Journal of Scheduling
https://doi.org/10.1007/s10951-020-00665-4

Scheduling jobs with a V-shaped time-dependent processing time

Helmut A. Sedding[1,2]

Abstract
In the field of time-dependent scheduling, a job's processing time is specified by a function of its start time. While monotonic processing time functions are well-known in the literature, this paper introduces non-monotonic functions with a convex, piecewise-linear V-shape similar to the absolute value function. They are minimum at an ideal start time, which is the same for all given jobs. Then, the processing time equals the job's basic processing time. Earlier or later, it increases linearly with slopes that can be asymmetric and job-specific. The objective is to sequence the given jobs on a single machine and minimize the makespan. This is motivated by production planning of moving car assembly lines, in particular, to sequence a worker's assembly operations such that the time-dependent walking times to gather materials from the line-side is minimized. This paper characterizes the problem's computational complexity in several angles. NP-hardness is observed even if the two slopes are the same for all jobs. A fully polynomial time approximation scheme is devised for the more generic case of agreeable ratios of basic processing time and slopes. In the most generic case with job-specific slopes, several polynomial cases are identified.

Keywords Single-machine scheduling · Time-dependent scheduling · Non-monotonic processing time · Piecewise-linear processing time · V-shaped processing time

1 Introduction

Sequencing a set of jobs on a single machine such that the makespan is minimized is trivial if each job's processing time is constant, because any job sequence is optimal in this case. In contrast, if each job's processing time is a function of its start time, then the job sequence alters the processing times. For example, if a swap of two jobs changes the sum of their processing times, then all succeeding jobs are shifted, which possibly necessitates a reoptimization. Hence, time-dependent processing times add a layer of complexity, and already the makespan minimization poses a challenge.

1.1 Processing time function

In time-dependent scheduling, the classic effect of a job's start time on its *basic processing time* is additive. Here, a *penalty function* ϖ_j of start time t is added to the basic processing time $\ell_j \geq 0$ of a job j to obtain the processing time

$$p_j(t) = \ell_j + \varpi_j(t) \tag{1}$$

of job j. Then, the job is completed at

$$C_j(t) = t + p_j(t). \tag{2}$$

Consequently, the completion time of a sequence of several jobs equals the composition of their completion time functions. For example, consider job sequence $(1, 3, 2)$: If it is started at time t, then it completes at $C_2(C_3(C_1(t)))$.

Although existing literature studies many variations of ϖ_j, they are largely restricted to monotonic, i.e., nondecreasing or nonincreasing forms (Gawiejnowicz 2008, 2020a, b; Strusevich and Rustogi 2017). A present practical case, arising in the context of moving assembly lines, requires a non-monotonic penalty function (Sedding 2020b), joining research lines on monotonic forms.

In particular, we explore the job-specific, non-monotonic, piecewise-linear V-shaped penalty function

$$\varpi_j(t) = \max\{-a_j\,(t-\tau),\ b_j\,(t-\tau)\}. \tag{3}$$

✉ Helmut A. Sedding
helmut.sedding@zhaw.ch

[1] Institute of Theoretical Computer Science, Ulm University, Ulm, Germany

[2] Institute of Data Analysis and Process Design, ZHAW Zurich University of Applied Sciences, Winterthur, Switzerland

Published online: 08 September 2020

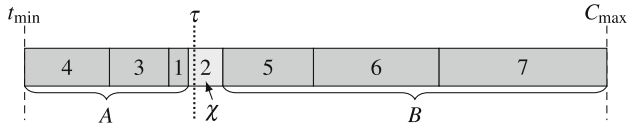

Fig. 1 Given an example instance of the studied problem $\mathcal{P}$ with the global start time $t_{\min} = 0$, the common ideal start time $\tau = 10$, eight jobs, and for job $j = 1, \ldots, 7$, the basic processing time $\ell_j = j$ and the two common slopes $a_j = 0.1$, $b_j = 0.2$. Depicted is the only one job sequence that provides the minimum makespan for this instance. It arranges job 2 as the straddler job χ, and partitions the other jobs into set $A = \{4, 3, 1\}$ and set $B = \{5, 6, 7\}$. Please observe that the jobs of A and B are sequenced in opposite orders, and that in this example, the straddler job χ is not the job with the smallest basic processing time

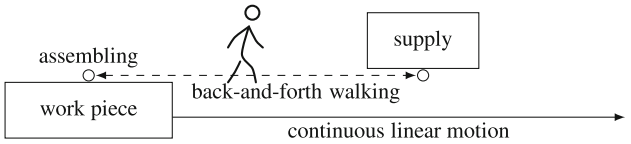

Fig. 2 The studied problem $\mathcal{P}$ models a moving assembly line planning problem, in which a worker's walking time to a material supply point and back depends on the current position of the worker's continuously moving work piece. The supply point is passed by the work piece at a certain point in time, τ, at which the incurred walking time is minimum. Earlier or later, is increases linearly with asymmetric slopes, which relate the back-and-forth walking velocities to the work piece's velocity

It joins two linear pieces at one certain point, the so-called *common ideal start time* τ, which is the same for all jobs. Each linear piece is described by a job-specific *slope*, namely $1 \geq a_j \geq 0$ and $b_j \geq 0$. Note that all numbers are rational. We observe that the domains of a_j and b_j ensure a nondecreasing completion time function C_j. Thus, delaying a job by inserting idle time does not reduce its completion time.

1.2 Problem setting

For a set of jobs of the described time-dependent processing times (1) with the additive V-shaped penalty functions (3), let us define the scheduling problem $\mathcal{P}$. Several rational numbers define an instance of $\mathcal{P}$: a start time $t_{\min}$ for the first job, a common ideal start time τ, and for each job j, a basic processing time $\ell_j \geq 0$ and slopes $1 \geq a_j \geq 0$, $b_j \geq 0$. A permutation of the jobs (a so-called *job sequence*) determines the order in which to successively execute the jobs starting from $t_{\min}$ on a single machine without idle time, completing at $C_{\max}$. Then, the objective in $\mathcal{P}$ is to find a job sequence that minimizes the *makespan* $\phi = C_{\max} - t_{\min}$. Such a sequence is called optimal, and solves the $\mathcal{P}$ instance.

For sequencing a given set of jobs, one needs to decide

- which jobs should complete before the common ideal start time τ (denoted by job set A), and
- which job should be the first job that starts before or at τ and that as well completes at or after τ (this job is called the *straddler job* χ) if it exists;
- then, the remaining jobs (excluding the straddler job, if it exists) all start at or after τ (job set B).

Once this decision is made, the corresponding job sequence can be constructed in polynomial time by sorting set A and set B, and linking them, if applicable, with the straddler job χ in between. Thus, the main computational effort to find an optimal job sequence resides in choosing a suitable partition into the two sets. Figure 1 visualizes an optimal job sequence and the described parts of an example instance.

Let us mention special cases of $\mathcal{P}$:

- Case $\mathcal{P}_{agreeable}$ asserts $\ell_i a_j \leq \ell_j a_i \iff \ell_i b_j \leq \ell_j b_i$ for any pair i, j of jobs, which we call *agreeable ratios of basic processing time and slopes*.
- This property is also fulfilled by the special case of *related slopes*, which scales common basic slopes $1 \geq a \geq 0$, $b \geq 0$ by a job-specific rational scale factor $1 \geq v_j \geq 0$ to $a_j = a v_j$ and $b_j = b v_j$ for each job j.
- Special cases of related slopes are *monotonic slopes* where either $a_j = 0$ for each job j, which yields a nondecreasing p_j, or $b_j = 0$ for each job j, which yields a nonincreasing p_j, and
- *common slopes* $a_j = a, b_j = b$ (case $\mathcal{P}_{common}$).

Our practical motivation for the described scheduling problem is to minimize costly walking time of workers at a moving automobile assembly line. This is attained by minimizing the makespan of each worker independently. A worker needs to complete a set of assembly operations (jobs) in any order at a his or her work piece, which is continuously transported by a conveyor belt. Each assembly operation consists of a constant assembly time and, before that, a time-dependent walking time to gather material from a central supply point at the line-side (see also Fig. 2).

The resulting assembly operation times can be adequately modeled by the described time-dependent processing times (Sedding 2020b). By permuting the operations, it is possible to minimize the total walking time, or equivalently, the worker's makespan.

2 Summary of results and organization

The results presented in this paper can be summarized as follows:

- Identification of three polynomial cases: first, if $t_{\min} \geq \tau$; second, if a certain job sequence starts each job before or at τ; third, if each basic processing time is zero.

- Proof that the studied problem is NP-hard already for the special case $\mathcal{P}_{common}$ of common slopes. This is shown by reduction from Even-Odd Partition. See Table 2 for an overview.
- Introduction of a fully polynomial time approximation scheme (FPTAS) for the case $\mathcal{P}_{agreeable}$ of agreeable ratios of basic processing time and slopes. This approach can be also be used in the common slope case and known monotonic slope cases, see Table 3 for an overview. Notably, the underlying dynamic program is not pseudopolynomial, which is exceptional (Garey and Johnson 1979, p. 140). Because the objective value can be exponential in input length and input values, please note that the existence of an FPTAS neither implies the existence of a pseudopolynomial algorithm, nor rules out NP-hardness in the strong sense.

This paper is structured as follows. In Sect. 3, relevant literature is reviewed and the practical motivation of the study is described. In Sect. 4, our notation for job sequences is given, and properties of the makespan calculation are presented. Polynomial cases are identified in Sect. 5. A symmetry property in optimal job sequences is described in Sect. 6. In Sect. 7, it is shown that $\mathcal{P}_{common}$ is NP-hard. In Sect. 8, a dynamic program is introduced for $\mathcal{P}_{agreeable}$, which is used to construct an FPTAS in Sect. 9.

3 Literature review

The studied non-monotonic penalty function (3) covers monotonic special cases in the literature, because it allows all-zero a_j (or b_j) slopes. A similar generalization occurred within the scheduling literature on constant processing times, which is summarized in the first subsection. We then continue with a review of relevant time-dependent literature. Finally, we describe the practical application that prompted the presented non-monotonic case.

3.1 Literature with constant processing times

From a historical point of view, a shift from (a) proportional to (b) monotonic piecewise-linear, then to (c) non-monotonic piecewise-linear measures similarly occurred before in the classic scheduling theory in terms of weighted completion costs, namely from the total weighted completion time criterion to the total weighted tardiness criterion with a common due date, then to the total weighted earliness and tardiness criterion with a restrictive common due date.

(a) $\sum_j w_j C_j$ A basic scheduling problem is to minimize the monotonic total weighted completion time $\sum_j w_j C_j$ with a given job-weight w_j for each job j, achieved by sorting the jobs by nonincreasing ratio w_j/p_j (Smith 1956).

(b) $\sum_j w_j T_j$ A harder problem is to minimize the piecewise-linear monotonic total weighted tardiness $\sum_j w_j T_j$ with job-tardiness $T_j = \max\{0,\ C_j - d\}$ for a given common due date d; it requires equal weights $w_j = w$ for a polynomial-time algorithm (Lawler and Moore 1969). Optimal job sequences can be divided into a set A of jobs completing before d, a straddler job starting before d and completing at or after d, and a set B of jobs starting at or after d. The order of jobs in A is arbitrary; set B is sorted according to Smith (1956). Therefore, an algorithm mainly needs to decide on a straddler job and partition the remaining jobs into sets A and B (Lawler and Moore 1969). For job-specific weights, the latter decision is NP-hard, as shown in Yuan (1992) by reduction from Partition. A pseudopolynomial-time dynamic programming algorithm is devised in Lawler and Moore (1969), a strongly polynomial FPTAS in Kacem (2010) for a given straddler job, see Kianfar and Moslehi (2013).

(c) $\sum_j w_j (E_j + T_j)$ A further complexity increase is caused by the piecewise-linear nonmonotonic total weighted earliness and tardiness criterion $\sum_j w_j (E_j + T_j)$ with job-earliness $E_j = \max\{d - C_j, 0\}$ and a so-called restrictive common due date $d < \sum_j p_j$. In optimal job sequences, the jobs are arranged in opposing orders around d: nondecreasingly by w_j/p_j before d and nonincreasingly by w_j/p_j after d. Again, it necessitates to decide on the straddler job and job set A and B. This problem is NP-hard already for common weights $w_j = w$, which is shown by reduction from Even-Odd-Partition, and permits a pseudopolynomial-time dynamic programming algorithm (Hall et al. 1991; Hoogeveen and van de Velde 1991). Kellerer and Strusevich (2010) show that the problem admits a strongly polynomial FPTAS by adopting an FPTAS for Symmetric Quadratic Knapsack.

An overview of complexity results for these classic scheduling problems is given in Table 1.

3.2 Literature on time-dependent scheduling

Time-dependent scheduling with the objective of minimizing the makespan is a research stream that dates back to Shafransky (1978); Melnikov and Shafransky (1979). The latter study job-uniform monotonic penalty functions $\varpi = \varpi_j$; hence, ϖ is nondecreasing or nonincreasing. For this generic model, they show that an optimal job sequence is found in polynomial time by sorting the jobs with respect to ℓ_j.

Turning to job-specific penalty functions ϖ_j, an interesting special case arises for all-zero basic processing times $\ell_j = 0$, which means that $p_j = \varpi_j$. This case is consid-

Table 1 Complexity results on related classic objectives

Objective	Common weights $w_j = w$	Job-specific weights w_j
$\sum_j w_j C_j$	∘ Polynomial (Smith 1956)	∘ Polynomial (Smith 1956)
$\sum_j w_j T_j$	∘ Polynomial (Lawler and Moore 1969)	∘ NP-hard (Yuan 1992)
		∘ Strongly polynomial FPTAS (Kacem 2010)
$\sum_j w_j (E_j + T_j)$	∘ NP-hard (Hall et al. 1991; Hoogeveen and van de Velde 1991)	∘ Strongly polynomial FPTAS (Kellerer and Strusevich 2010)

ered in the following three studies. Mosheiov (1994) studies the proportionally increasing penalty function $\varpi_j(t) = b_j\, t$ for $b_j \geq 0$ and a positive global start time $t_{\min} > 0$, and shows that any job sequence yields the same makespan and is optimal. Kawase et al. (2018) analyze monotonic piecewise-linear penalty functions equivalent to $\varpi_j(t) = \min\{0,\ (b_j - 1) \cdot t + c_j\}$ for $b_j \geq 0$, and show that an optimal job sequence is computed in polynomial time by sorting the jobs. Kononov (1998) considers non-monotonic penalty functions $\varpi_j(t) = b_j \cdot h(t)$ with a common convex or concave function h where, for any t, t' with $t' \geq t \geq t_{\min}$ and each job j, there holds $h(t_{\min}) > 0$ and $t' + b_j\, h(t') \geq t + b_j\, h(t)$. Note that the second condition on b_j and h is equivalent to restricting job j's completion time to be nondecreasing for any start time $t \geq t_{\min}$. Kononov (1998) shows that the minimum makespan is attained by sequencing the jobs in nondecreasing order with respect to b_j (or nonincreasing for concave h), see also Gawiejnowicz (2008,Theorem 6.43) for a description.

With nonnegative basic processing times $\ell_j \geq 0$, finding a job sequence with a minimum makespan is computationally more involved. The categorization for the classic scheduling models in Sect. 3.1 can be translated to (a) proportional penalty functions ϖ_j, (b) monotonic piecewise-linear ϖ_j, and (c) non-monotonic piecewise-linear ϖ_j. This categorization is elaborated below and visualized in Fig. 3. An overview of the complexity results is given in Table 2, a runtime comparison of the FPTASs in Table 3.

(a) *Proportional* ϖ_j The proportional increasing penalty function $\varpi_j(t) = b_j\, t$ with $b_j \geq 0$ is independently studied in Shafransky (1978), Wajs (1986), Gupta and Gupta (1988), Browne and Yechiali (1990), and Gawiejnowicz and Pankowska (1995). They show that an optimal sequence sorts the jobs nondecreasingly with respect to ℓ_j/b_j and so that all jobs with $b_j = 0$ are last. Gawiejnowicz (2008), Theorem 6.24 summarizes multiple ways for proving this: by partial order relations (Gawiejnowicz and Pankowska 1995), by a job interchange argument (Wajs 1986; Gupta and Gupta 1988), and by its formalized concept, the so-called priority-generating function (Shafransky 1978); for the latter also see Tanaev et al. (1984, 1994, chapter 3, section 1.2).

The symmetric case with proportional decreasing penalty functions $\varpi_j(t) = -a_j\, t$ with $0 \leq a_j < 1$ is considered first in Ho et al. (1993). Here, the jobs need to be nonincreasingly ordered by ℓ_j/a_j while jobs with $a_j = 0$ are last (Ho et al. 1993; Gordon et al. 2008).

(b) *Monotonic piecewise-linear* ϖ_j Adding a point in time until which the processing time is constant results in the piecewise-linear, job-specific, nondecreasing penalty function $\varpi_j(t) = \max\{0,\ b_j\,(t-\tau)\}$ for a given common τ. Then, the decision version of the scheduling problem is NP-hard, as shown in Kononov (1997) by reduction from Subset Sum, and in Kubiak and van de Velde (1998) by reduction from Partition. Kubiak and van de Velde (1998) also present a pseudopolynomial-time algorithm. FPTASs are described in Cai et al. (1998), Kovalyov and Kubiak (1998). Woeginger (2000), Kovalyov and Kubiak (2012), and Halman (2019) build upon Kovalyov and Kubiak (1998). Our independently devised FPTAS also applies. Retrospectively, it is most similar to Cai et al. (1998). All approaches use techniques for trimming-the-state-space as Ibarra and Kim (1975) except for Halman (2019) with K-approximation sets. All rely on the problem's property of allowing for the same order of jobs before and after τ: nondecreasingly by ℓ_j/b_j. Moreover, Kovalyov and Kubiak (1998) require that a straddler job χ completes at an integer valued completion time C_χ in order to repeat the calculation for a polynomial number of possible C_χ.

A symmetric problem exhibits similar properties and is introduced in Cheng et al. (2003) by the nonincreasing penalty function $\varpi_j(t) = \max\{-a_j\,(t - \tau),\ 0\}$ for $0 < a_j < 1$ and $\ell_j > a_j \min\{\tau,\ \sum_{k \neq j} \ell_k\}$. Cheng et al. (2003) prove NP-hardness by reduction from Partition, and introduce a pseudopolynomial-time algorithm. Later, Ji and Cheng (2007) devise an FPTAS for it by utilizing methods from Kovalyov and Kubiak (1998) and by relying on the same order of the job sets before and after τ: nonincreasingly with respect to ℓ_j/a_j. Moreover, they utilize the problem's property that the value of a straddler job's completion time only linearly influences the makespan because the processing times of the jobs that start at or after τ are constant.

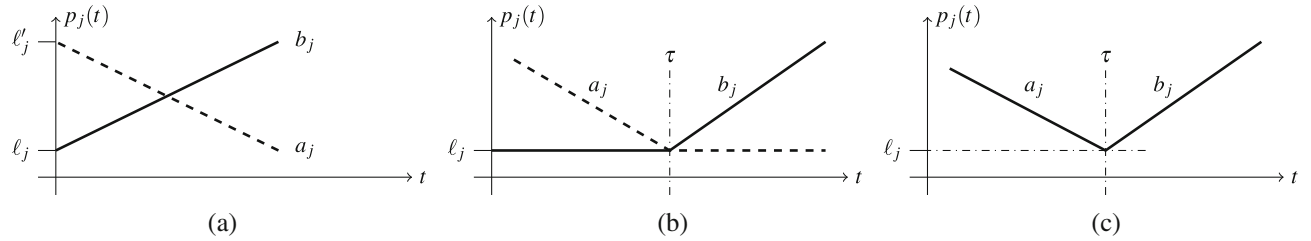

Fig. 3 Time-dependent scheduling models for processing times $p_j = \ell_j + \varpi_j(t)$ with an additive penalty function ϖ_j are mostly restricted to monotonic piecewise-linear ϖ_j in the literature, like (**a**) $\varpi_j(t) = b_j\, t$ or $\varpi_j(t) = -a_j\, t$, (**b**) $\varpi_j(t) = \max\{0,\, b_j\,(t-\tau)\}$ or $\varpi_j(t) = \max\{-a_j\,(t-\tau),\, 0\}$. These models are unified in this paper with (**c**) $\varpi_j(t) = \max\{-a_j(t-\tau),\, b_j(t-\tau)\}$

Table 2 Complexity results on single machine time-dependent scheduling with processing time $p_j(t) = \ell_j + \varpi_j(t)$ for an additive penalty function $\varpi_j(t) = \min\{-a_j\,(t-\tau),\, b_j\,(t-\tau)\}$ of job j's start time t in several settings of the real-valued slopes $0 \le a_j \le 1, b_j \ge 0$, assuming $t_{\min} = 0$

a_j	b_j	Complexity	Complexity of selected special cases
0	b	○ polynomial (Melnikov and Shafransky 1979)	
a	0	○ polynomial (Melnikov and Shafransky 1979)	
0	b_j	○ NP-hard (Kononov 1997; Kubiak and van de Velde 1998) • FPTAS (Theorem 2; Cai et al. 1998; Kovalyov and Kubiak 1998; Woeginger 2000; Halman 2019)	• polynomial if $\tau \le 0$ (Lemma 4; Shafransky 1978; Wajs 1986; Gupta and Gupta 1988; Browne and Yechiali 1990; Gawiejnowicz and Pankowska 1995; Tanaev et al. 1984, 1994)
a_j	0	○ NP-hard (Cheng et al. 2003) • FPTAS (Corollary 10; Ji and Cheng 2007)	• polynomial if a $\ell_j/a_j\searrow$-sorted sequence where the smaller ℓ_j is first for ties starts all jobs before or at τ (Lemma 5)
a	b	• NP-hard (Theorem 1) • FPTAS (Corollary 9)	• polynomial if $\tau \le 0$ (Lemma 4), or if $\tau = \infty$ (Ho et al. 1993) • polynomial if ordering the jobs nondecreasingly by ℓ_j starts each job before or at τ (Lemma 5)
a_j	b_j	• NP-hard (Corollary 7)	• polynomial if $\tau \le 0$ (Lemma 4), or if $\tau = \infty$ (Ho et al. 1993) • polynomial if a $\ell_j/a_j\searrow$-sorted sequence with smaller ℓ_j first (in the case of ties) starts all jobs before or at τ (Lemma 5) • polynomial if $\ell_j = 0$ (Lemma 6) • FPTAS for agreeable ratios of ℓ_j and the slopes (Theorem 2)

• Marks results devised or confirmed in this work,
○ Marks results from the literature

Table 3 Comparison of FPTASs' worst-case runtime for $\mathcal{P}_{agreeable}$ with n jobs and error $\varepsilon \in (0, 1]$ in different settings of slopes, where $\ell_{\max}$ denotes the maximum l_j, and $b_{\max}$ denotes the maximum b_j

a_j	b_j	FPTAS runtime in the agreeable ratios of basic processing time and slopes case $\mathcal{P}_{agreeable}$	
a_j	b_j	• $\mathcal{O}\big(n^5 \cdot \log(1+b_{\max}) \cdot (\log \ell_{\max} + n \log(1+b_{\max}))/\varepsilon^2\big)$	Theorem 2
0	b_j	○ $\mathcal{O}\big(n^5 \cdot \log(1+b_{\max}) \cdot (\log \ell_{\max} + n \log(1+b_{\max}))/\varepsilon^2\big)$	Cai et al. (1998) (runtime stated in Halman 2019)
		○ $\mathcal{O}\big(n^5 \cdot \log^4 \max\{n, 1/\varepsilon, \ell_{\max}, 1+b_{\max}\}/\varepsilon^3\big)$	Kovalyov and Kubiak (1998, 2012)
		○ $\mathcal{O}\big(n^4 \cdot \log^3 \max\{\ell_{\max}, 1+b_{\max}, \tau\} \cdot \log(n \log \max\{\ell_{\max}, 1+b_{\max}, \tau\}/\varepsilon)/\varepsilon^2\big)$	Halman (2019)
		• $\mathcal{O}\big(n^5 \cdot \log(1+b_{\max}) \cdot (\log \ell_{\max} + n \log(1+b_{\max}))/\varepsilon^2\big)$	Theorem 2
a_j	b	• $\mathcal{O}\big(n^4 \cdot (\log \ell_{\max} + n \log(1+b_{\max}))/\varepsilon\big)$	Corollary 9
a_j	0	○ $\mathcal{O}\big(n^3 \cdot \log^3 \max\{n, 1/\varepsilon, \ell_{\max}\}/\varepsilon^2\big)$	Ji and Cheng (2007)
		• $\mathcal{O}\big(n^3 \cdot \log(\ell_{\max} n)/\varepsilon\big)$	Corollary 10

For a common ground, let ℓ_j and b_j be natural numbers. For runtimes given in the literature, we rather state $1 + b_{\max}$ instead of just $b_{\max}$ to avoid adverse effects of zero $\log b_{\max}$ if $b_{\max} = 1$. As it is common, $\log^k x = (\log x)^k$
• Marks results devised in this work,
○ Marks results from the literature

(c) *Non-monotonic piecewise-linear* ϖ_j The described forms are extended by the non-monotonic piecewise-linear penalty function $\varpi_j(t) = \max\{-a_j\,(t-\tau),\ b_j\,(t-\tau))\}$ in $\mathcal{P}$, which has, to the best of our knowledge, not been studied up to now. The following studies lie closest. Farahani and Hosseini (2013) study the special case of such a penalty function with symmetric, common (all-equal) slopes $0 < a < 1, a = a_j = b_j$, while treating the global start time $t_{\min}$ as a decision variable with the objective of minimizing the cycle time $C_{\max} - t_{\min}$. Then, an optimal schedule exhibits the following properties: one job χ starts exactly at τ, the set A of jobs that complete before and at τ are sorted nonincreasingly by ℓ_j, and the set $\{\chi\}\cup B$ of jobs starting at or after τ are sorted non-decreasingly by ℓ_j. An exact polynomial time algorithm sets $\chi = \operatorname{argmin}_j \ell_j$ and assigns the other jobs iteratively to A and B. They describe a practical application of their problem setting related to scheduling a vehicle for delivery of commodities between two rush hours in an urban setting, assuming added travel time first decreases, then rises back up later on.

A similar non-monotonic time-dependent effect is considered in Jaehn and Sedding (2016). However, the model measures a job's *middle* time, instead of its start time for determining the processing time of a job j. In particular, it is stated by $p_j = \ell_j + a \cdot |m - M|$ with slope $0 < a < 2$, ideal middle time M, and the job's middle time m, which is related to the job's start time t by $m = t + p_j/2$, and specifies the point in time when exactly half of the job has been processed. Solving m for p_j and t yields the processing time function

$$p_j(t) = \begin{cases} \dfrac{\ell_j - a\,(t - M)}{1 + a/2}, & t < M - \ell_j/2, \\ \dfrac{\ell_j + a\,(t - M)}{1 - a/2}, & t \geq M - \ell_j/2. \end{cases} \tag{4}$$

This function is not expressible in terms of the ϖ_j penalty function because (a) the start time-dependent processing time function has a job-specific minimum at $M - \ell_j/2$ instead of one common minimum at some common ideal start time τ, and (b) the basic processing time ℓ_j is scaled by two different factors, depending on the start either before or at and after $M - \ell_j/2$. Although this model seems rather unconventional, its convincing advantage is that it allows to study a perfectly symmetric job prolongation before and after M. For example, consider arbitrary middle times m' and m'' such that $m'' - M = M - m'$. If job j is scheduled such that $m_j = m'$, then it starts at t' and completes at C'. Correspondingly, if $m_j = m''$, it starts at t'' and completes at C''. Then, there is $t'' - M = M - C'$ and $C'' - M = M - t'$. This symmetry around M allows for a polynomial reduction from Even-Odd Partition and prove the NP-hardness of the considered problem.

A much more generic problem is studied in Kawase et al. (2018) with the optimal composition ordering of convex or concave piecewise-linear functions. An interesting remark is that the minimization problem with functions $C_j(t)$ can be transformed to the maximization problem with $\tilde{C}_j(t) = -C_j(-t)$, and vice-versa. One of the studied cases is the maximum composition ordering of the concave $\tilde{C}_j(t) = \min\{a'_j\,t + a''_j,\ b'_j\,t + b''_j\}$ for $a'_j > 0, b'_j > 0$. Their result on this case is a NP-hardness proof by reduction from Partition. From this, we infer that the convex minimization counterpart $C_j(t) = -\tilde{C}_j(-t)$ is also NP-hard. We observe that a special case is problem $\mathcal{P}$ with parameters $a'_j = 1 - a_j$ (unless $a_j = 1$), $a''_j = \ell_j + a_j\tau$, $b'_j = 1 + b_j$, and $b''_j = \ell_j - b_j\tau$. Of course, as it is a special case, the hardness of $\mathcal{P}$ can, however, not be inferred from the more generic problem setting.

In addition, let us note that preliminary results of our paper are presented in Sedding (2017, 2020a) on the FPTAS, in Sedding (2018a, b) on the NP-hardness, and on both of them in Sedding (2020c) for the common slopes case $\mathcal{P}_{common}$.

A comprehensive treatise on the variety of time-dependent scheduling models is provided in Gawiejnowicz (2008, 2020a). A recent review is given in Gawiejnowicz (2020b). Further reviews are in Alidaee and Womer (1999), Błażewicz et al. (2019), Cheng et al. (2003), Agnetis et al. (2014), and Strusevich and Rustogi (2017).

3.3 Practical application in automobile production planning

The studied time-dependent scheduling problem $\mathcal{P}$ arises in production planning of moving assembly lines. A major German car manufacturer spends about 10–15% of working time at the moving final assembly line with fetching supplies from the line side (Scholl et al. 2013). This time expense incurs a high cost, and any reduction offers a high return. This walking time mainly occurs before the start of each assembly operation (job). There, the assembly worker needs to leave the continuously moving work piece, then walk along the assembly line to a nonmoving material supply point, and return to the same work piece (which continued to move during the worker's absence). See Figure 2 for a visualization of this scenario.

A worker's walking time is minimized by essentially two approaches. One is to reposition the supplies (Klampfl et al. 2006; Sedding 2020b); another is to resequence the worker's assembly operations (Sedding and Jaehn 2014; Jaehn and

Sedding 2016). We focus on the operation (re)sequencing approach, which avoids a physical reconfiguration of the assembly line; thus offers much faster reaction times on short term changes. The worker's operations are usually independent from each other. Hence, we can assume that any job sequence is feasible. The high number of possible job sequences raises the need for an algorithmic decision support (Sedding 2020c).

A special case is portrayed in Jaehn and Sedding (2016), where walking time occurs in the middle of an operation, which then exhibits a perfect symmetric processing time function (4). We need to deviate from this symmetry to consider a walking time that occurs at the start of each operation as in Klampfl et al. (2006) and Sedding (2020b).

We model the time-dependent walking time like in Sedding (2020b). Then, the walking time is proportional to the distance between the static supply point and the moving work piece. Hence, the walking time depends on the time that the worker starts to walk: the walking time is minimum when the working point just passes by the supply point. This is the ideal walking start time, which corresponds to τ in ϖ_j. Earlier or later, the walking time increases linearly.

Sedding (2020b) elaborates how conveyor and worker velocities are translated to asymmetric slopes $0 < a < 1$ and $b > 0$. The slopes' domains originate from an assembly line velocity that is generally lower than the worker velocity. Their asymmetry arises from the continuous conveyor movement, which is divided in two cases. While it moves the work piece towards the supply point, the walking time shortens. While it moves the work piece away, the walking time increases.

Sometimes, properties of the carried material such as its weight can influence the walking velocity for some operations (Klampfl et al. 2006). In this case, job-specific slopes can be set, which typically yields a $\mathcal{P}_{agreeable}$ instance.

4 Preliminaries

In this section, our notation is introduced, and the makespan calculation is expressed in closed formulae.

4.1 Notation of sequences

We specify our notation of (job) sequences, and denote two sequence sort criteria.

Given a set J of n jobs, we denote by sequence $S = (S(1), \ldots, S(n))$ a permutation of the jobs in J, where $S(i)$ specifies the job that occupies position $i \in \{1, \ldots, n\}$. We denote by $S^{-1}(j)$ the position of job j in sequence S, hence $S(S^{-1}(j)) = j$. A sequence can be split, for example, we write $(1, 2, 3, \ldots, n) = S_1 S_2$ with $S_1 = (1, 2)$, $S_2 = (3, \ldots, n)$ (then, $S_2(1) = 3$).

The start time t and completion time C of a sequence corresponds to the start time of the first job and the completion time of the last job in the sequence. Then, the makespan of a sequence is $C - t$.

We say a sequence S of a set of jobs J is

- '$\ell_j/a_j \searrow$-sorted' if $\ell_j a_k \geq \ell_k a_j$, or
- '$\ell_j/b_j \nearrow$-sorted' if $\ell_j b_k \leq \ell_k b_j$

holds for any two jobs $j, k \in J$ at positions $S^{-1}(j) < S^{-1}(k)$, respectively.

Remark 1 For the set of all jobs in a $\mathcal{P}_{agreeable}$ instance, there exists a $\ell_j/a_j \searrow$-sorted sequence such that its reversed sequence is $\ell_j/b_j \nearrow$-sorted.

4.2 Makespan calculation

For a sequence S of a (sub)set J of n jobs of a $\mathcal{P}$ instance, the completion time is given in a recursive form by

$$C = C_{S(n)}\left(C_{S(n-1)}\left(\cdots C_{S(2)}(C_{S(1)}(t))\cdots\right)\right)$$

for the sequence's start time t. This recursive equation can be difficult to handle. However, it is possible to transform the calculation to a closed form, as we show in this subsection. Then, we state the derivatives of the sequence's completion time with respect to its start time if the sequence either starts at or after the ideal start time τ, or completes before or at τ.

First, we substitute p_j and f_j in C_j (see (2), (1), (3)) to

$$\begin{aligned} C_j(t) = \max\{&(1-a)\,t_j + \ell_j + a_j\tau,\\ &(1+b)\,t_j + \ell_j - b_j\tau\}. \end{aligned} \tag{5}$$

Then, we define the functions

$$\alpha_S(t) = t\cdot\prod_{j\in J}(1-a_j) + \sum_{j\in J}\Big((\ell_j + a_j\tau)\cdot\prod_{\substack{k\in J,\\ S^{-1}(k)>S^{-1}(j)}}(1-a_k)\Big), \tag{6}$$

$$\beta_S(t) = t\cdot\prod_{j\in J}(1+b_j) + \sum_{j\in J}\Big((\ell_j - b_j\tau)\cdot\prod_{\substack{k\in J,\\ S^{-1}(k)>S^{-1}(j)}}(1+b_k)\Big). \tag{7}$$

For common $a_j = a$ or $b_j = b$, respectively, they collapse to

$$\alpha_S(t) = t\,(1-a)^n + \sum_{i=1,\ldots,n}\left(\ell_{S(i)} + a\tau\right)(1-a)^{n-i}, \tag{8}$$

$$\beta_S(t) = t\,(1+b)^n + \sum_{i=1,\ldots,n}\left(\ell_{S(i)} - b\tau\right)(1+b)^{n-i} \tag{9}$$

with n jobs in sequence S.

We use the functions α_S and β_S to calculate the completion time of a given sequence S with a closed formula, where we distinguish three cases.

Lemma 1 *If a sequence S with n jobs starts at $t < \tau$ and there is $\alpha_S(t) \leq \tau + \ell_{S(n)}$, then it completes at $\alpha_S(t)$.*

Proof Let C be the completion time of S and its last job $S(n)$. We renumber the jobs such that $S = (1, \ldots, n)$. Then, let us show $\alpha_S(t_1) = C$ by induction: We begin with $n = 1$, starting job 1 at $t_1 = t \leq \tau$. By (5), job 1 completes at $C_1(t_1) = (1 - a_1)\, t_1 + \ell_1 + a_1\tau = \alpha_{(1)}(t_1)$ as stated. For $n > 1$, job j completes, if starting at $t_j = \alpha_{(1,\ldots,j-1)}(t_1) \leq \tau$, at $C_j(t_j) = \big(1 - a_j\big)\, t_j + \ell_j + a_j\tau$, and by induction $C_j(t_j) = \ell_j + a_j\tau + (1 - a_j) \cdot \alpha_{(1,\ldots,j-1)}(t_1) = \alpha_{(1,\ldots,j)}(t_1)$. □

Lemma 2 *If a sequence S starts at $t \geq \tau$, then it completes at $\beta_S(t)$.*

Proof Shown similar to Lemma 1 by induction from $t_1 = t \geq \tau$ to $\beta_S(t_1)$. □

Corollary 1 *If a sequence S with n jobs starts at $t < \tau$ and there is $\alpha_S(t) > \tau + \ell_{S(n)}$, then it completes at $\beta_{S_2}(\alpha_{S_1}(t))$ while the sequence is split into $S = S_1 S_2$ such that $\tau \leq \alpha_{S_1}(t) \leq \tau + \ell_\chi$ for the last job χ in S_1.*

The effect of changing a sequence's start time t can be observed by considering the derivatives of α_S and β_S.

Corollary 2 *Let a sequence S of a set of jobs J start at t.*

(a) *If $t \leq \tau$, then $1 \geq \frac{\mathrm{d}}{\mathrm{d}t}\alpha_S(t) = \prod_{j \in J} \big(1 - a_j\big) \geq 0$.*
(b) *If $t \geq \tau$, then $\frac{\mathrm{d}}{\mathrm{d}t}\beta_S(t) = \prod_{j \in J} \big(1 + b_j\big) \geq 1$.*

Thus, increasing a sequence's start time t does not decrease the sequence's completion time C. In other words, C does not increase if t is decreased.

Corollary 3 *Inserting idle time in front of any job does not decrease a sequence's makespan, for any fixed start time.*

Hence, it is not necessary to consider idle times in $\mathcal{P}$.

5 Polynomial cases of $\mathcal{P}$

In this section, we analyze properties of job (sub)sets of $\mathcal{P}$ instances, which lead to three polynomial cases of $\mathcal{P}$: if the ideal start time τ is early ($\tau \leq t_{\min}$), if the ideal start time is late ($\tau \geq \alpha_S(t_{\min}) - \ell_{S(n)}$ given a $\ell_j/a_j\searrow$-sorted sequence S with all n jobs), or if all basic processing times are zero.

5.1 Early ideal start time

If the start time t of a sequence is not less than the ideal start time τ (as in Lemma 2) and $\tau = 0$, then all jobs start at or after τ. This corresponds to the known monotonic scheduling problem with proportional penalty functions $\varpi_j(t) = b_j\, t$. Here, $\ell_j/b_j\nearrow$-sorted sequences yield the minimum makespan, which is observed in Shafransky (1978), Tanaev et al. (1984, 1994, chapter 3, section 1.2), Wajs (1986), Gupta and Gupta (1988), Browne and Yechiali (1990), and Gawiejnowicz and Pankowska (1995).

Please note that the special case with all-zero basic processing times is solved for any sequence S of a set of jobs J: its completion time $\beta_S(t) = t \cdot \prod_{j \in J}(1 + b_j)$ is independent of the order of jobs, which corresponds to the problem in Mosheiov (1994).

An instance with ideal start time $\tau \neq 0$ can be transformed to an instance with a zero ideal start time by performing a time-shift of $-\tau$. Then, the result for $\tau = 0$ applies as well.

Proposition 1 *A sequence S that is started at or after τ provides the minimum makespan if and only if S is $\ell_j/b_j\nearrow$-sorted.*

Corollary 4 *A $\mathcal{P}$ instance of n jobs with $t_{\min} \geq \tau$ is solved in $\mathcal{O}(n \log n)$ time by any $\ell_j/b_j\nearrow$-sorted sequence.*

5.2 Late ideal start time

Similarly, if $t \leq \tau = 0$, then a sequence might start each job before or at τ (like in Lemma 1). Such a case corresponds to the penalty function $\varpi_j(t) = -a_j\, t$, in which a $\ell_j/a_j\searrow$-sorted sequence provides a minimum makespan (Ho et al. 1993). It follows that in $\mathcal{P}$, if a $\ell_j/a_j\searrow$-sorted sequence starts each job (or equivalently, the last job) before or at τ, then it provides the minimum makespan.

In the special case of all-zero basic processing times and $t \leq \tau = 0$, any sequence S of a set of jobs J attains the same completion time $\alpha_S(t) = t \cdot \prod_{j \in J}(1 - a_j) \leq 0$.

Again, it is possible to convert an instance with $\tau \neq 0$ by a time-shift of $-\tau$ to an instance with a zero ideal start time.

Proposition 2 *If a sequence S starts its last job before or at τ, then S provides the minimum makespan if and only if S is $\ell_j/a_j\searrow$-sorted.*

Proposition 2 is only applicable to sequences that start each job at or before τ. But this may apply only for some of several existing $\ell_j/a_j\searrow$-sorted sequences. However, one can strengthen the sorting criterion such that for any two jobs j, k in sequence S at positions $S^{-1}(j) < S^{-1}(k)$, there is

$$\ell_j a_k > \ell_k a_j \;\vee\; \big(\ell_j a_k = \ell_k a_j \;\wedge\; \ell_j \leq \ell_k\big). \tag{10}$$

If there are multiple possible last jobs, this criterion assigns the one with the longest basic processing time to the last position. This minimizes the start time at the last position without changing the sequence's completion time.

Corollary 5 *For a $\mathcal{P}$ instance of n jobs with $t_{\min} \leq \tau$, a sequence S respecting (10) is constructed in $\mathcal{O}(n \log n)$ time. If $\alpha_S(t_{\min}) \leq \tau + \ell_{S(n)}$, then S is optimal.*

5.3 Zero basic processing times

The combination of the aforementioned special cases of all-zero basic processing times $\ell_j = 0$ is valid for any ideal start time τ and any start time $t_{\min}$. This generalizes the result on instances with $t_{\min} > \tau = 0$ in Mosheiov (1994).

Lemma 3 *If $\ell_j = 0$ for each job j in a set J, then any sequence of J provides the minimum makespan for any start time t, and completes at*

$$\tau + \max\left\{(t-\tau)\cdot\prod_{j\in J}(1-a_j),\ (t-\tau)\cdot\prod_{j\in J}(1+b_j)\right\}.$$

Corollary 6 *A $\mathcal{P}$ instance with $\ell_j = 0$ for each job j is solved by an arbitrary sequence; it is returned in $\mathcal{O}(n)$ time.*

6 Symmetry in optimal sequences for $\mathcal{P}$

Even if none of the described polynomial cases of $\mathcal{P}$ applies, they allow to observe a central property of optimal sequences: the symmetric sorting of the jobs before and after τ.

Proposition 3 *If a $\mathcal{P}$ sequence provides the minimum make span, then*

(a) *all jobs that complete before or at τ are $\ell_j/a_j\searrow$-sorted,*
(b) *all jobs that start at or after τ are $\ell_j/b_j\nearrow$-sorted.*

Proof Given a sequence S, split S into $S_1S_0S_2$ such that S_1 completes before or at τ, and S_2 starts at or after τ. Assume S_1 is not $\ell_j/a_j\searrow$-sorted. Then, the completion time of S_1 is not minimal: it decreases by re-ordering S_1 as a $\ell_j/a_j\searrow$-sorted sequence. Then, all jobs still complete (and start) before or at τ, and by Corollary 2, the ensuing sequence S_0S_2 starts earlier. Hence, S does not provide a minimum makespan if S_1 is not $\ell_j/a_j\searrow$-sorted (Proposition 2). An analogous observation holds for S_2: it has to be $\ell_j/b_j\nearrow$-sorted (Proposition 1). □

Remark 2 (Implications for $\mathcal{P}_{agreeable}$) According to Remark 1, a $\mathcal{P}_{agreeable}$ instance permits a job sequence that is $\ell_j/a_j\searrow$-sorted and, in reversed order, $\ell_j/b_j\nearrow$-sorted. Let $(1,\ldots,n)$ denote such a sequence by renumbering the given jobs accordingly. Now, consider an optimum sequence S and two jobs j,k with $1 \leq j < k \leq n$. If both jobs complete before or at τ, then their positions are $S^{-1}(j) < S^{-1}(k)$. If both start at or after τ, then $S^{-1}(j) > S^{-1}(k)$.

Remark 3 (On the choice of the straddler job) Please note that Proposition 3 excludes a statement about a potential straddler job. Indeed in optimal solutions, the straddler job (if it exists) is neither necessarily the job with the shortest basic processing time ℓ_j, nor with highest a_j and b_j value; see Figure 1 for an example of this. In the polynomial cases above however, the straddler job (if it exists) can be chosen according to the respective sorting criterion.

Remark 4 (On the existence of the straddler job) A straddler job exists in all optimal sequences if and only if $t_{\min} \leq \tau$ and a sequence sorted according to (10) that is started at $t_{\min}$ yields $C_{\max} \geq \tau$. If a straddler job exists in an optimal sequence, then it also exists in any other sequence of the same job set starting at t for $t_{\min} \leq t \leq \tau$ because its completion time is not less than the minimum completion time.

7 Computational complexity of $\mathcal{P}$, $\mathcal{P}_{agreeable}$, $\mathcal{P}_{common}$

In this section, a reduction from the NP-complete Even-Odd Partition problem shows that already the common slopes case $\mathcal{P}_{common}$ is NP-hard. Thus, the more general problems $\mathcal{P}$, $\mathcal{P}_{agreeable}$ are also NP-hard. Let us outline the proof, but beforehand, state the NP-complete Even-Odd Partition problem.

Definition 1 (*Even-Odd Partition* (Garey et al. 1988)) Given a set of $n = 2h$ natural numbers $X = \{x_1,\ldots,x_n\}$ where $x_{j-1} < x_j$ for $j = 2,\ldots,n$, does there exist a partition of X into subsets X_1 and X_2 such that $\sum_{x\in X_1} x = \sum_{x\in X_2} x$ and such that for each $i = 1,\ldots,h$, set X_1 (and hence X_2) contains exactly one of $\{x_{2i-1}, x_{2i}\}$?

NP-hardness of $\mathcal{P}_{common}$ is shown by proving the NP-hardness of its decision version, which asks, for a given rational-valued threshold Φ, if there exists a sequence S of the given jobs that is started at $t_{\min}$ and yields makespan $\phi \leq \Phi$.

The major steps of the proof are outlined as follows. The first trick is to choose slopes a and b such that assignment 'costs' are the same for the same ordinal *position* away from the ideal start time. Hence, a job's impact on the makespan no longer depends on its deviation from the ideal start time. Furthermore, the impact is the same on either side of τ within each Even-Odd pair. Then, the assignment decision of jobs to a side represents the partitioning problem. The second major step is to use a polynomial number of filler jobs that take up the time between the early jobs (if they complete too early) and τ, such that a No-instance is correctly recognized.

Theorem 1 $\mathcal{P}_{common}$ *is* NP*-hard.*

Proof Given an arbitrary instance of Even-Odd Partition, let us first define a corresponding instance of the decision version of $\mathcal{P}_{common}$. Then, we show it has a solution if and only if there exists a solution for the Even-Odd Partition instance.

Let $q = \frac{1}{2}\sum_{i\in X} x_i$. For the corresponding instance, we give the threshold $\Phi = 4q$, the common ideal start time $\tau = 0$, the global start time $t_{\min} = -q$, and the jobs $\{1, \ldots, 2n+1\}$, with $\ell_{n+j} = 0$ for $j = 1, \ldots, n$, with $\ell_{2n+1} = 2q$, and with $\ell_{2k-i} = x_{2k-i}\,(1+b)^{k-h-1}$ for $k = 1, \ldots, h$ and $i = 0, 1$. Then, $\ell_{j-1} < \ell_j$ for $j = 2, \ldots, n$, and $\ell_n < \ell_{2n+1}$. We may choose an arbitrary common slope a with $0 < a < 1$, and set $b = (1-a)^{-1} - 1$. Then, $b > 0$ and $(1+b) = (1-a)^{-1}$. It is feasible to conduct the reduction for any such slope values. However, we choose to simplify the presentation in the following by fixing the slopes to $a = 1/2$ and $b = 1$ such that $(1-a) = 1/2$ and $(1+b) = 2$.

Assume that a given corresponding instance possesses a sequence S with makespan $\phi \le \Phi$. Then, S either already has a certain format, or it can be aligned to this format in polynomial time without increasing the makespan as follows.

First, we assume that job $2n+1$ is the last job in S. Else, let t_{2n+1} denote this job's start time. If $t_{2n+1} \ge 0$, by sorting the jobs in S starting after 0 according to Proposition 1 in polynomial time, job $2n+1$ can take the last position without increasing the sequence's completion time. Otherwise if job $2n+1$ starts at $t_{2n+1} < 0$, then it completes after 0 because $\ell_{2n+1} > \tau$. In this case, repeatedly swap it with its successor job j and sort all other jobs starting before 0 according to Proposition 2. This does not increase the sequence's completion time either, because $\ell_j < \ell_{2n+1}$ and, with (5),

$$\begin{aligned} C_j(C_{2n+1}(t_{2n+1})) &= (t_{2n+1}/2 + \ell_{2n+1})\cdot 2 + \ell_j \\ &> \left(t_{2n+1}/2 + \ell_j\right)\cdot 2 + \ell_{2n+1} \\ &= C_{2n+1}(C_j(t_{2n+1})). \end{aligned}$$

Second, the jobs that complete before or at 0 can be ordered according to Proposition 2, while the jobs with zero basic processing time $\ell_j = 0$ are the last that complete before or at 0, in any order (Lemma 3). Analogously, let the jobs starting at or after 0 adhere to Proposition 1, while the jobs with $\ell_j = 0$ are the first, in any order (again according to Lemma 3). Then, Proposition 3 holds.

Now, sequence S can be narrowed down to attain either of the following two forms:

(i) Either, the sequence can be split into $S = S_1 S_0 S_2$ such that partial sequence S_1 contains the jobs completing before or at 0, while S_0 contains all the jobs that start and complete at 0, and S_2 contains the jobs starting at or after 0.

(ii) Otherwise, it can be split into $S = S_1 S_{01} S_\chi S_{02} S_2$ such that S_{01} and S_{02} together contain all the jobs with $\ell_j = 0$, while sequence $S_\chi = (\chi)$ consists of the straddler job χ that starts strictly before 0 and completes strictly after 0, partial sequence S_1 contains the jobs completing before or at 0, and S_2 the remaining jobs.

While form (i) is the desired form, let us rule out form (ii).

Consider sequences S_{01} and S_{02}. They contain all n jobs with zero basic processing time $\ell_j = 0$. Let v denote the number of jobs in S_{02}. Then, S_{01} contains $n-v$ jobs. Sequence S_{01} starts at some time $t < 0$. According to (8), it completes at $t/2^{n-v}$, which equals t_χ, the start time of the straddler job. Then, the straddler job χ completes at $C_\chi = t_\chi/2 + \ell_\chi$. Sequence S_{02} starts at C_χ, hence it completes according to (9) at $C = C_\chi \cdot 2^v$. Together, the completion time of $S_{01} S_\chi S_{02}$ starting at t is

$$C = \left(t/2^{n-v+1} + \ell_\chi\right)\cdot 2^v = \left(t\cdot 2^{v-n-1} + \ell_\chi\right)\cdot 2^v.$$

Its first and second derivatives are

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}v} C &= \left(2t\cdot 2^{v-n-1} + \ell_\chi\right)\cdot 2^v \cdot \ln 2, \\ \frac{\mathrm{d}^2}{\mathrm{d}v^2} C &= \left(4t\cdot 2^{v-n-1} + \ell_\chi\right)\cdot 2^v \cdot \ln^2 2. \end{aligned}$$

The completion time C has an extremum at a v with $\frac{\mathrm{d}}{\mathrm{d}v} C = 0$. As $t < 0$, the second derivative at the same v is $\frac{\mathrm{d}^2}{\mathrm{d}v^2} C < 0$. Therefore, this v value maximizes C. It follows that C is minimized either for $v = 0$ or for $v = n$ with $0 \le v \le n$. Therefore, the jobs with zero basic processing time can be moved altogether to either S_{01} or S_{02} without increasing C.

Assume that the zero basic processing time jobs $\{n+1, \ldots, 2n\}$ all start at or after 0. Hence, they are either in sequence S_0 for case (i), or they are in sequence S_{02} while S_{01} is empty for case (ii) in the following elaboration; the opposite case where they are in S_{01} while S_{02} is empty is performed analogously. With this assumption, iff S adheres to form (i), sequence S_1 completes at time 0, denoted by $\hat{C}$. Otherwise for form (ii), the straddler job χ completes at a time strictly after 0, denoted by $\hat{C}$ as well. Thus, $\hat{C} \ge 0$ in sequence S. Let $\hat{t}$ specify the start time of S_2. Hence, sequence S_0 in case (i), or S_{02} in case (ii), starts at $\hat{C}$ and completes at $\hat{t} = \hat{C}\cdot 2^n$.

Define h_1 as the number of jobs in S_1, and define $h_2 = n - h_1$. Given $\hat{C} \ge 0$, and the inverse of α_S in (8), which is $\alpha_S^{-1}(\tilde{C}) = \tilde{C}\,(1-a)^{-n} - \sum_{j\in J} \ell_{S(j)}\,(1-a)^{-j}$, then there is

$$t_{\min} = \alpha_{S_1}^{-1}(\hat{C}) = \hat{C}\cdot 2^{h_1} - \sum_{k=1,\ldots,h_1} \ell_{S_1(k)}\cdot 2^k. \tag{11}$$

Sequence S_2 starts at $\hat{t} = \hat{C} \cdot 2^n$. It consists of $h_2 + 1$ jobs. With the closed form (9), it completes at $C_{\max} = \beta_{S_2}(\hat{t})$. Then,

$$\begin{aligned} C_{\max} &= \hat{C} \cdot 2^{n+h_2+1} + \sum_{k=1,\ldots,h_2+1} \ell_{S_2(k)} \cdot 2^{h_2+1-k} \\ &= \hat{C} \cdot 2^{n+h_2+1} + \ell_{2n+1} + \sum_{k=1,\ldots,h_2} \ell_{S_2(h_2+1-k)} \cdot 2^k. \quad (12) \end{aligned}$$

Define

$$\begin{aligned} g_1(k) &= \begin{cases} \ell_{S_1(k)}, & 1 \le k \le h_1, \\ 0, & \text{else}, \end{cases} \\ g_2(k) &= \begin{cases} \ell_{S_2(h_2+1-k)}, & 1 \le k \le h_2, \\ 0, & \text{else}, \end{cases} \\ \bar{g} &= \sum_{k=1,\ldots,n} (g_1(k) + g_2(k)) \cdot 2^k, \\ d &= 2^{n+h_2+1} - 2^{h_1}. \end{aligned}$$

Because $h_1 \le n$ and $h_2 \ge 0$, we have $d > 0$. Then,

$$\begin{aligned} & \Phi \ge C_{\max} - t_{\min} \\ \iff & 4q \ge \hat{C} d + \ell_{2n+1} + \sum_{k=1,\ldots,h_1} \ell_{S_1(k)} \cdot 2^k \\ & \qquad + \sum_{k=1,\ldots,h_2} \ell_{S_2(h_2+1-k)} \cdot 2^k \\ \iff & 2q \ge \hat{C} d + \sum_{k=1,\ldots,n} (g_1(k) + g_2(k)) \cdot 2^k = \hat{C} d + \bar{g}. \end{aligned}$$

Sequence S satisfies the inequality since its makespan $\phi = C_{\max} - t_{\min} \le \Phi$. Let us show that the minimum of $\bar{g}$ is $2q$, which means that $\hat{C} = 0$ in the inequality.

For any $i, j \in \{1, 2\}$ such that $i \ne j$, if $g_i(k) = 0$ for some k while $g_j(k+1) > 0$, then sequence S does not provide a minimum for $\bar{g}$: it decreases by resequencing the jobs such that $g_i(k) > 0$ and $g_j(k+1) = 0$ because $2^k < 2^{k+1}$.

By this argument and as $h_1 + h_2 = 2h$, it follows that $h_1 = h_2 = h$.

Moreover, a minimum $\bar{g}$ has $g_i(k-1) \ge g_j(k)$ for $k = 2, \ldots, h$ and any $i, j = 1, 2$, because $2^{k-1} < 2^k$. This is the case for an optimal S as in Proposition 3.

Therefore, a minimum $\bar{g}$ requires $\{S_1(h+1-k),\ S_2(k)\} = \{2k-1,\ 2k\}$ (in any order) for $k = 1, \ldots, h$. Then,

$$\begin{aligned} \bar{g} &= \sum_{k=1,\ldots,h} \left(\ell_{S_1(h+1-k)} + \ell_{S_2(k)}\right) \cdot 2^{h+1-k} \\ &= \sum_{k=1,\ldots,h} \left(\ell_{2k-1} + \ell_{2k}\right) \cdot 2^{h+1-k} \\ &= \sum_{k=1,\ldots,h} \left(x_{2k-1} \cdot 2^{k-h-1} + x_{2k} \cdot 2^{k-h-1}\right) \cdot 2^{h+1-k} \\ &= \sum_{k=1,\ldots,h} x_{2k-1} + x_{2k} = 2q. \end{aligned}$$

By the arguments above, we have $\bar{g} = 2q$, $h_1 = h_2 = h$, and it follows that $C_{\max} - t_{\min} = \Phi$ and $\hat{C} = 0$. Sequence S thus adheres to form (i).

With $t_{\min} = -q$ and $\hat{C} = 0$, we transform (11) by using $\{S_1(h+1-k),\ S_2(k)\} = \{2k-1,\ 2k\}$ for $k = 1, \ldots, h$ to

$$\begin{aligned} q &= \sum_{k=1,\ldots,h} \ell_{S_1(k)} \cdot 2^k \\ &= \sum_{k=1,\ldots,h} \ell_{S_1(h+1-k)} \cdot 2^{h+1-k} \\ &= \sum_{k=1,\ldots,h} \left(x_{S_1(h+1-k)} \cdot 2^{k-h-1}\right) \cdot 2^{h+1-k} \\ &= \sum_{k=1,\ldots,h} x_{S_1(k)}. \end{aligned}$$

Applying similar steps for (12) with $C_{\max} = 3q$, we get $q = \sum_{k=1,\ldots,h} x_{S_2(k)}$. It follows the equality

$$\sum_{k=1,\ldots,h} x_{S_2(k)} = \sum_{k=1,\ldots,h} x_{S_1(k)}.$$

Concluding, sets $X_1 = \{x_{S_1(k)} \mid k = 1, \ldots, h\}$ and $X_2 = X \setminus X_1$ are a solution for the Even-Odd Partition instance.

Therefore, a solution to the Even-Odd Partition instance allows us to solve the corresponding $\mathcal{P}_{common}$ decision instance and vice versa. As the reduction is polynomial, it follows that $\mathcal{P}_{common}$ is NP-hard. □

Problem $\mathcal{P}_{common}$ is a special case of $\mathcal{P}_{agreeable}$, which in turn is a special case of $\mathcal{P}$.

Corollary 7 *$\mathcal{P}$ and $\mathcal{P}_{agreeable}$ are both* NP*-hard.*

The latter hardness result can also be inferred from the monotonic special cases in $\mathcal{P}_{agreeable}$ (where either $a_j = 0$ or $b_j = 0$, and the nonzero slopes are job-specific), which are NP-hard following the results in Kononov (1997), Kubiak and van de Velde (1998), and Cheng et al. (2003).

8 Dynamic programming algorithm for $\mathcal{P}_{agreeable}$

In this section, we describe a dynamic programming algorithm for $\mathcal{P}_{agreeable}$, and analyze its runtime. This algorithm is employed later (in Sect. 9) for constructing a fully polynomial time approximation scheme.

We explicitly exclude instances that already correspond to a polynomial case in Corollary 4 or Corollary 5 in the following consideration. Hence, we can assume that the straddler job exists (see Remark 4).

Denote by J the set of all given jobs. Let $n = |J|-1$ where $|J|$ is the number of jobs in J. Then, the following algorithm runs repeatedly, once for each possible straddler job $\chi \in J$. In each run, renumber the jobs to $\{1, \ldots, n, n+1\}$ such that $\chi = n+1$ and such that $\ell_j a_k \geq \ell_k a_j$ and $\ell_j b_k \geq \ell_k b_j$ for $1 \leq j < k \leq n$. Such a numbering exists (Remark 1), and implies that sequence $(1, \ldots, n)$ is $\ell_j/a_j \searrow$-sorted, and $(n, \ldots, 1)$ is $\ell_j/b_j \nearrow$-sorted. Please remember the according symmetry around τ in an optimal sequence (Remark 2).

The dynamic programming algorithm solving $\mathcal{P}_{agreeable}$ for a straddler job $\chi = n+1$ consists of n stages. Stage $j = 1, \ldots, n$ is represented by a set V_j of partial solutions. A partial solution can be imagined as a pair (S_1, S_2) of two partial sequences that respect the following invariant: S_1 and S_2 represent a partition of jobs $\{1, \ldots, j\}$ into sets A, B while

- sequence S_1 of job set A is $\ell_j/a_j \searrow$-sorted, to start at $t_{\min}$ and guaranteed to complete before τ, and
- sequence S_2 of job set B is $\ell_j/b_j \nearrow$-sorted, to start at τ.

In the j'th stage, job j is inserted into all partial solutions V_{j-1} of the preceding stage $j-1$. We consider two possible ways of inserting job j to the sequences, which respects the above invariant. First, inserting j as the last job in sequence S_1 unless this yields a completion time after τ. This has no effect on the start times of the other jobs in S_1. Second, inserting job j as the first job in S_2. Then, job j starts exactly at τ, which postpones all other jobs in S_2 by job j's processing time $p_j(\tau) = \ell_j$.

The dynamic program does, to save memory, not explicitly store the partial sequences S_1 and S_2. Instead, a partial solution is represented by a three-dimensional vector $[x, y, z]$ of nonnegative rational numbers, described as follows:

- The first component, x, denotes sequence S_1's completion time, hence, $x = \alpha_{S_1}(t_{\min})$, see Lemma 1.
- The y component describes the proportional increase of sequence S_2's makespan if increasing its start time $t \geq \tau$, hence, $y = \frac{\mathrm{d}}{\mathrm{d}t}\beta_{S_2}(t) = \frac{\mathrm{d}}{\mathrm{d}t}\beta_{S_2}(\tau)$, see Corollary 2(b).
- Lastly, z represents sequence S_2's makespan if starting it at τ, hence, $z = \beta_{S_2}(\tau) - \tau$, see Lemma 2.

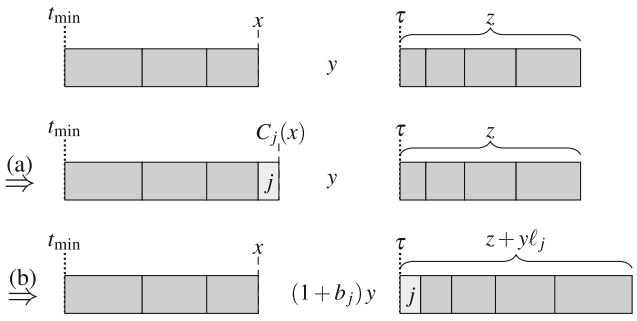

Fig. 4 The dynamic programming algorithm for $\mathcal{P}_{agreeable}$ with a certain straddler job χ stores a vector $[x, y, z]$ to represent a state, where x denotes the completion time of sequence S_1 starting at $t_{\min}$, z denotes the makespan of sequence S_2 starting at τ, and y denotes the derivative value of z on changing the start time of S_2. In each iteration $j = 1, \ldots, n$, at most two new vectors are generated from each state $[x, y, z]$: **(a)** vector $[C_j(x), y, z]$ that appends job j to x as long as $C_j(x) < \tau$, and **(b)** a vector $[x, (1+b_j)\, y, z + y\ell_j]$ that prepends job j to S_2 and modifies y accordingly. After the last iteration, the straddler job χ is appended to S_1, after which S_2 is then started, increasing its makespan accordingly

After stage n, the straddler job χ is appended to sequence S_1, after which S_2 continues. Figure 4 displays the partial solution of such an intermediate state, and shows the two successor states that emerge from adding the next job to either sequence S_1 or sequence S_2.

Algorithm 1 (Dynamic Programming for $\mathcal{P}_{agreeable}$ with straddler job χ)
Initialize state set

$$V_0 = \{[t_{\min},\ 1,\ 0]\}. \tag{13a}$$

For job $j = 1, \ldots, n$, generate state set

$$V_j = \big\{[C_j(x),\ y,\ z] \,\big|\, [x,\ y,\ z] \in V_{j-1},\ C_j(x) < \tau\big\} \tag{13b}$$

$$\cup\, \big\{\big[x,\ (1+b_j)\, y,\ z + y\,\ell_j\big] \,\big|\, [x,\ y,\ z] \in V_{j-1}\big\}. \tag{13c}$$

Return

$$C^{\chi}_{\max} = \min\big\{\tau + y\, \max\{C_\chi(x) - \tau,\ 0\} + z \,\big|\, [x,\ y,\ z] \in V_n\big\}. \tag{13d}$$

The resulting sequence $S = S_1 S_2$ is reconstructed in $\mathcal{O}(n)$ time by recording for stage $j = 1, \ldots, n$ and each state in V_j from which state in V_{j-1} it originates. With this information, one can determine a backwards path from the final state in V_n to the initial state in V_0. Then, the sequence is built by following the path from $j = 1$ to n. Begin with empty partial sequences S_1 and S_2. If the path's state in V_k was generated in (13b), append job k to S_1. If instead, it was generated in (13c), then prepend job k to S_2. In (13d), χ is appended to S_1, and S_2 is started at $\max\{C_\chi(x), \tau\}$. If the state was invalid

in the sense that job χ completes at $C_\chi(x) < \tau$, this inserts idle time before S_2 such that it starts at τ; then the result is dominated by a solution from running the algorithm for another straddler job.

Proposition 4 *For a $\mathcal{P}_{agreeable}$ instance, repeatedly running Algorithm 1 for each possible straddler job $\chi \in J$ returns the minimum makespan ϕ^*.*

Proof Given an instance of $\mathcal{P}_{agreeable}$, the algorithm is run as follows for each possible χ.

Consider stage $j = 1, \ldots, n$. In V_j, there is at least one vector for each possible subset of jobs $A \subseteq \{1, \ldots, j\}$ where each job $k \in A$ completes before τ, and $B = \{1, \ldots, j\} \setminus A$. Each vector $[x, y, z] \in V_j$ stems from a source vector $[x', y', z']$. Two cases are distinguished:

- If the vector is generated in (13b), job j is in set A. The value x' describes the start time of job j, completing at x. If $x' = t_{\min}$, job j is the first job in set A and it starts at the global start time $t_{\min}$. As $\ell_{j-1}a_j \geq \ell_j a_{j-1}$, the makespan x of the jobs in set A is minimum, see Proposition 2. The condition $C_j(x') < \tau$ ensures that j does not turn into the straddler job. As the set B is unchanged, $y = y'$ and $z = z'$ remain the same.
- Else, if the vector is generated in (13c), job j is instead in set B. For this, j is (for now) started at τ. Then, j completes at $C_j(\tau)$. If $z' = 0$, job j is the first job in set B. Then, $z = C_j(\tau) - \tau = \ell_j$. If $z' > 0$, job j is prepended to the jobs $B' = B \setminus \{j\}$. Then, they start later, by $C_j(\tau) - \tau = \ell_j$. As of Corollary 2(b), their completion time increases by $y \cdot \prod_{j \in B'} (1 + b_j)$. Each job that is inserted in set B multiplies the previous y by $(1 + b_j)$. Therefore, $y' = \prod_{j \in B'} (1 + b_j)$. Then, z expresses the sum of processing times of all jobs in set B when started at τ. Moreover, the jobs are sequenced as $S_B = (j, \ldots, \min B)$. Thus, $z = \beta_{S_B}(\tau) - \tau$. As $\ell_j \leq \ell_{j-1}$, this makespan is minimum for the jobs in set B if started at or after τ.

In the last step, the straddler job χ is appended to the early jobs in each source vector $[x', y', z']$.

For this, χ starts at time x', and completes at $x = C_\chi(x')$. To return a correct $C^\chi_{\max}$, two cases are treated in (13d):

Case $x \geq \tau$: Then, the jobs in set B start at x. In (13d), their completion time $\tau + z'$ is correctly increased by $(x - \tau) \cdot y'$, according to Corollary 2(b), with time difference $x - \tau$ and slope $y' = \prod_{j \in B} (1 + b_j)$. Therefore, the return value correctly calculates $C^\chi_{\max}$ corresponding to $[x', y', z']$.

Case $x < \tau$: In this case, idle time is inserted from x to τ. Then, the first job in set B, $k = \max B$, is scheduled at the common ideal start time: $t_k = \tau$. The resulting $C^\chi_{\max}$ in (13d) is dominated by $C^k_{\max}$ for k as the straddler job.

It is assumed that an optimal sequence has a straddler job, else the instance corresponds to a polynomial case in Sect. 5 for which the algorithm stops upfront. Therefore, the repeated execution of the algorithm to obtain $C^\chi_{\max}$ for each $\chi \in J$ yields $\phi^* = \min_{\chi \in J} C^\chi_{\max}$. □

The total number of states in Algorithm 1 is $\mathcal{O}(2^n)$, which corresponds to the number of branchings.

Corollary 8 *A $\mathcal{P}_{agreeable}$ instance with n jobs is solved by a n times repeated call of Algorithm 1 in $\mathcal{O}(n \cdot 2^n)$ time total.*

The runtime still is non-polynomial (i.e., not pseudopolynomial) if measured in terms of input length *and* values, or similarly, in terms of unary encoded input length.

Proposition 5 *Algorithm 1 is not pseudopolynomial.*

Proof The fundamental theorem of arithmetic states that any natural number greater than 1 can be expressed by a unique product of a nonempty multiset of prime numbers up to the order of the factors (Hardy and Wright 2008, chapter 1). Conversely, the product of any nonempty multiset of prime numbers is unique. Thus, all the 2^n distinct subsets of the set of the first n primes yield 2^n distinct products.

Let P_i for $i \geq 1$ denote the i'th prime number. Create a $\mathcal{P}_{agreeable}$ instance with $\tau = 0$, some straddler job, and an arbitrary number of jobs $\{1, \ldots, n\}$ where $\ell_j = 1$, $a_j = 0$, and $b_j = P_j - 1$ for $j = 1, \ldots, n$. Then, Algorithm 1 creates vectors where the y component corresponds to a product of a subset of the first n prime numbers. Hence, at least 2^n distinct values (and states) are created.

The sum of the first n primes is polynomial in n (see, e.g., Axler 2019). Respectively, a unary encoded input of the stated instance has a length that is polynomial in n, but Algorithm 1 remains exponential in unary encoded input length. Thus, Algorithm 1 is not pseudopolynomial. □

Since Algorithm 1 is not pseudopolynomial, it does not settle the question whether $\mathcal{P}_{agreeable}$ is NP-hard in the strong sense.

It is interesting to observe that despite this result, Algorithm 1 is suited for constructing an FPTAS, as it is shown below. This is unusual and counter-intuitive because commonly, an FPTAS is derived from a pseudopolynomial exact algorithm (Garey and Johnson 1979, p. 140).

9 Fully polynomial time approximation scheme for $\mathcal{P}_{agreeable}$

A fully polynomial time approximation scheme (FPTAS) is introduced for $\mathcal{P}_{agreeable}$ in this section.

An FPTAS is an algorithm that, given a problem's input and any approximation factor $\varepsilon \in (0, 1]$, runs in polynomial time of input length and $1/\varepsilon$ to return a solution with objective value $\phi^{\varepsilon} \leq (1+\varepsilon) \cdot \phi^*$, where ϕ^* denotes the minimum objective value.

The following FPTAS for $\mathcal{P}_{agreeable}$ is based on Algorithm 1. It is based on the idea of trimming-the-state-space as described in Ibarra and Kim (1975), combined with the interval partition technique described in Woeginger (2000). The latter technique defines

$$\Delta = 1 + \frac{\varepsilon}{2n}, \text{ and } h(x) = \Delta^{\lceil \log_\Delta x \rceil} \text{ for any real } x > 0,$$

where h intentionally satisfies $x/\Delta < h(x) \leq x \cdot \Delta$.

For an approximation factor $\varepsilon \in (0, 1]$ and corresponding Δ and h, let us define, similar to Algorithm 1, with the same preconditions and a given straddler job χ:

Algorithm 2 (FPTAS for $\mathcal{P}_{agreeable}$ with straddler job χ and ε)
Initialize

$$V_0^{\#} = \{[t_{\min},\ 1,\ 0]\}. \tag{14a}$$

For job $j = 1, \ldots, n$, generate state set

$$\tilde{V}_j^{\#} = \left\{ [C_j(x),\ y,\ z] \,\middle|\, [x,\ y,\ z] \in V_{j-1}^{\#},\ C_j(x) < \tau \right\} \tag{14b}$$

$$\cup \left\{ \left[x,\ \left(1+b_j\right) y,\ z + y\, \ell_j \right] \,\middle|\, [x,\ y,\ z] \in V_{j-1}^{\#} \right\} \tag{14c}$$

and trimmed state set

$$V_j^{\#} = \left\{ [\tilde{x}, \tilde{y}, \tilde{z}] \in \tilde{V}_j^{\#} \,\middle|\, \tilde{x} = x_j^{\min}(\tilde{y}, \tilde{z}) \right\}$$
$$\text{with } x_j^{\min}(\tilde{y}, \tilde{z}) = \min \Big\{ x \,\Big|\, [x,\ y,\ z] \in \tilde{V}_j^{\#},\ h(y) \leq h(\tilde{y}),\ h(z) = h(\tilde{z}) \Big\}. \tag{14d}$$

Return

$$C_{\max}^{\chi\varepsilon} = \min\left\{ \tau + y \max\{C_\chi(x) - \tau, 0\} + z \,\middle|\, [x,\ y,\ z] \in V_n^{\#} \right\}. \tag{14e}$$

The algorithm's approximation guarantee and its worst-case runtime is shown in the remainder of this section.

Lemma 4 *For $j = 0, \ldots, n$ and all vectors $[x, y, z] \in V_j$ of Algorithm 1, there exists a vector $[x^{\#}, y^{\#}, z^{\#}] \in V_j^{\#}$ in Algorithm 2 with*

$$x^{\#} \leq x, \tag{15a}$$
$$y^{\#} \leq y \cdot \Delta^j, \text{ and} \tag{15b}$$
$$z^{\#} \leq z \cdot \Delta^j. \tag{15c}$$

Proof Let us show the given hypothesis by forward-induction for $j = 0, \ldots, n$.

For $j = 0$, the trimmed set equals the original set: $V_0 = V_0^{\#} = \{[t_{\min}, 1, 0]\}$, as of (13a) and (14a). As $\Delta^0 = 1$, the hypothesis is shown.

For $j = 1, \ldots, n$, there are two cases, corresponding to (14b) and (14c):

- The first case applies if $x < \tau$, i.e., the job j is appended to S_1 such that it completes before τ. Consider vector $[x', y', z'] \in V_{j-1}$ where $C_j(x') = x$, $y' = y$, $z' = z$ as generated in (14b).
  Then, by induction, the corresponding vector $[x'^{\#}, y'^{\#}, z'^{\#}] \in V_{j-1}^{\#}$ with $x'^{\#} \leq x'$, $y'^{\#} \leq y' \cdot \Delta^{j-1}$, and $z'^{\#} \leq z' \cdot \Delta^{j-1}$ exists. Also, the condition $x'^{\#} \leq \tau$ is satisfied. Furthermore, the algorithm created $[\tilde{x}', \tilde{y}', \tilde{z}'] = [C_j(x'^{\#}), y'^{\#}, z'^{\#}] \in \tilde{V}_j^{\#}$, see (14b). Although, after the trimming operation in (14d) this vector may not be in $V_j^{\#}$, there exists a vector $[x^{\#}, y^{\#}, z^{\#}] \in V_j^{\#}$ with $x^{\#} \leq \tilde{x}'$, $h(y^{\#}) \leq h(\tilde{y}')$, and $h(z^{\#}) = h(\tilde{z}')$ (thus, $y^{\#} \leq \tilde{y}' \cdot \Delta$ and $z^{\#} \leq \tilde{z}' \cdot \Delta$).
  Let us show the induction hypothesis for this vector. Remember that $C_j(t)$ is an nondecreasing function. Thus, $x^{\#} \leq \tilde{x}' = C_j(x'^{\#}) \leq C_j(x') = x$ for (15a). As $y^{\#} \leq \tilde{y}' \cdot \Delta = y'^{\#} \cdot \Delta \leq y' \cdot \Delta^j = y \cdot \Delta^j$, (15b) is satisfied. Inequality $z^{\#} \leq \tilde{z}' \cdot \Delta = z'^{\#} \cdot \Delta \leq z' \cdot \Delta^j = z \cdot \Delta^j$ satisfies (15c).
- In the second case corresponding to (14c), consider vector $[x', y', z'] \in V_{j-1}$ where $x' = x$, $\left(1+b_j\right) y' = y$, and $z' + y'\, \ell_j = y$.
  By induction hypothesis, the corresponding vector $[x'^{\#}, y'^{\#}, z'^{\#}] \in V_{j-1}^{\#}$ with $x'^{\#} \leq x'$, $y'^{\#} = y' \cdot \Delta^{j-1}$, and $z'^{\#} \leq z' \cdot \Delta^{j-1}$ exists. Then, the algorithm created a corresponding vector $[\tilde{x}', \tilde{y}', \tilde{z}'] = [x'^{\#}, \left(1+b_j\right) y'^{\#}, z'^{\#} + y'^{\#}\, \ell_j] \in \tilde{V}_j^{\#}$ in (14c). Even though, by trimming, this vector may not be in set $V_j^{\#}$, there must exist some vector $[x^{\#}, y^{\#}, z^{\#}] \in V_j^{\#}$ with $x^{\#} \leq \tilde{x}'$, $h(y^{\#}) \leq h(\tilde{y}')$, and $h(z^{\#}) = h(\tilde{z}')$ (thus $y^{\#} \leq \tilde{y}' \cdot \Delta$ and $z^{\#} \leq \tilde{z}' \cdot \Delta$).
  Let us show the induction hypothesis. For (15a): $x^{\#} \leq \tilde{x}' = x'^{\#} \leq x' = x$. As $y^{\#} \leq \tilde{y}' \cdot \Delta = \left(1+b_j\right) \cdot y'^{\#} \cdot \Delta \leq \left(1+b_j\right) \cdot \left(y' \cdot \Delta^{j-1}\right) \cdot \Delta = y \cdot \Delta^j$, (15b) is satisfied. Lastly, (15c) is satisfied because

$$\begin{aligned} z^{\#} &\leq \tilde{z}' \cdot \Delta = \left(z'^{\#} + y'^{\#}\, \ell_j\right) \cdot \Delta \\ &\leq \left(z' \cdot \Delta^{j-1} + y'\, \ell_j \cdot \Delta^{j-1}\right) \cdot \Delta = \left(z' + y'\, \ell_j\right) \cdot \Delta^j \\ &= z \cdot \Delta^j. \end{aligned}$$

□

Lemma 5 *For $0 < \varepsilon \leq 1$ and a $\mathcal{P}_{agreeable}$ instance with straddler job χ and minimum makespan ϕ^*, Algorithm 2 yields $C^{\chi\varepsilon}_{\max}$ such that $\phi^\varepsilon = C^{\chi\varepsilon}_{\max} - t_{\min} \leq (1+\varepsilon)\,\phi^*$.*

Proof Let χ be the straddler job and $[x, y, z] \in V_n$ be the vector that corresponds to $\phi^* = C^{\chi}_{\max} - t_{\min}$ in Algorithm 1. Then, $C^{\chi}_{\max} = \tau + y\left(C_\chi(x) - \tau\right) + z$, with $C_\chi(x) \geq \tau$. By Lemma 4, there exists a vector $[x^\#, y^\#, z^\#] \in V^\#_n$ with $x^\# \leq x$, $y^\# \leq y \cdot \Delta^n$, and $z^\# \leq z \cdot \Delta^n$. Then,

$$\begin{aligned} C^{\chi\varepsilon}_{\max} &= \tau + y^\# \cdot \max\left\{C_\chi(x^\#) - \tau,\, 0\right\} + z^\# \\ &\leq \tau + y \cdot \Delta^n \cdot \max\left\{C_\chi(x) - \tau,\, 0\right\} + z \cdot \Delta^n \\ &= \tau + \left(y \cdot \max\left\{C_\chi(x) - \tau,\, 0\right\} + z\right) \cdot \Delta^n \\ &= \tau + \left(C^{\chi}_{\max} - \tau\right) \cdot \Delta^n \\ &= C^{\chi}_{\max} \cdot \Delta^n + \tau \cdot \left(1 - \Delta^n\right). \end{aligned}$$

Because $1 - \Delta^n \leq 0$ and $t_{\min} \leq \tau$, there is $t_{\min} \cdot (1 - \Delta^n) \geq \tau \cdot (1 - \Delta^n)$, thus

$$\begin{aligned} C^{\chi\varepsilon}_{\max} - t_{\min} &\leq C^{\chi}_{\max} \cdot \Delta^n + \tau \cdot \left(1 - \Delta^n\right) - t_{\min} \\ &\leq C^{\chi}_{\max} \cdot \Delta^n + t_{\min} \cdot \left(1 - \Delta^n\right) - t_{\min} \\ &= \left(C^{\chi}_{\max} - t_{\min}\right) \cdot \Delta^n. \end{aligned}$$

Thus, together with Proposition 4, $\phi^\varepsilon \leq \phi^* \cdot \Delta^n$. A known inequality is $(1 + \delta/n)^n \leq 1 + 2\delta$ for $0 \leq \delta \leq 1$ (Woeginger 2000, Proposition 3.1). Setting $\delta = \varepsilon/2$, it follows $\Delta^n \leq 1 + \varepsilon$. Thus, $\phi^\varepsilon \leq (1+\varepsilon)\,\phi^*$. □

For a worst-case runtime analysis, let us bound the number of states in each stage to a polynomial number. This uses the respective logarithm of

$$\ell_{ratio} = \frac{\max\left\{\ell_j \,\middle|\, k = 1, \ldots, n\right\}}{\min\left\{\ell_j > 0 \,\middle|\, j = 1, \ldots, n\right\}},$$
$$b_{\max} = \max\left\{b_j \,\middle|\, j = 1, \ldots, n\right\}.$$

Lemma 6 *For $0 < \varepsilon \leq 1$ and stage $j = 0, \ldots, n$, the number $|V^\#_j|$ of states is in $\mathcal{O}(n^3 \cdot \log(1 + b_{\max}) \cdot (\log \max\{\ell_{\text{ratio}}, 1/b_{\max}\} + n \log(1 + b_{\max}))/\varepsilon^2)$.*

Proof Starting with $|V^\#_0| = 1$, let us analyze state set $V^\#_j$ for $j = 1, \ldots, n$ in the following. Consider a vector $[x, y, z] \in V^\#_j$. For each y and z value, there is one x value. Thus, $|V^\#_j|$ is bounded by the product of the number of possible y and z values, which is bounded in the following.

Let $\ell^{(j)}_{\max} = \max\{\ell_k \,|\, k = 1, \ldots, j\}$, $\ell^{(j)}_{\min} = \min\{\ell_k > 0 \,|\, k = 1, \ldots, j\}$, and $b^{(j)}_{\max} = \max\{b_k \,|\, k = 1, \ldots, j\}$. Then, the y value is bounded by

$$1 \leq y \leq \prod_{k=1,\ldots,j} (1 + b_k) \leq \left(1 + b^{(j)}_{\max}\right)^j =: Y_j.$$

The z value represents the makespan of the sequence that starts at the ideal start time. For $z > 0$, it is bounded by

$$\begin{aligned} \ell_{\min} \leq z &\leq \sum_{j'=1,\ldots,j} \ell_{j'} \prod_{k=j'+1,\ldots,j} (1 + b_k) \\ &\leq \ell^{(j)}_{\max} \frac{\left(1 + b^{(j)}_{\max}\right)^j - 1}{b^{(j)}_{\max}} =: Z_j. \end{aligned}$$

The trimming step in (14d) ensures that there is at most a single y and z value for the same $h(y)$ and $h(z)$ value, respectively. Moreover, the rounded values are bounded by $1 \leq h(y) \leq h(Y_j)$ and by $h(\ell^{(j)}_{\min}) \leq h(z) \leq h(Z_j)$ for $z > 0$.

It follows from the definition of h that the number of distinct $h(y)$ values is at most

$$\begin{aligned} \log_\Delta Y_j - \log_\Delta 1 &= \log Y_j \cdot \ln 2 \,/\, \ln \Delta \\ &\leq \left(1 + \frac{2n}{\varepsilon}\right) \cdot \log Y_j \\ &= \left(1 + \frac{2n}{\varepsilon}\right) \cdot j \cdot \log\left(1 + b^{(j)}_{\max}\right), \end{aligned}$$

which uses the inequality $\ln \Delta \geq (\Delta - 1)/\Delta$ (Woeginger 2000, Proposition 3.1). Similarly, the number of distinct $h(z)$ values for $z > 0$ is at most

$$\begin{aligned} &\log_\Delta Z_j - \log_\Delta \ell^{(j)}_{\min} = \log\left(Z_j \Big/ \ell^{(j)}_{\min}\right) \cdot \ln 2 \,/\, \ln \Delta \\ &< \left(1 + \frac{2n}{\varepsilon}\right) \left(\log \frac{\ell^{(j)}_{\max}}{\ell^{(j)}_{\min}} + \log \frac{1}{b^{(j)}_{\max}} + j \cdot \log\left(1 + b^{(j)}_{\max}\right)\right). \end{aligned}$$

In summary, there are at most $\mathcal{O}\left(n^2 \log(1 + b_{\max})/\varepsilon\right)$ distinct y values, and there are at most $\mathcal{O}(n \cdot (\log \ell_{ratio} + \log(1/b_{\max}) + n \log(1 + b_{\max}))/\varepsilon)$ distinct z values. Both upper bounds are polynomial in input length in a binary encoding. This includes rational numbers because they can be encoded as a division of two integers. The product of both bounds yields $\mathcal{O}(n^3 \cdot \log(1 + b_{\max}) \cdot (\log \ell_{ratio} + \log(1/b_{\max}) + n \log(1 + b_{\max}))/\varepsilon^2)$, which is not more than the upper bound stated above. □

Algorithm 2 is repeatedly started for each possible straddler job, hence $n + 1$ times, and has at most n stages, each with a polynomial number of states (Lemma 6). Furthermore, as of Lemma 5, the resulting makespan is guaranteed to be at most $(1+\varepsilon)$ times the minimum makespan ϕ^*. This leads to the conclusion.

Theorem 2 *For a $\mathcal{P}_{agreeable}$ instance of n jobs with minimum makespan ϕ^* and an approximation factor $0 < \varepsilon \leq 1$, the n times repeated call of Algorithm 2 returns a solution with makespan $\phi^\varepsilon \leq (1+\varepsilon) \cdot \phi^*$ in $\mathcal{O}(n^5 \cdot \log(1 + b_{\max}) \cdot (\log \max\{\ell_{\text{ratio}}, 1/b_{\max}\} + n \log(1 + b_{\max}))/\varepsilon^2))$ time total.*

This runtime is polynomial, hence Algorithm 2 is an FPTAS for $\mathcal{P}_{agreeable}$. With a common slope $b_j = b$, the y component of a state can attain at most j different values in each stage $j \in \{1, \ldots, n\}$. Then, there are only $\mathcal{O}(n^2 \cdot (\log \ell_{ratio} + \log(1/b) + n \log(1+b))/\varepsilon)$ states in each of the $\mathcal{O}(n^2)$ stages.

Corollary 9 *For instances with a common slope $b_j = b$ for each given job j, the runtime given in Theorem 2 is reduced to $\mathcal{O}(n^4 \cdot (\log \max\{\ell_{\max}, 1/b\} + n \log(1+b))/\varepsilon)$.*

In the monotonic case $b_j = 0$, the y component equals 1 at all times, and z is bounded by the sum of basic processing times. Then, the number of distinct z values is bounded by $\mathcal{O}(\ell_{ratio} \cdot n)$, and there are only $\mathcal{O}(n \cdot \log(\ell_{ratio} n)/\varepsilon)$ states per stage.

Corollary 10 *For instances with $b_j = 0$ for each given job j, the runtime in Theorem 2 is reduced to $\mathcal{O}(n^3 \cdot \log(\ell_{\text{ratio}} n)/\varepsilon)$.*

If each job j has the same $\ell_j = \ell$, then $\ell_{ratio} = 1$. If in addition $\max\{b_j, 1/b_j\}$ is smaller than a constant, then the FPTAS finishes in strongly polynomial $\mathcal{O}(n^5/\varepsilon^2)$ time. For $b_j = 0$, it takes only $\mathcal{O}(n^3 \log n/\varepsilon)$ time, although this particular case is even easier to solve optimally by sorting the jobs with respect to nondecreasing a_j values (Cheng et al. 2003).

Remark 5 (Implications on $\mathcal{P}_{agreeable}$'s computational complexity) Garey and Johnson (1978, Theorem 1) state that if, for all instances, the optimal objective value of a minimization problem is upper bounded by a polynomial in input length and input values, then the existence of an FPTAS implies that a pseudopolynomial algorithm exists. In $\mathcal{P}_{agreeable}$, the makespan can be exponential in input length and input values: e.g., for $t_{\min} = \tau$, $b_j = b$, and $\ell_j \geq 1$ for $j = 1, \ldots, n$, the makespan is not less than $(1+b)^{n-1}$. Therefore, the existence of an FPTAS for $\mathcal{P}_{agreeable}$ does, in this particular case, *not* imply the existence of a pseudopolynomial algorithm. Hence, it remains open whether a pseudopolynomial algorithm exists for $\mathcal{P}_{agreeable}$, and whether $\mathcal{P}_{agreeable}$ is NP-hard in the strong sense.

Acknowledgements Let me thank the associate editor and the anonymous referees very much for their constructive comments and thorough reviews, thank Joanna Berlińska, Peter Fúsek, Stanisław Gawiejnowicz, Nir Halman, Jan-Hendrik Lorenz, Bartłomiej Przybylski for helpful discussions, and especially thank Uwe Schöning at the Institute of Theoretical Computer Science at Ulm University in Ulm, Germany for the generous support of the majority of this paper.

Funding Open access funding provided by ZHAW Zurich University of Applied Sciences.

References

Agnetis, A., Billaut, J. C., Gawiejnowicz, S., Pacciarelli, D., & Soukhal, A. (2014). *Multiagent scheduling*. Berlin, Heidelberg: Springer. https://doi.org/10.1007/978-3-642-41880-8.

Alidaee, B., & Womer, N. K. (1999). Scheduling with time dependent processing times: Review and extensions. *The Journal of the Operational Research Society*, *50*(7), 711–720. https://doi.org/10.2307/3010325.

Axler, C. (2019). On the sum of the first n prime numbers. *Journal de Théorie des Nombres de Bordeaux*, *31*(2), 293–311. https://doi.org/10.5802/jtnb.1081.

Browne, S., & Yechiali, U. (1990). Scheduling deteriorating jobs on a single processor. *Operations Research*, *38*(3), 495–498. https://doi.org/10.1287/opre.38.3.495.

Błażewicz, J., Ecker, K. H., Pesch, E., Schmidt, G., Sterna, M., & Węglarz, J. (2019). *Handbook on scheduling: From theory to practice*. Cham: Springer. https://doi.org/10.1007/978-3-319-99849-7.

Cai, J. Y., Cai, P., & Zhu, Y. (1998). On a scheduling problem of time deteriorating jobs. *Journal of Complexity*, *14*(2), 190–209. https://doi.org/10.1006/jcom.1998.0473.

Cheng, T. C. E., Ding, Q., Kovalyov, M. Y., Bachman, A., & Janiak, A. (2003). Scheduling jobs with piecewise linear decreasing processing times. *Naval Research Logistics*, *50*(6), 531–554. https://doi.org/10.1002/nav.10073.

Farahani, M. H., & Hosseini, L. (2013). Minimizing cycle time in single machine scheduling with start time-dependent processing times. *The International Journal of Advanced Manufacturing Technology*, *64*(9), 1479–1486. https://doi.org/10.1007/s00170-012-4116-1.

Garey, M. R., & Johnson, D. S. (1978). "Strong" NP-completeness results—Motivation, examples, and implications. *Journal of the ACM*, *25*(3), 499–508. https://doi.org/10.1145/322077.322090.

Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability–A guide to the theory of NP-completeness. Series of books in the mathematical sciences*. San Francisco: W.H. Freeman.

Garey, M. R., Tarjan, R. E., & Wilfong, G. T. (1988). One-processor scheduling with symmetric earliness and tardiness penalties. *Mathematics of Operations Research*, *13*(2), 330–348. https://doi.org/10.2307/3689828.

Gawiejnowicz, S. (2008). *Time-dependent scheduling. Monographs in theoretical computer science*. Berlin, Heidelberg: Springer. https://doi.org/10.1007/978-3-540-69446-5.

Gawiejnowicz, S. (2020a). *Models and algorithms of time-dependent scheduling. Monographs in theoretical computer science* (2nd ed.). Berlin, Heidelberg: Springer. https://doi.org/10.1007/978-3-662-59362-2.

Gawiejnowicz, S. (2020b). A review of four decades of time-dependent scheduling: Main results, new topics, and open problems. *Journal of Scheduling*,. https://doi.org/10.1007/s10951-019-00630-w.

Gawiejnowicz, S., & Pankowska, L. (1995). Scheduling jobs with varying processing times. *Information Processing Letters*, *54*(3), 175–178. https://doi.org/10.1016/0020-0190(95)00009-2.

Gordon, V. S., Potts, C. N., Strusevich, V. A., & Whitehead, J. D. (2008). Single machine scheduling models with deterioration and learning: Handling precedence constraints via priority generation. *Journal of Scheduling*, *11*(5), 357–370. https://doi.org/10.1007/s10951-008-0064-x.

Gupta, J. N. D., & Gupta, S. K. (1988). Single facility scheduling with nonlinear processing times. *Computers & Industrial Engineering*, *14*(4), 387–393. https://doi.org/10.1016/0360-8352(88)90041-1.

Hall, N. G., Kubiak, W., & 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. https://doi.org/10.1287/opre.39.5.847.

Halman, N. (2019). A technical note: Fully polynomial time approximation schemes for minimizing the makespan of deteriorating jobs with nonlinear processing times. *Journal of Scheduling*,. https://doi.org/10.1007/s10951-019-00616-8.

Hardy, G. H., & Wright, E. M. (2008). *An introduction to the theory of numbers* (6th ed.). Oxford, New York: Oxford University Press.

Ho, K. I. J., Leung, J. Y. T., & Wei, W. D. (1993). Complexity of scheduling tasks with time-dependent execution times. *Information Processing Letters*, *48*(6), 315–320. https://doi.org/10.1016/0020-0190(93)90175-9.

Hoogeveen, J. A., & van de Velde, S. L. (1991). Scheduling around a small common due date. *European Journal of Operational Research*, *55*(2), 237–242. https://doi.org/10.1016/0377-2217(91)90228-N.

Ibarra, O. H., & Kim, C. E. (1975). Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the ACM*, *22*(4), 463–468. https://doi.org/10.1145/321906.321909.

Jaehn, F., & Sedding, H. A. (2016). Scheduling with time-dependent discrepancy times. *Journal of Scheduling*, *19*(6), 737–757. https://doi.org/10.1007/s10951-016-0472-2.

Ji, M., & Cheng, T. C. E. (2007). An FPTAS for scheduling jobs with piecewise linear decreasing processing times to minimize makespan. *Information Processing Letters*, *102*(2–3), 41–47. https://doi.org/10.1016/j.ipl.2006.11.014.

Kacem, I. (2010). Fully polynomial time approximation scheme for the total weighted tardiness minimization with a common due date. *Discrete Applied Mathematics*, *158*(9), 1035–1040. Erratum: Kianfar and Moleshi (2013) https://doi.org/10.1016/j.dam.2010.01.013.

Kawase, Y., Makino, K., & Seimi, K. (2018). Optimal composition ordering problems for piecewise linear functions. *Algorithmica*, *80*(7), 2134–2159. https://doi.org/10.1007/s00453-017-0397-y.

Kellerer, H., & Strusevich, V. A. (2010). Minimizing total weighted earliness-tardiness on a single machine around a small common due date: An FPTAS using quadratic knapsack. *International Journal of Foundations of Computer Science*, *21*(3), 357–383. https://doi.org/10.1142/S0129054110007301.

Kianfar, K., & Moslehi, G. (2013). A note on "Fully polynomial time approximation scheme for the total weighted tardiness minimization with a common due date". *Discrete Applied Mathematics*, *161*(13–14), 2205–2206. https://doi.org/10.1016/j.dam.2013.02.026.

Klampfl, E., Gusikhin, O., & Rossi, G. (2006). Optimization of workcell layouts in a mixed-model assembly line environment. *International Journal of Flexible Manufacturing Systems*, *17*(4), 277–299. https://doi.org/10.1007/s10696-006-9029-6.

Kononov, A. V. (1997). On schedules of a single machine jobs with processing times nonlinear in time. *Discrete Analysis and Operational Research*, *391*, 109–122. https://doi.org/10.1007/978-94-011-5678-3_10.

Kononov, A. V. (1998). Problems in scheduling theory on a single machine with job durations proportional to an arbitrary function. *Diskretnyĭ Analiz i Issledovanie Operatsiĭ*, *5*(3), 17–37.

Kovalyov, M. Y., & Kubiak, W. (1998). A fully polynomial approximation scheme for minimizing makespan of deteriorating jobs. *Journal of Heuristics*, *3*(4), 287–297. https://doi.org/10.1023/A:1009626427432.

Kovalyov, M. Y., & Kubiak, W. (2012). A generic FPTAS for partition type optimisation problems. *International Journal of Planning and Scheduling*, *1*(3), 209. https://doi.org/10.1504/IJPS.2012.050127.

Kubiak, W., & van de Velde, S. L. (1998). Scheduling deteriorating jobs to minimize makespan. *Naval Research Logistics*, *45*(5), 511–523. https://doi.org/10.1002/(SICI)1520-6750(199808)45:5<511::AID-NAV5>3.0.CO;2-6.

Lawler, E. L., & Moore, J. M. (1969). A functional equation and its application to resource allocation and sequencing problems. *Management Science*, *16*(1), 77–84. https://doi.org/10.1287/mnsc.16.1.77.

Melnikov, O. I., & Shafransky, Y. M. (1979). Parametric problem in scheduling theory. *Cybernetics*, *15*(3), 352–357. https://doi.org/10.1007/BF01075095.

Mosheiov, G. (1994). Scheduling jobs under simple linear deterioration. *Computers & Operations Research*, *21*(6), 653–659. https://doi.org/10.1016/0305-0548(94)90080-9.

Scholl, A., Boysen, N., & Fliedner, M. (2013). The assembly line balancing and scheduling problem with sequence-dependent setup times: Problem extension, model formulation and efficient heuristics. *OR Spectrum*, *35*(1), 291–320. https://doi.org/10.1007/s00291-011-0265-0.

Sedding, H. A. (2017). Scheduling of time-dependent asymmetric non-monotonic processing times permits an FPTAS. In: *Proceedings of the 15th Cologne Twente Workshop on Graphs and Combinatorial Optimization*, University of Cologne, Cologne, Germany (pp. 135-138).

Sedding, H. A. (2018a). On the complexity of scheduling start time dependent asymmetric convex processing times. In: *Proceedings of the 16th International Conference on Project Management and Scheduling*, Universitá di Roma "Tor Vergata", Rome, Italy (pp. 209–212).

Sedding, H. A. (2018b). Scheduling non-monotonous convex piecewise-linear time-dependent processing times. In: *Proceedings of the 2nd International Workshop on Dynamic Scheduling Problems*, Adam Mickiewicz University, Poznań, Poland (pp. 79–84).

Sedding, H. A. (2020a). An FPTAS for scheduling with piecewise-linear nonmonotonic convex time-dependent processing times and job-specific agreeable slopes. In: *Proceedings of the 17th International Conference on Project Management and Scheduling*, Toulouse Business School, Toulouse, France, postponed to 2021.

Sedding, H. A. (2020b). Line side placement for shorter assembly line worker paths. *IISE Transactions*, *52*(2), 181–198. https://doi.org/10.1080/24725854.2018.1508929.

Sedding, H. A. (2020c). *Time-dependent path scheduling: Algorithmic minimization of walking time at the moving assembly line*. Wiesbaden: Springer Vieweg. https://doi.org/10.1007/978-3-658-28415-2.

Sedding, H. A., & Jaehn, F. (2014). Single machine scheduling with non-monotonic piecewise linear time dependent processing times. In: *Proceedings of the 14th International Conference on Project Management and Scheduling*, TUM School of Management, Munich, Germany (pp. 222–225).

Shafransky, Y. M. (1978). On optimal ordering in deterministic systems with tree-like partial serving order. *Proceedings of the Academy of Sciences of BSSR, Physics and Mathematics Series*, *1978*(2), 120.

Smith, W. E. (1956). Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, *3*(1–2), 59–66. https://doi.org/10.1002/nav.3800030106.

Strusevich, V. A., & Rustogi, K. (2017). *Scheduling with time-changing effects and rate-modifying activities*. Cham: Springer. https://doi.org/10.1007/978-3-319-39574-6.

Tanaev, V. S., Gordon, V. S., & Shafransky, Y. M. (1984). *Scheduling theory: Single-stage systems*. Moscow: Nauka.

Tanaev, V. S., Gordon, V. S., & Shafransky, Y. M. (1994). *Scheduling theory: Single-stage systems*. Dordrecht: Springer. https://doi.org/10.1007/978-94-011-1190-4.

Wajs, W. (1986). Polynomial algorithm for dynamic sequencing problem. *Archiwum Automatyki i Telemechaniki*, *31*(3), 209–213.

Woeginger, G. J. (2000). When does a dynamic programming formulation guarantee the existence of a fully polynomial time approximation scheme (FPTAS)? *INFORMS Journal on Computing*, *12*(1), 57–74. https://doi.org/10.1287/ijoc.12.1.57.11901.

Yuan, J. (1992). The NP-hardness of the single machine common due date weighted tardiness problem. *Systems Science and Mathematical Sciences*, *5*(4), 328–333.