfloor \frac{1}{1-\Theta} \right\rfloor \le 1$. Therefore, $e_j^*(t_j) - e_j^*(t_j+1)$ equals to either $\left\lceil \frac{1}{1-\Theta} \right\rceil$ or $\left\lfloor \frac{1}{1-\Theta} \right\rfloor$ if $\lceil (p_j-1)\Theta \rceil + 1 \le t_j < p_j - 1$.

The analysis is similar for $t_j = p_j - 1$:

$$\begin{aligned}
e_j^*(p_j-1) - e_j^*(p_j) &= \min\{e\in\mathbb{N}_0 \mid p_j - e + \lceil e\Theta\rceil = p_j - 1\} - \min\{e\in\mathbb{N}_0 \mid p_j - e + \lceil e\Theta\rceil = p_j\} \\
&= \min\{e\in\mathbb{N}_0 \mid \lceil e(\Theta-1)\rceil = -1\} - 0 = \min\{e\in\mathbb{N}_0 \mid \lfloor e(1-\Theta)\rfloor = 1\} \\
&= \min\{e\in\mathbb{N}_0 \mid e(1-\Theta) \ge 1\} = \left\lceil \frac{1}{1-\Theta} \right\rceil.
\end{aligned}$$

□

I.2 Proof of Lemma 5.7

Proof.

$$\begin{aligned}
e_j^*(p_j-i) - e_j^*(p_j) &= \min\{e\in\mathbb{N}_0 \mid p_j - e + \lceil e\Theta\rceil = p_j - i\} - \min\{e\in\mathbb{N}_0 \mid p_j - e + \lceil e\Theta\rceil = p_j\} \\
&= \min\{e\in\mathbb{N}_0 \mid \lceil e(\Theta-1)\rceil = -i\} - 0 = \min\{e\in\mathbb{N}_0 \mid \lfloor e(1-\Theta)\rfloor = i\} \\
&= \min\{e\in\mathbb{N}_0 \mid e(1-\Theta) \ge i\} = \left\lceil \frac{i}{1-\Theta} \right\rceil \ge \frac{i}{1-\Theta}.
\end{aligned}$$

□

I.3 Result Required for Lemma 5.10

Lemma I.1 *The function $g(p) = p - \lceil (p-1)\Theta \rceil$ defined over $p = 1, 2, \ldots$, is non-decreasing for $0 \le \Theta \le 1$.*

Proof. The result holds trivially for $\Theta = 0, 1$, and we restrict our attention to $0 < \Theta < 1$. Obviously, $g(p) = p - \lceil (p-1)\Theta \rceil \le p - (p-1)\Theta = p(1-\Theta) + \Theta$. Then,

$$(p(1-\Theta) + \Theta - g(p)) - ((p+1)(1-\Theta) + \Theta - g(p+1)) \tag{30}$$

$$= (p(1-\Theta) + \Theta - p + \lceil (p-1)\Theta \rceil) - ((p+1)(1-\Theta) + \Theta - p - 1 + \lceil p\Theta \rceil) \tag{31}$$

$$= \Theta + \lceil p\Theta - \Theta \rceil - \lceil p\Theta \rceil. \tag{32}$$

Since $0 < \Theta < 1$, the difference $\lceil p\Theta - \Theta \rceil - \lceil p\Theta \rceil$ is either 0 or -1, and accordingly, the right hand side of (32) is either Θ or $\Theta - 1$. We analyze these two cases separately.

Re-arranging the terms in (30) yields $g(p+1) - g(p) - (1-\Theta)$, and this expression is either equal to Θ or $\Theta - 1$. In the former case, $g(p+1) - g(p) = 1$, while the latter case results in $g(p+1) - g(p) = 0$. Thus, we conclude that $g(p)$ is non-decreasing over $p = 1, 2, \ldots$.

□

Appendix J. Technical Results for Section 6.1

J.1 Proof of Lemma 6.2

Proof.

i. The tardiness of any job that completes after the break is no less than $B_f - d = d + (t^2+1)A + y_0 - d = (t^2+1)A + y_0 > y_0$. Thus, all jobs must terminate before B_s if $f(S) \leq y_0$.

ii. The dummy jobs are longer than all partition jobs because the sum of the processing times of the longest t partition jobs are $\sum_{i=1}^{t} p_i = \sum_{i=1}^{t} x_i + t^2 A < (t^2+2)A < 3(t^2+1)A = p'_0 = p''_0 = p'''_0$, where the first strict inequality stems from $\sum_{i=1}^{t} x_i < \sum_{i=1}^{2t} x_i = 2A$. There is no processing following the break due to item i, and therefore, we can complete the argument by showing that the total processing time of the shortest t partition jobs does not exceed $(B_s - d)$ and that the next shortest partition job does not fit into the time interval between d and B_s. The relation $\sum_{i=t+1}^{2t} x_i < \frac{1}{2}\sum_{i=1}^{2t} x_i = A$ must hold for the t smallest numbers in X, and the first part is then obtained from $\sum_{i=t+1}^{2t} p_i = \sum_{i=t+1}^{2t} x_i + t^2 A < (t^2+1)A = B_s - d$. Moreover, it turns out that $\sum_{i=t}^{2t} p_i = \sum_{i=t}^{2t} x_i + (t^2+t)A > (t^2+1)A = B_s - d$, and no additional job can be scheduled during $[d, B_s]$ because job t is the shortest among the remaining jobs $J_1, \ldots, J_t, J'_0, J''_0, J'''_0$.

iii. Item ii directly implies that there are at most $t+1$ tardy or on-time jobs, and consequently, at least $t+2$ early jobs in S if $f(S) \leq y_0$. Therefore, pushing the entire schedule later by the amount of idle time between the completion time of the final job in the schedule and B_s is guaranteed to lead to a strict decrease in $f(S)$. Without loss of generality, we can assume that a schedule S with a total cost of no more than y_0 possesses this property.

iv. S is charged a cost of $f(S) \leq y_0$, and without loss of generality, also observes the V-shaped property. Furthermore, note that the time available for processing between d and B_s is $(t^2+1)A < 3(t^2+1)A = p'_0 = p''_0 = p'''_0$ units, and suppose that at least one of J'_0, J''_0, or J'''_0 does not fill one of the first three positions in S. Then, the only remaining possible structure for S is that exactly two of the three dummy jobs are put into the initial two positions, while the third dummy job is straddling – say J'''_0. Moreover, $p'''_0 - (B_s - d) = 3(t^2+1)A - (t^2+1)A = 2(t^2+1)A$ is the minimum amount of processing J'''_0 receives prior to the due date. The partition jobs are distributed around J'''_0 with at most t starting after J'''_0 due to item ii and the final job terminating at time B_s as mandated by item iii.

In the development below, we construct a lower bound on the total cost of S with exactly k early partition jobs – denoted as $\underline{f}(S,k)$ – by assuming that the k early and $(2t-k)$ tardy partition jobs are selected from the k shortest and $(2t-k)$ shortest jobs, respectively. The relevant range of k is from t to $2t$ based on item ii. We obtain

$$\underline{f}(S,t) = \underbrace{5(t^2+1)A + \sum_{i=t+1}^{2t} p_i}_{\leq \text{ cost of } J'_0} + \underbrace{2(t^2+1)A + \sum_{i=t+1}^{2t} p_i}_{\leq \text{ cost of } J''_0} + \underbrace{2t(t^2+1)A + \sum_{i=t+1}^{2t} (i-t-1)p_i}_{\leq \text{ cost of early partition jobs}} + \underbrace{\sum_{i=t+1}^{2t} (i-t)p_i}_{\leq \text{ cost of tardy partition jobs}}$$

$$= (2t+7)(t^2+1)A + \sum_{i=t+1}^{2t}(i-t+1)p_i + \sum_{i=t+1}^{2t}(i-t)p_i = (2t+7)(t^2+1)A + \sum_{i=1}^{t}(2i+1)p_{i+t},$$

and a similar reasoning leads to

$$\underline{f}(S,t+1) = (2t+9)(t^2+1)A + \sum_{i=t}^{2t}(i-t+2)p_i + \sum_{i=t+2}^{2t}(i-t-1)p_i,$$

$$\underline{f}(S,t+2) = (2t+11)(t^2+1)A + \sum_{i=t-1}^{2t}(i-t+3)p_i + \sum_{i=t+3}^{2t}(i-t-2)p_i.$$

The pattern is clear, and it is a simple matter to verify that the coefficients associated with the processing times never decrease from $\underline{f}(S,k)$ to $\underline{f}(S,k+1)$ for $k = t,\ldots,2t-1$. This analysis of the relationships among $\underline{f}(S,t), \underline{f}(S,t+1)\ldots,\underline{f}(S,2t)$ reveals that $\underline{f}(S,k)$ is non-decreasing over $k = t,\ldots,2t$, and we arrive at the inequality $f(S) \geq \min_{k=t,\ldots,2t}\underline{f}(S,k) = \underline{f}(S,t)$. Thus, the only remaining piece for a contradiction in the proof is to establish that $\underline{f}(S,t) > y_0$. To this end, we compute

$$y_0 = \sum_{i=1}^{t} i\,(p_{2i-1}+p_{2i}) + d \tag{33}$$

$$\leq \frac{1}{2}(t+1)\sum_{i=1}^{2t} p_i + d = \frac{1}{2}(t+1)\,2\left(t^2+1\right)A + 11\left(t^2+1\right)A \tag{34}$$

$$= (t+12)\left(t^2+1\right)A = \left(t^3+12t^2+t+12\right)A.$$

The transition from (33) to (34) is justified because $\frac{1}{2t}\sum_{i=1}^{t} 2i = \frac{1}{2}(t+1)$. In other words, (33) matches the longest processing times with the smallest coefficients while the same "average coefficient" is applied to all processing times in (34), and the earlier expression is therefore a lower bound on the latter. Note that the equality in (34) stems from $\sum_{i=1}^{2t} p_i = \sum_{i=1}^{2t}(x_i + tA) = 2A + 2t^2A = 2(t^2+1)A$. However,

$$f(S) \geq \underline{f}(S,t) = (2t+7)(t^2+1)A + \sum_{i=1}^{t}(2i+1)p_{i+t}$$

$$> (2t+7)(t^2+1)A + \sum_{i=1}^{t}(2i+1)tA$$

$$= (3t^3+9t^2+2t+7)A \tag{35}$$

$$\geq \left(t^3+12t^2+t+12\right)A \geq y_0, \tag{36}$$

and contradicts our initial assumption that $f(S) \leq y_0$. Unfortunately, this completes the proof just for $t \geq 2$ because the relation between (35) and (36) is violated for $t = 1$, and this case must be treated separately.

For $t = 1$, we obtain $d = 11(t^2+1)A = 22A$, $y_0 = \sum_{i=1}^{t} i\,(p_{2i-1}+p_{2i}) + d = 2(t^2+1)A + d = 26A$, $p'_0, p''_0, p'''_0 = 3(t^2+1)A = 6A$, $B_s - d = (t^2+1)A = 2A$. In this case, S must assume one of the three possible structures illustrated in Figure 8. In Figure 8a, $f(S) = 14A + 8A + (p_2 + 4A) + 4A + 2A = 32A + p_2 > 26A = y_0$. In Figure 8b, $f(S) > 13A + 7A + 6A + 2A = 28A > y_0$, where the earliness of J'''_0 is strictly smaller than A and cannot be incorporated. Finally, in Figure 8c, $f(S) > 13A + 7A + 5A + 2A = 27A > y_0$, where the tardiness of J'''_0 is strictly smaller than A and ignored. In all cases, we obtain a contradiction with our initial assumption that $f(S) \leq y_0$.

v. It is sufficient to verify that the time interval $[d, B_s]$ is long enough to accommodate the processing of the longest $t-1$ partition jobs because items iii and iv together mandate that all partition jobs are placed

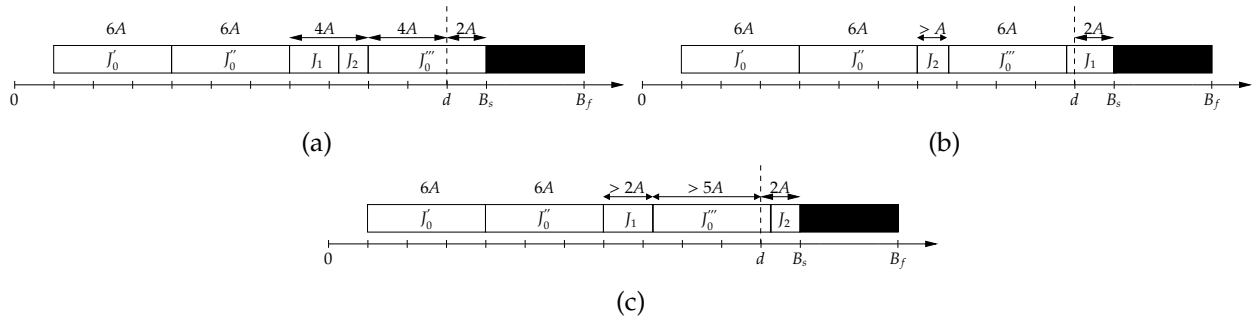


Figure 8 The possible structures of schedule S for $t = 1$.

following the dummy jobs and that there is no idle time between the completion of the final job and the start time of the break. The sequence of relations $\sum_{i=1}^{t-1} p_i = \sum_{i=1}^{t-1} x_i + (t-1)tA < 2A + (t^2-t)A = (t^2-t+2)A \leq (t^2+1)A = B_s - d$ yields the desired result, where the first strict inequality is implied by $\sum_{i=1}^{t-1} x_i < \sum_{i=1}^{2t} x_i = 2A$ for $t \geq 1$.

□

J.2 Proof of Lemma 6.3

Proof. Schedule S fulfills the properties put forward in Lemma 6.2 because $f(S) \leq y_0$. By Lemma 6.2-iv, the dummy jobs J_0', J_0'', J_0''' are executed consecutively at the start of S, followed by $2t$ partition jobs. In the presentation below, $s(i)$ denotes the index of the job that is scheduled in position i from the end in S, and C_i stands for the completion time of job i. We initially compute the total cost $f_k(S)$ incurred in S with respect to a hypothetical due date $k = B_s - \left(p_{s(1)} + \cdots + p_{s(t)}\right)$, which coincides with $C_{s(t+1)}$. Then, we point out the relationship between $f_k(S)$ and the true cost $f(S)$ of S with respect to the actual due date d. This relationship is exploited to prove the final result. The cost $f_k(S)$ is computed as:

$$
\begin{aligned}
f_k(S) &= \sum_{i=1}^{2t} |C_i - k| + \left(k - C_0'\right) + \left(k - C_0''\right) + \left(k - C_0'''\right) \\
&= \underbrace{\sum_{i=1}^{t} (i) p_{s(i)}}_{\substack{\text{cost of tardy} \\ \text{partition jobs}}} + \underbrace{\sum_{i=t+1}^{2t} (2t-i) p_{s(i)}}_{\substack{\text{cost of early} \\ \text{partition jobs}}} + \underbrace{\sum_{i=t+1}^{2t} p_{s(i)} + 6\left(t^2+1\right)A}_{\text{cost of } J_0'} \\
&\qquad + \underbrace{\sum_{i=t+1}^{2t} p_{s(i)} + 3\left(t^2+1\right)A}_{\text{cost of } J_0''} + \underbrace{\sum_{i=t+1}^{2t} p_{s(i)}}_{\text{cost of } J_0'''} \\
&= \sum_{i=1}^{t} (i) p_{s(i)} + \sum_{i=t+1}^{2t} (2t-i+1) p_{s(i)} + 9\left(t^2+1\right)A + 2\sum_{i=t+1}^{2t} p_{s(i)} \\
&= \sum_{i=1}^{t} (i) p_{s(i)} + \sum_{i=1}^{t} (i) p_{s(2t+1-i)} + \underbrace{11\left(t^2+1\right)A}_{d} - 2\left(t^2+1\right)A + 2\sum_{i=t+1}^{2t} p_{s(i)} \qquad (37) \\
&\geq \underbrace{\sum_{i=1}^{t} \left[i\left(p_{2i-1} + p_{2i}\right)\right]}_{y_0} + d - \underbrace{2\left(t^2+1\right)A}_{\sum_{i=1}^{2t} p_i} + 2\sum_{i=t+1}^{2t} p_{s(i)} \qquad (38)
\end{aligned}
$$

$$= y_0 - \sum_{i=1}^{2t} p_i + 2\sum_{i=t+1}^{2t} p_{s(i)} = y_0 - 2\underbrace{\sum_{i=1}^{t} p_{s(i)}}_{B_s - k} + \sum_{i=1}^{t} p_{s(i)} + \sum_{i=t+1}^{2t} p_{s(i)}$$

$$= y_0 - 2\left(B_s - k\right) + \sum_{i=1}^{2t} p_i = y_0 - 2\left(d + \left(t^2+1\right)A - k\right) + \sum_{i=1}^{2t} p_i = y_0 - 2(d-k).$$

The smallest value the expression $\sum_{i=1}^{t}\left[i\left(p_{s(i)} + p_{s(2t+1-i)}\right)\right]$ could conceivably attain is acquired by matching the longest processing times with the smallest coefficients – as in $\sum_{i=1}^{t}\left[i\left(p_{2i-1} + p_{2i}\right)\right]$ – and lends validity to the transition from (37) to (38).

The key to relating $f_k(S)$ to the actual cost incurred by the jobs in S with respect to d is to rely on items ii and v in Lemma 6.2 to argue that $C_{s(t+2)} < d \leq C_{s(t)}$ must be satisfied if $f(S) \leq y_0$ because at least $t-1$ and at most t partition jobs must start and complete after the due date. Observe that a total of $B(S) = t+3$ jobs complete at or before k in S, and the corresponding number of tardy jobs is $A(S) = t$. We analyze three relevant cases by keeping in mind that the change in the total cost of S per unit time change in the due date from its current value k can be computed based on $A(S)$ and $B(S)$ for $C_{s(t+2)} < d \leq C_{s(t)}$.

- If $d = k$, then $f(S) \geq y_0$.
- If $k < d \leq C_{s(t)}$, then $f(S) \geq y_0 - 2(d-k) + \left[B(S) - A(S)\right](d-k) = y_0 + (d-k) > y_0$.
- If $C_{s(t+2)} < d < k$, then $f(S) \geq y_0 - 2(d-k) + \left[(A(S)+1) - (B(S)-1)\right](k-d) = y_0 + 2(k-d) + \left[(t+1) - (t+2)\right](k-d) = y_0 + (k-d) > y_0$.

This analysis concludes that $f(S) \leq y_0$ prevails only if $d = k$ and the transition from (37) to (38) preserves the equality; that is, if $\sum_{i=1}^{t}\left[i\left(p_{s(i)} + p_{s(2t+1-i)}\right)\right] = \sum_{i=1}^{t}\left[i\left(p_{2i-1} + p_{2i}\right)\right]$. If these two properties hold, we arrive at the conclusion that the schedule S is identical to the schedule S_0 depicted in Figure 6, and consequently, that the underlying instance of the `EVEN-ODD PARTITION` problem is a *yes*-instance. □

References

Adiri, I., Bruno, J., Frostig, E., and Kan, A. R. (1989). Single machine flow-time scheduling with a single breakdown. *Acta Informatica*, 26(7):679–696.

Bagchi, U., Sullivan, R. S., and Chang, Y.-L. (1986). Minimizing mean absolute deviation of completion times about a common due date. *Naval Research Logistics Quarterly*, 33(2):227–240.

Baker, K. R. and Scudder, G. D. (1990). Sequencing with earliness and tardiness penalties: a review. *Operations Research*, 38(1):22–36.

Benmansour, R., Allaoui, H., and Artiba, A. (2011). Single machine scheduling problem in a just-in-time environment. In *Logistics (LOGISTIQUA), 2011 4th International Conference on*, pages 362–366. IEEE.

Benmansour, R., Allaoui, H., Artiba, A., and Hanafi, S. (2014). Minimizing the weighted sum of maximum earliness and maximum tardiness costs on a single machine with periodic preventive maintenance. *Comp. & Operations Research*, 47:106–113.

Chen, W.-J. (2009). Minimizing number of tardy jobs on a single machine subject to periodic maintenance. *Omega*, 37(3):591–599.

Cui, W.-W. and Lu, Z. (2017). Minimizing the makespan on a single machine with flexible maintenances and jobs' release dates. *Computers & Operations Research*, 80:11–22.

Detienne, B. (2014). A mixed integer linear programming approach to minimize the number of late jobs with and without machine availability constraints. *European Journal of Operational Research*, 235(3):540–552.

Drozdowski, M., Jaehn, F., and Paszkowski, R. (2017). Scheduling position-dependent maintenance operations. *Operations Research*, 65(6):1657–1677.

Emmons, H. (1987). Scheduling to a common due date on parallel uniform processors. *Naval Research Logistics*, 34(6):803–810.

Federgruen, A. and Mosheiov, G. (1997). Single machine scheduling problems with general breakdowns, earliness and tardiness costs. *Operations Research*, 45(1):66–71.

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

Garg, A. and Deshmukh, S. (2006). Maintenance management: literature review and directions. *Journal of Quality in Maintenance Engineering*, 12(3):205–238.

Graves, G. H. and Lee, C.-Y. (1999). Scheduling maintenance and semiresumable jobs on a single machine. *Naval Research Logistics*, 46(7):845–863.

Hall, N. G. (1986). Single-and multiple-processor models for minimizing completion time variance. *Naval Research Logistics Quarterly*, 33(1):49–54.

Hall, N. G., Kubiak, W., and Sethi, S. P. (1991). Earliness–tardiness scheduling problems, ii: Deviation of completion times about a restrictive common due date. *Operations Research*, 39(5):847–856.

Hoogeveen, J. and Van de Velde, S. (1991). Scheduling around a small common due date. *European J. of Operational Research*, 55(2):237–242.

Huo, Y. (2017). Parallel machine makespan minimization subject to machine availability and total completion time constraints. *Journal of Scheduling*, pages 1–15.

Ji, M., He, Y., and Cheng, T. E. (2007). Single-machine scheduling with periodic maintenance to minimize makespan. *Computers & Operations Research*, 34(6):1764–1770.

Kacem, I. and Chu, C. (2008). Efficient branch-and-bound algorithm for minimizing the weighted sum of completion times on a single machine with one availability constraint. *International Journal of Production Economics*, 112(1):138–150.

Kacem, I., Chu, C., and Souissi, A. (2008). Single-machine scheduling with an availability constraint to minimize the weighted sum of the completion times. *Computers & operations research*, 35(3):827–844.

Kanet, J. (1981). Minimizing the average deviation of job completion times about a common due date. *Naval Research Logistics Quarterly*, 28(4):643–651.

Kanet, J. J. and Sridharan, V. (2000). Scheduling with inserted idle time: problem taxonomy and literature review. *Operations Research*, 48:99–110.

Laalaoui, Y. and M'Hallah, R. (2016). A binary multiple knapsack model for single machine scheduling with machine unavailability. *Comp. & Operations Research*, 72:71–82.

Lee, C.-Y. (1996). Machine scheduling with an availability constraint. *J. of Glob. Optim.*, 9:395–416.

Lee, C.-Y. (1999). Two-machine flowshop scheduling with availability constraints. *European Journal of Operational Research*, 114(2):420–429.

Lee, J.-Y. and Kim, Y.-D. (2012). Minimizing the number of tardy jobs in a single-machine scheduling problem with periodic maintenance. *Computers & Operations Research*, 39(9):2196–2205.

Leon, V. J. and Wu, S. . D. (1992). On scheduling with ready-times, due-dates and vacations. *Naval Research Logistics*, 39:53–65.

Liu, M., Wang, S., Chu, C., and Chu, F. (2016). An improved exact algorithm for single-machine scheduling to

minimise the number of tardy jobs with periodic maintenance. *International Journal of Production Research*, 54(12):3591–3602.

Liu, Z. and Sanlaville, E. (1997). Stochastic scheduling with variable profile and precedence constraints. *SIAM Journal on Computing*, 26(1):173–1997.

Low, C., Ji, M., Hsu, C.-J., and Su, C.-T. (2010). Minimizing the makespan in a single machine scheduling problems with flexible and periodic maintenance. *Applied Mathematical Modelling*, 34(2):334–342.

Low, C., Li, R.-K., Wu, G.-H., and Huang, C.-L. (2015). Minimizing the sum of absolute deviations under a common due date for a single-machine scheduling problem with availability constraints. *Journal of Industrial and Production Engineering*, 32(3):204–217.

Ma, Y., Chu, C., and Zuo, C. (2010). A survey of scheduling with deterministic machine availability constraints. *Comp. & Industrial Engineering*, 58(2):199–211.

Mannur, N. R. and Addagatla, J. B. (1993). Heuristic algorithms for solving earliness-tardiness scheduling problem with machine vacations. *Computers and Industrial Engineering*, 25(1-4):255–258.

Molaee, E., Moslehi, G., and Reisi, M. (2011). Minimizing maximum earliness and number of tardy jobs in the single machine scheduling problem with availability constraint. *Computers and Mathematics with Applications*, 62:3622–3641.

Rapine, C., Brauner, N., Finke, G., and Lebacque, V. (2012). Single machine scheduling with small operator-non-availability periods. *Journal of Scheduling*, 15(2):127–139.

Schmidt, G. (2000). Scheduling with limited machine availability. *European Journal of Operational Research*, 121(1):1–15.

Ventura, J. A. and Weng, M. X. (1995). An improved dynamic programming algorithm for the single-machine mean absolute deviation problem with a restrictive common due date. *Operations Research Letters*, 17(3):149–152.

Wang, G., Sun, H., and Chu, C. (2005). Preemptive scheduling with availability constraints to minimize total weighted completion times. *Annals of Operations Research*, 133(1-4):183–192.

Yin, Y., Cheng, T., and Wang, D.-J. (2016a). Rescheduling on identical parallel machines with machine disruptions to minimize total completion time. *European Journal of Operational Research*, 252(3):737–749.

Yin, Y., Wang, Y., Cheng, T., Liu, W., and Li, J. (2017). Parallel-machine scheduling of deteriorating jobs with potential machine disruptions. *Omega*, 69:17–28.

Yin, Y., Xu, J., Cheng, T., Wu, C.-C., and Wang, D.-J. (2016b). Approximation schemes for single-machine scheduling with a fixed maintenance activity to minimize the total amount of late work. *Naval Research Logistics (NRL)*, 63(2):172–183.

Yoo, J. and Lee, I. S. (2016). Parallel machine scheduling with maintenance activities. *Comp. & Industrial Engineering*, 101:361–371.