Parameterized Complexity of Scheduling Chains of Jobs with Delays

Hans L. Bodlaender

Department of Information and Computing Sciences, Utrecht University, The Netherlands
H.L.Bodlaender@uu.nl

Marieke van der Wegen

Department of Information and Computing Sciences, Utrecht University, The Netherlands
Mathematical Institute, Utrecht University, The Netherlands
M.vanderWegen@uu.nl

──── **Abstract** ────

In this paper, we consider the parameterized complexity of the following scheduling problem. We must schedule a number of jobs on m machines, where each job has unit length, and the graph of precedence constraints consists of a set of chains. Each precedence constraint is labelled with an integer that denotes the exact (or minimum) delay between the jobs. We study different cases; delays can be given in unary and in binary, and the case that we have a single machine is discussed separately. We consider the complexity of this problem parameterized by the number of chains, and by the thickness of the instance, which is the maximum number of chains whose intervals between release date and deadline overlap.

We show that this scheduling problem with exact delays in unary is $W[t]$-hard for all t, when parameterized by the thickness, even when we have a single machine ($m = 1$). When parameterized by the number of chains, this problem is $W[1]$-complete when we have a single or a constant number of machines, and $W[2]$-complete when the number of machines is a variable. The problem with minimum delays, given in unary, parameterized by the number of chains (and as a simple corollary, also when parameterized by the thickness) is $W[1]$-hard for a single or a constant number of machines, and $W[2]$-hard when the number of machines is variable.

With a dynamic programming algorithm, one can show membership in XP for exact and minimum delays in unary, for any number of machines, when parameterized by thickness or number of chains. For a single machine, with exact delays in binary, parameterized by the number of chains, membership in XP can be shown with branching and solving a system of difference constraints. For all other cases for delays in binary, membership in XP is open.

2012 ACM Subject Classification Computing methodologies → Planning and scheduling; Theory of computation → Fixed parameter tractability; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Scheduling, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.4

Related Version Missing proofs can be found at the full version of this paper, which can be found at arXiv [4] (https://arxiv.org/abs/2007.09023).

Acknowledgements We would like to thank Sukanya Pandey for helpful discussions.

1 Introduction

In this paper, we study a problem in the field of parameterized complexity of scheduling problems. Here, we look at scheduling jobs with precedence constraints with exact or minimum delays, and assume that all jobs have unit length. We study one of the simplest types of precedence constraint graphs: we assume that the precedences form a collection of disjoint chains. Chains have a release date and deadline. It is not hard to see (by a simple

reduction from 3-Partition) that this problem is NP-hard, even when all delays are 0. In this paper, we study the parameterized complexity of the problem, and look at two different parameters: the number of chains, and the thickness of the instance – that is, the maximum number of chains that have overlapping intervals from release time to deadline. We look at different variants: a constraint gives an exact bound or a lower bound on the delay between successive jobs; we can have one, a constant, or a variable number of machines, and the delays can be encoded in unary or binary. If delays are given in unary, then each of the studied variants belongs to XP, and is hard for $W[1]$ (or classes higher in the W-hierarchy). For one variation (see below), we also show membership in XP when delays are given in binary. We call the studied problems Chain Scheduling with Exact Delays and Chain Scheduling with Minimum Delays, for details see Section 2.

1.1 Related literature

Looking at variants of scheduling problems with special attention to parameters (like the number of available machines) is a common approach in the rich field of study of scheduling problems. Studying such parameterizations using techniques and terminology from the field of parameterized algorithms and complexity was pioneered in 1995 [3], but recently receives growing attention, e.g. [2, 11, 14].

The scheduling of chains of jobs (without delays) was studied by Woeginger [17] and Bruckner [5], who gave respectively a 2-approximation algorithm, and a linear time algorithm for two machines. General precedence graphs with delays between jobs were studied already in 1992 by Wikum et al. [16]; this was followed by a large body of literature, studying different variations and approaches, including theoretical and experimental studies.

Cieliebak et al. [6] considered scheduling jobs with release dates and deadlines, under several parameterizations, one being the height, which is similar to the parameter called thickness in this paper – the height of an instance is the maximum number of jobs that have mutually overlapping intervals from release date to deadline. Several additional, and stronger results on this problem were obtained by van Bevern et al. [15], including a proof that for each fixed looseness $\lambda > 0$, the problem to schedule jobs with release dates and deadlines on a given number of machines is $W[1]$-hard, where λ is the maximum ratio between the difference of release date and deadline and the duration of jobs. This latter result is related to ours: where we have $W[t]$-hardness for all t or $W[2]$-hardness for chains of jobs with exact or minimum delays, respectively, parameterized by thickness, the result by van Bevern et al. [15] gives $W[1]$-hardness for single jobs, i.e., chains of size one.

A special case of chains of jobs is the case where we have coupled jobs, i.e., each chain consists of two jobs. Bessy and Giroudeau [2] consider the parameterized complexity of scheduled jobs with compatibility constraints, where jobs can be executed in the idle time of another pair of coupled jobs when they are compatible.

A survey of parameterized algorithms for scheduling with a number of interesting open problems was given by Mnich and van Bevern [13].

1.2 Our results

In this paper, we give a number of hardness results, which are summarized in Table 1. All variants are already hard when the (exact or minimum) delays are given in unary. We also give the following algorithmic results:

- With a dynamic programming algorithm, one can show that the Chain Scheduling with Exact Delays and Chain Scheduling with Minimum Delays belong to XP, when delays are given in unary, and parameterized by either thickness or number of

■ **Table 1** Hardness results for different variants of the problem. Exact and minimum delays are given in unary. (*) = $W[1]$-hardness follows from [15].

	parameter	exact delays	minimum delays
Single machine	thickness	$W[t]$-hard for all t	$W[1]$-hard
	chains	$W[1]$-complete	$W[1]$-hard
Constant number of machines	thickness	$W[t]$-hard for all t	$W[1]$-hard
	chains	$W[1]$-complete	$W[1]$-hard
Variable number of machines	thickness	$W[t]$-hard for all t (*)	$W[2]$-hard (*)
	chains	$W[2]$-complete	$W[2]$-hard

chains, for any number of machines. Our algorithm is similar to an algorithm by Cieliebak et al. [6], for a related problem – they consider the problem where each job has a release date and deadline, and show that this problem belongs to XP when we use the maximum number of jobs whose intervals between release date and deadline mutually overlap.

▬ Combining branching with solving a set of difference constraints shows XP-membership of CHAIN SCHEDULING WITH EXACT DELAYS when parameterized by the number of chains, for the case of one machine, when delays are given in binary.

For all other cases, the membership in XP when delays are given in binary is open.

1.3 Organization of this paper

In Section 2, we give a number of preliminary definitions. Section 3 gives hardness proofs for CHAIN SCHEDULING WITH EXACT DELAYS when parameterized by the thickness. The complexity of CHAIN SCHEDULING WITH EXACT DELAYS parameterized by the number of chains is established in Section 4; a relatively simple modification then gives hardness for the corresponding problems with minimum delays. Section 5 gives our algorithmic results (membership in XP). Some conclusions are given in Section 6.

2 Preliminaries

We first describe the problems we study in more details. We have a number of identical machines m. In the paper, we study separately the cases that we have a single machine ($m = 1$), the number of machines is some fixed constant, or the number of machines is variable.

On these machines, we must schedule n jobs. Each job has unit length. On the set of jobs, we have a collection of precedence constraints. Each precedence constraint is an ordered pair of jobs (j, j'): it tells that job j' cannot be started before job j is completed. We say that j is a *direct predecessor* of j', and j is a *predecessor* of j' if there is a directed path from j to j' in the graph formed by the precedence constraints; j' then is a *successor* of j.

The precedence constraints have associated with them a *delay*, denoted $l_{j,j'}$: each delay is a non-negative integer. We study two variations of the problem: exact delays and minimum delays. If we consider exact (resp. minimum) delays, then if a constraint (j, j') has delay $l_{j,j'}$, then job j' must be started exactly (resp. at least) $l_{j,j'}$ time steps after job j was finished. That is: when job j starts at time t, then job j' starts at time exactly (resp. at least) $t + l_{j,j'} + 1$. (Note that jobs run directly after each other, only if the delay is 0.) It is allowed to schedule a job on a different machine than its predecessor – thus, we do not need to specify on which machine a job is running, but only ensure that at each time step, the number of scheduled jobs is at most the number of available machines.

In this paper, we consider the case that the graph of the precedence constraints consists of a set of chains. I.e., each job has at most one direct predecessor and at most one direct successor. Chains are the maximal sets of jobs that are predecessors or successors of each other.

Each chain C has a *release date* r_C and a deadline d_C. We have that the first job in the chain cannot start before time r_C and the last job in the chain should be completed at or before time d_C.

In this paper, we consider the following two parameterizations of the problem. The first is the number of chains, denoted by c. The second is the *thickness*, denoted by τ, defined as follows. We say that two chains *overlap*, when their intervals $[r_C, d_C)$ have a non-empty intersection. We define the thickness τ to be the maximum size of a collection of chains that mutually overlap. That is, for any time t, there are at most τ chains C for which we have that $r_C \leq t$ and $d_C > t$.

CHAIN SCHEDULING WITH EXACT DELAYS is the problem where we are given as input the set of jobs with chains of precedence constraints, delays for each precedence constraint, release dates and deadlines of chains, and number of machines, and ask whether there exists a schedule that fulfills all the demands: at each time step, the number of jobs scheduled is at most the number of machines; jobs in a chain are not scheduled before the release date or after the deadline, and for each precedence constraint (j, j') the delay between j and j' is exactly $l_{j,j'}$.

As said, we study several variants of this problem: delays can be given in unary or binary, the number of machines can be 1, fixed or variable, and we can parameterize by the number of chains or by thickness. If we require that the stated delays are lower bounds, we obtain the CHAIN SCHEDULING WITH MINIMUM DELAYS problem: here, when we have a precedence constraint (j, j') with delay $l_{j,j'}$, we must have that job j' starts at least $l_{j,j'}$ time steps after job j is finished.

For the $W[t]$-hardness proofs, we use fpt-reductions from the following version of the SATISFIABILITY problem. A Boolean formula is said to be *t-normalized*, if it is the conjunction of the disjunction of the conjunction of ... of literals, with t alternations of AND's and OR's.

The following parameterized problem was considered by Downey and Fellows [8].

WEIGHTED t-NORMALIZED SATISFIABILITY
Given: A t-normalized Boolean formula F and a positive integer $k \in \mathbb{N}$.
Parameter: k
Question: Can F be satisfied by setting exactly k variables to true?

▶ **Theorem 2.1** (Downey and Fellows [8, 9])**.** *For every $t \geq 2$,* WEIGHTED t-NORMALIZED SATISFIABILITY *is $W[t]$-complete.*

For the $W[1]$- and $W[2]$-completeness results, we use reductions from INDEPENDENT SET and DOMINATING SET. It is know that INDEPENDENT SET is $W[1]$-complete [9] and DOMINATING SET is $W[2]$-complete [8].

3   Parameterization by thickness

In this section, we look at the CHAIN SCHEDULING WITH EXACT DELAYS problem, when parameterized by the thickness τ. We will show, for several variations, that the problem is hard for the class $W[t]$, for all integers t.

3.1 Parallel machines

We consider the version with m parallel machines, where m is part of the input. The delays are assumed to be exact.

We will give a reduction from WEIGHTED t-NORMALIZED SATISFIABLITY. Assume we have a t-normalized Boolean formula F and an integer k. Let t' be the number of "levels" of disjunction. Notice that $t' = \lfloor t/2 \rfloor$. We assume the variables of F to be $x_0, \ldots, x_{n-1}$.

We make an instance of the CHAIN SCHEDULING WITH EXACT DELAYS problem, with $m = k + t'$ machines and thickness $\tau = 2k + t'$.

An *element* of the formula is either a literal, or a disjunction or conjunction of smaller elements. We will first assign to each element F' an *interval size* $s(F')$, and then we assign to each element F' an integer interval with length $s(F')$.

The *interval size* of a literal (i.e., a formula of the form x_i or $\neg x_i$) is $2n$. The interval size of a conjunction is the sum of the size of the terms, i.e., $s(F_1 \wedge F_2 \wedge \cdots \wedge F_q) = \sum_{i=1}^{q} s(F_i)$. For each disjunction F' of q terms, its interval size is $2q + 1$ times the maximum size of its terms: define $s_{\max}(F') = \max_{1 \le i \le q} s(F_i)$, and then $s(F_1 \vee F_2 \vee \cdots \vee F_q) = (2q+1) \cdot s_{\max}(F')$.

To each element F' of F we assign an integer interval $[\ell(F'), r(F')]$ with $s(F') = r(F') - \ell(F')$. We will do this top-down: first we assign an interval to F, then we define a subinterval for every term of F, etc.

To F, we assign the interval $[n, n + s(F)]$. To elements of a conjunction and disjunction, we assign subintervals of the intervals assigned to the conjunction of disjunction, in such a way that these intervals have the same nesting as the elements in the formula.

Consider an element F' that is the conjunction $F_1 \wedge F_2 \wedge \cdots \wedge F_q$. Then assign F_1 the interval $[\ell(F'), \ell(F') + s(F_1)]$; F_2 the interval $[\ell(F') + s(F_1), \ell(F') + s(F_1) + s(F_2)]$, etc. I.e., F_i is assigned the interval $[\ell(F') + \sum_{j=1}^{i-1} s(F_j), \ell(F') + \sum_{j=1}^{i} s(F_j)]$.

Suppose an element F' is the disjunction $F_1 \vee F_2 \vee \cdots \vee F_q$. The construction is similar to that of conjunctions, but now we assign each term the same length interval and keep unused intervals between the terms. Recall that $s_{\max}(F') = \max_{1 \le i \le q} s(F_i)$. Assign to F_i the interval $[\ell(F') + (2i - 1) \cdot s_{\max}(F'), \ell(F') + 2i \cdot s_{max}(F')]$.

Note the nesting of intervals, and that we assigned to each element an interval equal to its size. Also note that we can compute all intervals and sizes in polynomial time in the size of the input instance.

We now can describe the jobs, precedence constraints, and release dates and deadlines.

For each i, $1 \le i \le k$, we start a chain c_i. Each of those chains starts with a job and then a delay of $n - 1$. The first job of the chain is released at time 0. We will add jobs and specify delays between jobs in the chain such that the total processing time including the delay times is $n + s(F) + 1$. Set the deadline of those chains to $2n + s(F)$, so that the first job can start at times $0, 1, \ldots, n - 1$.

These chains reflect the variables that are set to true; more precisely, when the first job of one of the chains starts at a time i, then this corresponds to setting x_i to true. We call these the *true variable chains*.

To prevent two chains selecting the same variable, we add $m - 1$ chains, each with n jobs with delay 0, release date 0 and deadline n. We call those chains *fill chains*. Those chains have to be scheduled from time 0 until time n. This implies that at each time $0, 1, \ldots, n - 1$ at most one other job can be scheduled, thus at most one true variable chain starts. Hence, the true variable chains select exactly k variable to be set to true.

We will now extend the true variable chains. Consider the timesteps in the interval $[n, n + s(F)]$ from left to right.
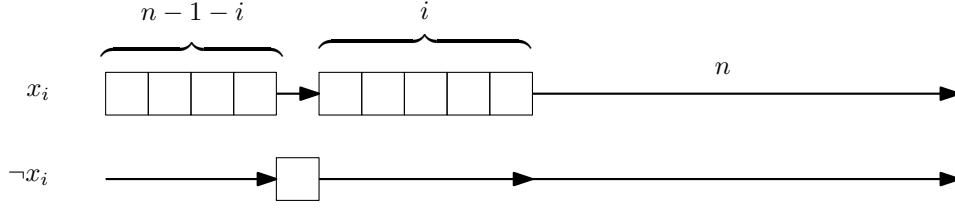
Figure 1 The variable gadgets.

- For each timestep that we encounter that is not part of an interval that corresponds to a literal, we add a delay of 1 at the end of the chain.
- For each interval $[\ell(F'), r(F')]$ that corresponds to a positive literal x_i, we add the following gadget to the chain: $n - 1 - i$ jobs with delay 0, then a delay of 1, then i jobs with delay 0 and then a delay of n. (See Figure 1.) Notice that no job is scheduled from $\ell(F') + n - 1$ until $\ell(F') + n$ if the chain starts at time i, and there is a job scheduled at this time otherwise.
- For each element F' of F that is a negative literal $\neg x_i$, we make the following gadget: a delay of $n - 1 - i$, a job, then a delay of i, and then a delay of n. (See Figure 1.) Notice that a job is scheduled from $\ell(F') + n - 1$ until $\ell(F') + n$ if the chain starts at time i, and there is no job scheduled at this time otherwise.

Add one job at the end of the chain. Notice that the total processing time of those chains is indeed $n + s(F) + 1$.

To check whether variables are true, we add some chains that consist of a single job. We call those chains *variable check chains*.

- For each element F' of F that consists of a single positive literal (i.e., is of the form x_i), we make a chain with one job, that is released at time $\ell(F') + n - 1$ and has deadline $\ell(F') + n$.
- For each element F' of F that consists of a single negative literal (i.e., is of the form $\neg x_i$), we make k chains with one job, release date $\ell(F') + n - 1$ and deadline $\ell(F') + n$.

The intuition behind this construction is as follows: suppose that we have k machines. For each element F' of F that is of the form x_i, there is one job scheduled from $\ell(F') + n - 1$ until $\ell(F') + n$. So for at least one of the true variable chains, we need that no job of this chain to be scheduled from $\ell(F') + n - 1$ until $\ell(F') + n$. This means that one of the true variable chains starts at time i. For each element F' of F that is of the form $\neg x_i$, there are k jobs scheduled from $\ell(F') + n - 1$ until $\ell(F') + n$. So for none of the true variable chains we can schedule a job of this chain from $\ell(F') + n - 1$ until $\ell(F') + n$. This means that none of the true variable chains starts at time i. The other t' machines take care of the disjunctions.

For each element $F' = F_1 \vee F_2 \vee \cdots \vee F_q$ of F that is a disjunction, we make one chain. This chain has $3 \cdot s_{\max}(F')$ jobs with delay 0. The chain will be released at time $\ell(F')$ and has deadline $r(F')$. We call those chains *disjunction chains*. Notice that for every element F' of F that is a literal, there are exactly t' disjunction chains that overlap the interval $[l(F'), r(F')]$, that is, there are exactly t' disjunction chains C with release time at most $l(F')$ and deadline at least $r(F')$.

We now have specified all jobs and the machines they run on. Note that the thickness of this construction is at most $2k + t'$.

▶ **Example 3.1.** Figure 2 shows an example of this construction for the formula $(x_0 \vee \neg x_1 \vee x_2) \wedge (\neg x_0 \vee x_1)$ and $k = 1$. The intervals that correspond to the literals are indicated. The interval that corresponds to the first disjunction $x_0 \vee \neg x_1 \vee x_2$ is $[3, 45]$, and the interval
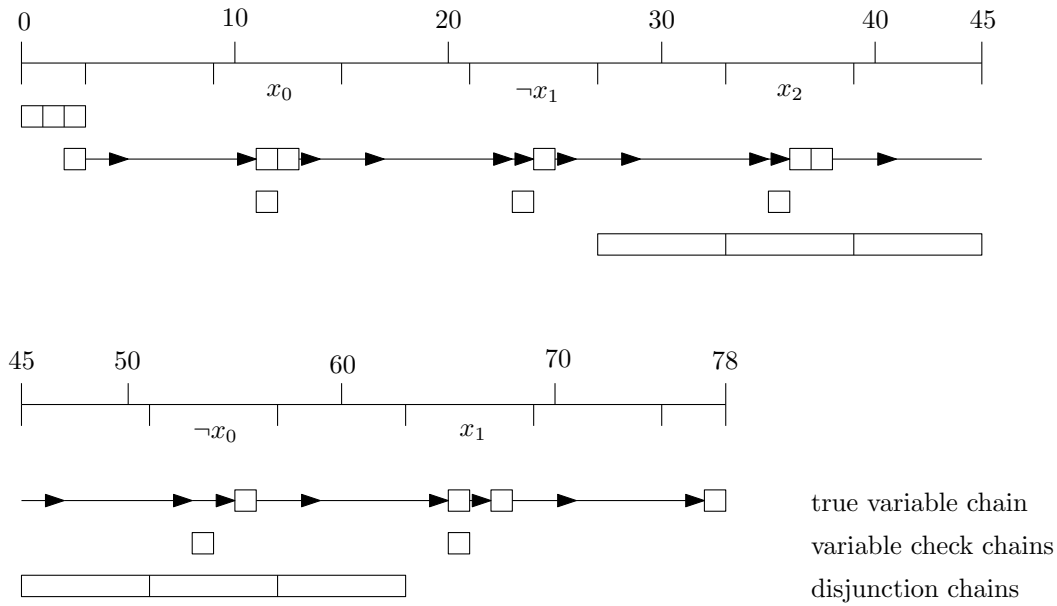
Figure 2 An example of the construction of Section 3 for the formula $(x_0 \vee \neg x_1 \vee x_2) \wedge (\neg x_0 \vee x_1)$ and $k = 1$. A feasible solution is shown which corresponds to setting x_2 to true.

$[45, 75]$ is assigned to the second disjunction $\neg x_0 \vee x_1$. The figure shows a feasible solution of the scheduling problem, that corresponds to setting x_2 to true, and x_0 and x_1 to false. The term x_2 satisfies the first disjunction, and the term $\neg x_0$ satisfies the second disjunction.

▷ **Claim 3.2.** If F is satisfiable by setting exactly k variables to true, then the constructed instance of the CHAIN SCHEDULING WITH EXACT DELAYS problem has a solution.

Proof. Suppose F is satisfiable by making variables $x_{i_1}, \ldots, x_{i_k}$ true. For each j, with $1 \leq j \leq k$, we let one true variable chain start at time i_j.

First we introduce a notion *satisfying*, intuitively, this will be the elements that make F true. We define this top-down. First we call F satisfying. For each element F' of F: for a satisfying conjunction, all its terms are satisfying, for a satisfying disjunction, at least one of its terms is satisfied, we fix one of them and call it satisfying.

For each disjunction $F' = F_1 \vee \cdots \vee F_q$, consider the disjunction chain C associated with this element. If F' is not satisfying, then start this chain C arbitrarily, say at its release time. If F' is satisfying, let F_j be its term that is satisfying. Now, start this chain C at time $\ell(F') + (2j - 2)s_{\max}(F')$, where $s_{\max}(F')$ is again the maximum over the terms F_i of their interval size $s(F_i)$.

To check that this is a feasible schedule, we have to check that we never use more than m machines. For most timesteps this is straightforward, see [4] for the details. The interesting timesteps are the timesteps from $l(F') + n - 1$ to $l(F') + n$ for some satisfying literal, since at those times a variable check chain and t' disjunction chains are scheduled. Since F is true and F' is satisfying, we know that the literal F' is set to true (resp. false). It follows that the corresponding true variable chain has no job starting at time $l(F') + n - 1$. Feasiblity follows, for details see the full version [4]. ◁

▷ **Claim 3.3 (See [4]).** Suppose the CHAIN SCHEDULING WITH EXACT DELAYS problem has a solution, then F can be satisfied by setting exactly k variables to true.

We now have shown:

▶ **Theorem 3.4.** *The* CHAIN SCHEDULING WITH EXACT DELAYS *problem, parameterized by the thickness* τ*, is* $W[t]$*-hard for all* $t \in \mathbb{N}$*.*

3.2   Single machine

Again, assume the delays are exact. We now show that the problem stays hard when there is only one machine.

▶ **Theorem 3.5.** *The* CHAIN SCHEDULING WITH EXACT DELAYS *problem, parameterized by the thickness* τ*, is* $W[t]$*-hard for all* $t \in \mathbb{N}$*, when only* 1 *machine is available.*

Let τ be the thickness of the original instance. Notice that we may assume that $\tau \geq m$ without loss of generality. The main idea of the reduction is to replace each time step on m machines by τ time steps on a single machine. Every chain will be assigned a number $i \in \{0, 1, \ldots, \tau - 1\}$ and its jobs will be scheduled at times $i \pmod{\tau}$, this will make sure that at every timestep only one job is scheduled. For every interval $[i\tau, (i + 1)\tau]$, we will have $\tau - m$ chains that have one job; this ensures that in that interval at most m jobs of an original chain are scheduled. We now proceed with the formal description.

We reduce from the case with m machines (Theorem 3.4). Suppose we have an instance with m machines. For every interval $[i\tau, (i + 1)\tau]$, we add $\tau - m$ additional chains, with a single job, release date $i\tau$ and deadline $(i + 1)\tau$. We call those chains *extra*.

We copy the chains from the given instance, except that:
- If a chain has release date α, then it now has release date $\alpha \cdot \tau$.
- If a chain has deadline β, then it now has deadline $\beta \cdot \tau$.
- Every delay d is replaced by a delay $\tau d + \tau - 1$.

We call these chains *regular*.

▷ **Claim 3.6.**   Suppose we have a solution for the constructed instance with one machine. Then we have a solution for the original instance with m machines.

Proof. For each regular chain, let it start at time $\lfloor t/\tau \rfloor$, when its copy in the constructed instance starts at time t. This implies that for every job in the chain it will start at time $\lfloor t/\tau \rfloor$, when its copy in the constructed instance starts at time t. We know that for each time interval $[t\tau, (t + 1)\tau]$, $\tau - m$ steps are used for extra jobs, so m time steps are available for jobs of regular chains. Thus, at every time step t in the original instance, at most m jobs are scheduled.   ◁

▷ **Claim 3.7.**   Suppose we have a solution for the original instance with m machines. Then we have a solution for the constructed instance with 1 machine.

Proof. We start with assigning a number $0, 1, \ldots, \tau - 1$ to every chain such that for every time step all the chains that overlap this time step have different number. We denote this assignment by c. We can do this as follows: go through time $0, 1, 2, \ldots$, and every time a chain is released, assign a number that is currently unused, if a deadline passes, the number of the corresponding chain becomes available again. We can do this with τ numbers, since by definition for every timestep there are at most τ chains that overlap this timestep.

For every regular chain C, let C start at time $t\tau + c(C)$, where t is the starting time of the corresponding original chain. Then chains of thickness number $c(C)$ only have jobs starting at times t with $t \equiv c(C) \pmod{\tau}$. For every time t, all jobs that are scheduled to start

at time t in the original instance, will now be scheduled in the interval $[t\tau, (t+1)\tau]$ in the constructed instance. Those jobs are in different chains and those chains are assigned different numbers. Thus those jobs are scheduled at different times in the constructed instance.

In the original instance, at each time t at most m chains have a job scheduled to start at t, so, in the constructed instance, for each interval $[t\tau, (t+1)\tau]$, at most m regular chains have a job scheduled in this interval. Thus, we can schedule the $\tau - m$ extra chains in this time interval at the time steps where no job of a regular chain is scheduled. $\triangleleft$

As the reduction can be carried out in polynomial time, Theorem 3.5 follows from the reduction and Theorem 3.4.

3.3 Constant number of parallel machines

We can easily reduce the single machine instance to an instance with a constant number m of parallel machines. Let T be the maximum deadline of all chains. Introduce $m - 1$ new chains with T jobs each and 0 delays. The $m - 1$ new machines will be processing those $m - 1$ new chains, while the original machine processes the original chains. We conclude the following result.

▶ **Theorem 3.8.** *The* Chain Scheduling with Exact Delays *problem with a fixed number of machines, parameterized by the thickness τ, is $W[t]$-hard for all $t \in \mathbb{N}$.*

3.4 Minimum delays

The proof above seems not to be modifiable to the Chain Scheduling with Minimum Delays problem. In Section 4.4, we show that Chain Scheduling with Minimum Delays, parameterized by the number of chains is $W[1]$-hard when $m = 1$, and $W[2]$-hard when the number of machines m is variable. As the thickness is at most the number of chains, it follows that Chain Scheduling with Minimum Delays, parameterized by the thickness is $W[1]$-hard for one machine, and $W[2]$-hard for a variable number of machines. Membership in $W[1]$ or $W[2]$ is open, however.

4 Parameterization by the number of chains

We now give the complexity results when we use the number of chains as the parameter. In Sections 4.1, 4.2, and 4.3, we consider Chain Scheduling with Exact Delays, with the number of machines respectively 1, a constant, or variable. In Section 4.4, we consider Chain Scheduling with Exact Delays.

4.1 Single machine

In this section, we consider the variant where the number of chains c is a parameter, and there is one machine available. We assume that the delays are exact.

▶ **Theorem 4.1.** *The* Chain Scheduling with Exact Delays *problem, parameterized by the number of chains c is $W[1]$-complete, when there is one machine.*

Theorem 4.1 is proven by two reductions: from and to Independent Set with standard parameterization.

▶ **Lemma 4.2.** *The* Chain Scheduling with Exact Delays *problem with one machine, parameterized by the number of chains c, is $W[1]$-hard.*
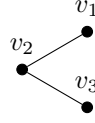
■ **Figure 3** An example graph G. If we pick $p = 3$, the corresponding Golomb ruler is $\{0, 7, 13\}$.

Proof. A set of integers S is said to be a *Golomb ruler* if all differences $a - b$ of two elements $a, b \in S$ are unique, that is, $s_1 - s_2 \neq s_3 - s_4$ for $s_1, s_2, s_3, s_4 \in S$ with $s_1 \neq s_2$ and $s_3 \neq s_4$.

Erdös and Turán [10] gave the following explicit construction of a Golomb ruler. Let $p > 2$ be a prime number. Then the set $\{2pk + (k^2 \bmod p) \mid k \in \{0, 1, \ldots, p - 1\}\}$ is a Golomb ruler with p elements. We can build a Golomb ruler of size n in $O(n\sqrt{n})$ time: with help of the Sieve of Eratosthenes, we find a prime number p between n and $2n$ (such a number always exist, by the classic postulate of Bertrand (see [1])), and then follow the Erdös-Turán-construction with this value of p, and take the first n elements of this set. Notice that the elements in this set are smaller than $4n^2$.

Suppose we have an input of INDEPENDENT SET $G = (V, E)$ and k. Assume $V = \{v_1, \ldots, v_n\}$ and let m be the number of edges. First, build a Golomb ruler S^n of size n. Denote the elements by $s_1 \leq s_2 \leq \cdots \leq s_n$. Notice that $s_1 = 0$. Write $c_0 = s_n + 1$.

We will construct an instance of CHAIN SCHEDULING WITH EXACT DELAYS.

We will make $k + 1$ chains. We call one chain the *start time forcing chain*, the other k chains are the *vertex selection chains*.

The start time forcing chain has release date 1, deadline c_0 and total execution time (including delays) $c_0 - 1$. I.e., it must start at time 1. The chain will have a job starting at every time in $[1, c_0 - 1]$ except the times $s_2, s_3, \ldots, s_n$, there, it has a delay of 1.

The vertex selection chains have release date 0. They have deadline $c_0 + T - 1$ and total execution time T, where $T = (m \cdot k(k - 1))(2c_0 + 1) + 1$. So, they can start at times $0, 1, \ldots, c_0 - 1$. The vertex selection chains start with a job and then a delay of $c_0 - 1$. Note that as a result of this, in order not to conflict with the start time forcing chain, they have to start at an element of S^n.

Now, for each edge $v_i, v_j \in E$, and each ordered pair of vertex selection chains C_a, C_b with $a, b \in \{1, 2, \ldots, k\}$ we dedicate an interval I_{v_i, v_j, C_a, C_b} of $2c_0 + 1$ time steps. More precisely, we have $m \cdot k(k - 1)$ intervals $[c_0 + \alpha(2c_0 + 1), c_0 + (\alpha + 1)(2c_0 + 1)]$ for $\alpha = 0, 1, \ldots, m \cdot k(k - 1) - 1$. And to each interval we assign a unique label I_{v_i, v_j, C_a, C_b} where $v_i v_j \in E$ and $a, b \in \{1, 2, \ldots, k\}$. In the interval I_{v_i, v_j, C_a, C_b} we will check whether the chains C_a and C_b did not select the edge $v_i v_j$, that is, whether C_a does not start at s_i or C_b does not start at s_j.

We will now extend the vertex selection chains. Consider the interval $[c_0, c_0 + (m \cdot k(k - 1))(2c_0 + 1)]$ from left to right. For each interval I_{v_i, v_j, C_a, C_b} that we encounter, we extend the vertex selection chains as follows.

■ For each chain C, with $C \neq C_a$, $C \neq C_b$, add a delay of $2c_0 + 1$.

■ Add the following gadget to the chain C_a: a delay of $c_0 - s_i$, a job, and then a delay of $c_0 + s_i$.

■ Add the following gadget to the chain C_b: a delay of $c_0 - s_j$, a job, and then a delay of $c_0 + s_j$.

Add one job at the end of all chains. See Figures 3, 4, and 5 for an example of the construction and a feasible schedule.

We claim that there is a feasible schedule, if and only if G has an independent set of size at least k.
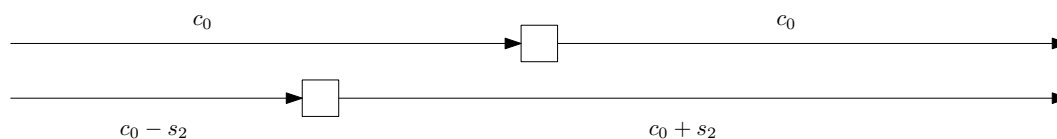
Figure 4 The part of the chains C_a and C_b for the interval I_{v_1,v_2,C_a,C_b} for the graph in Figure 3.
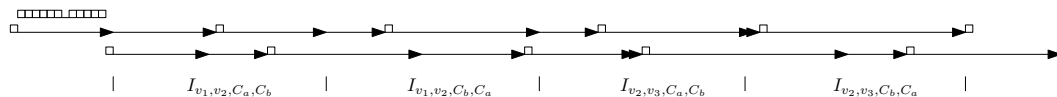
Figure 5 The instance of the scheduling problem constructed from the graph in Figure 3 and $k = 2$, and a feasible schedule for this instance.

▷ **Claim 4.3.** If G has an independent set of size at least k, then there is a feasible schedule.

Proof. Suppose $v_{i_1}, \ldots, v_{i_k}$ form an independent set in G. Let the vertex selection chain C_a start at times s_{i_a} for $a = 1, 2, \ldots, k$.

Notice that for the first c_0 time steps, at most one machine is used. The same holds for the last c_0 time steps. We show that this is a feasible scheduling by contradiction. Suppose that there are two machines needed at some time step β, and that β is in the interval $I = I_{v_i,v_j,C_a,C_b}$ for checking whether the chains C_a and C_b do not select the edge $v_i v_j$. Write $I = [\ell(I), r(I)]$. Notice that the job of C_a in the interval I starts at time $\ell(I) + c_0 - s_i + s_{i_a}$. Furthermore, the job of C_b in the interval I starts at time $\ell(I) + c_0 - s_j + s_{i_b}$. Thus, $\ell(I) + c_0 - s_i + s_{i_a} = \beta = \ell(I) + c_0 - s_j + s_{i_b}$. Equivalently, $s_{i_a} - s_i = s_{i_b} - s_j$. Since S^n is a Golomb ruler, it follows that either $s_{i_a} = s_i$ and $s_{i_b} = s_j$ or $s_{i_a} = s_{i_b}$ and $s_i = s_j$.

In the first case, $s_{i_a} = s_i$ and $s_{i_b} = s_j$, we see that $v_i = v_{i_a}$ and $v_j = v_{i_b}$. But there is no edge $v_{i_a} v_{i_b}$, since v_{i_a} and v_{i_b} are in an independent set. This yields a contradiction.

In the second case, $s_{i_a} = s_{i_b}$ and $s_i = s_j$, we see that $v_{i_a} = v_{i_b}$, but this yields a contradiction with the fact that the vertices of the independent set are distinct.

We conclude that this schedule always uses at most one machine. ◁

▷ **Claim 4.4.** If there is a feasible schedule, then G has an independent set of size at least k.

Proof. Suppose that there is a feasible schedule. Notice that the time of the first step of a vertex selection chains must be an element of S^n, otherwise the job conflicts with the start time forcing chain. Now, set $W = \{v_i \mid$ there is a vertex selection chain that starts at time $s_i\}$. Notice that $|W| = k$, as otherwise two vertex selection chains start at the same time, and conflict with each other for their first job.

We prove that W is an independent set by contradiction. Suppose that there exists an edge $v_i v_j$, with $v_i, v_j \in W$. Let C_a be the chain that starts at s_i and C_b the chain that starts at s_j. Now consider the interval $I = I_{v_i,v_j,C_a,C_b}$, and write $I = [\ell(I), r(I)]$. By the construction of the chains, it follows that C_a starts its job of the interval I at time $\ell(I) + c_0 - s_i + s_i = \ell(I) + c_0$. The job of C_b in the interval starts at time $\ell(I) + c_0$ as well. This yields a contradiction. We conclude that W is an independent set. ◁

This shows that the CHAIN SCHEDULING WITH EXACT DELAYS problem with a single machine, parametrized by the number of chains, is $W[1]$-hard. ◀

▶ **Lemma 4.5** (See [4, Lemma 4.5]). *The* CHAIN SCHEDULING WITH EXACT DELAYS *problem with one machine, parameterized by the number of chains c is in* $W[1]$.

4.2   Constant number of parallel machines

If we assume that there are m machines, but m is considered to be a constant (i.e., not a fixed parameter; m is part of the problem description), then the problem is $W[1]$-complete.

▶ **Theorem 4.6.** *The* Chain Scheduling with Exact Delays *problem with m machines, parameterized by the number of chains, is $W[1]$-complete.*

Hardness follows easily from the case that $m = 1$: add $m - 1$ chains with maximum length that consist of jobs with 0 delay. Membership follows by formulating the problem as a Weighted CNF-SAT problem, where we have a variable for each chain C and each time t that it can start and clauses that check that we never use more than m machines. For details see the full version [4].

4.3   Variable number of parallel machines

The main result of this section is the following theorem.

▶ **Theorem 4.7.** *The* Chain Scheduling with Exact Delays *problems with a variable number of machines, parameterized by the number of chains c, is $W[2]$-complete.*

The proof can be found in the full version [4]. Hardness is shown by a reduction from Dominating Set; membership by a reduction to Threshold Dominating Set.

4.4   Minimum delays

With a simple modification, the hardness proofs for exact delays in unary can be modified to hardness proofs for minimum delays (still in unary). The modification consists of taking a number of copies of the instance, as described below.

Suppose we have an instance for Chain Scheduling with Exact Delays with c chains and m machines. We build an instance for Chain Scheduling with Minimum Delays. The intuition behind the construction is the following: we build $cT + 1$ identical copies of the original instance after each other. Here, T is the maximum deadline of all chains in the Chain Scheduling with Exact Delays instance. The delay between two copies of a chain C is $T - \ell_C$, where ℓ_C is the minimum duration of C. So there will be at least T time units between the execution of a job in copy i and the same job in copy $i + 1$. Each copy is executed in its own slot of T consecutive time steps, in the same way as a solution of the original instance. The release date of the new chain will stay r_C, while we set the deadline as $cT^2 + d_C$.

Suppose we have a solution of the new instance. Every new chain has a minimum duration of $cT^2 + \ell_C$. Thus, the total slack in a chain, *i.e.*, the sum of the differences between the scheduled delay time and the stated minimum delay, cannot be larger than $cT^2 + d_C - r_C - cT^2 - l_C = d_C - r_C - l_C$. Notice that this is at most T. Thus, for one chain, we have at most T copies where there is a pair of successive jobs with delays not equal to the minimum delay. As we have c chains, and $cT + 1$ copies, there must be at least one copy where each pair of successive jobs of all chains has delay equal to the minimum delay. This copy gives a solution of the original instance.

See Figure 6 for an illustration, and [4, Section 4.4] for more details.
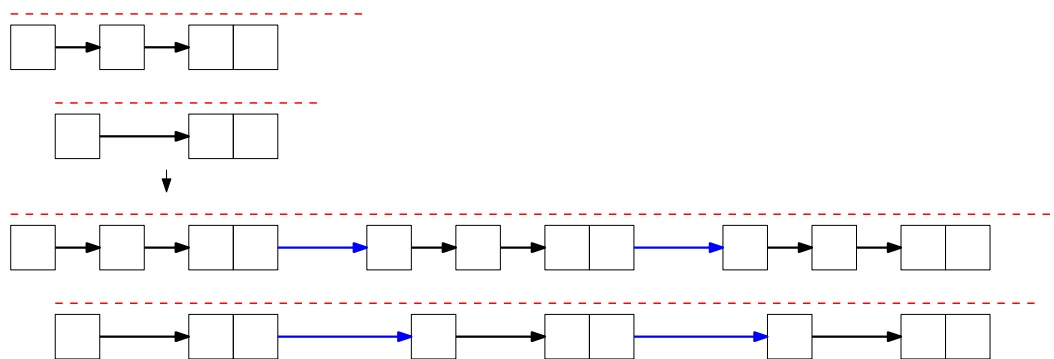
Figure 6 Construction in Section 4.4: copy every instance, the blue arrows are the delays between two copies of a chain, the red dashed line represent the interval from release date until deadline. The example shows the construction with three copies.

5   XP algorithms

In this section, we give two positive results. First, we show membership in XP for all studied variants, when delays are given in unary, with a relatively straightforward dynamic programming algorithm (Theorem 5.2; the proof can be found in the full version [4]). Then, in the case of one machine, we show that SCHEDULING WITH EXACT DELAYS is in XP, when parameterized by the number of chains, even when delays are given in binary. For a related, earlier result, see [6].

▶ **Lemma 5.1.** *Given an instance of* CHAIN SCHEDULING WITH EXACT DELAYS *or* CHAIN SCHEDULING WITH MINIMUM DELAYS, *one can build in polynomial time an equivalent instance, where all release dates and deadlines of chains are nonnegative integers, bounded by $cn(D+1)$, where c is the number of chains, n the number of jobs, and D the maximum delay between two successive jobs in a chain. In addition, for each chain C, we have $d_C - r_C \le n(D+1)$.*

▶ **Theorem 5.2.** CHAIN SCHEDULING WITH EXACT DELAYS *and* CHAIN SCHEDULING WITH MINIMUM DELAYS *belong to XP, when delays are given in unary, and parameterized by the number of chains or thickness, for any number of machines.*

The algorithm uses dynamic programming: we compute for each time step a table of states of partial solutions; the state tells for each chain what is the last job of the chain that is executed and at what time step this job was executed. For details, see the full version [4].

▶ **Theorem 5.3.** CHAIN SCHEDULING WITH EXACT DELAYS *with $m = 1$, parameterized by the number of chains, with delays in binary belongs to XP.*

Proof. Suppose we have chains $C_1, \ldots, C_c$. Suppose chain C_i has jobs $j_{i,1}, j_{i,2}, \ldots, j_{i,\ell_i}$, with $j_{i,a}$ directly preceding $j_{i,a+1}$; we write the exact delay of this constraint as $l_{i,a}$. Write $s(i,a) = \sum_{b=1}^{a-1}(l_{i,b}+1)$. Note that $j_{i,a}$ has to be scheduled exactly $s(i,a)$ time steps after $j_{i,1}$ starts.

For each chain C_i, we take a variable x_i that denotes the time that the first job of C_i is scheduled.

▷ **Claim 5.4.**   Variables $x_1, x_2, \ldots, x_c$ give a valid schedule, if and only if the following constraints are fulfilled.

1. For each i, $x_i \geq r_{C_i}$.
2. For each i, $x_i + s(i, \ell_i) < d_{C_i}$.
3. For each i and i' with $i \neq i'$, and each j, j' with $1 \leq j \leq \ell_i$, $1 \leq j' \leq \ell_{i'}$, we have $x_i + s(i, j) \neq x_{i'} + s(i', j')$.

The conditions state that chains do not start before the release date (1), do not finish after the deadline (2), and that no pair of jobs are scheduled at the same time (3). More details in [4].

The first step of the algorithm is to compute for each pair i, i' with $i \neq i'$ a set $U(i, i') = \{s(i', j') - s(i, j) \mid 1 \leq j \leq \ell_i,\ 1 \leq j' \leq \ell_{i'}\}$. Note that each of these sets has size $O(n^2)$, or more precisely, is at most the product of the sizes of the two chains. Now, sort each set $U(i, i')$.

Suppose $U(i, i') = \{a_1, a_2, \ldots, a_r\}$ with $a_1 < a_2 < \cdots < a_r$. Condition 3 of Claim 5.4 for the pair i, i' can be expressed as

$$(x_i - x_{i'} < a_1) \vee (a_1 < x_i - x_{i'} < a_2) \vee (a_2 < x_i - x_{i'} < a_3) \vee \cdots$$
$$\cdots \vee (a_{r-1} < x_i - x_{i'} < a_r) \vee (a_r < x_i - x_{i'})$$

Our algorithm now branches on these $O(n^2)$ possibilities. For each of the $O(c^2)$ pairs of chains, we have $O(n^2)$ branches, which gives a total of $O(n^{O(c^2)})$ subproblems.

Each of these subproblems asks to solve a set of inequalities. These inequalities are of the form $x_i - x_{i'} < a$ or $x_i \geq a$ (Condition 1 of Claim 5.4) or $x_i \leq a$ (Condition 2 of Claim 5.4), for some integers a. As we work with integers and look for integer solutions, we reformulate constraints of the form $x_i - x_{i'} < a$ as $x_i - x_{i'} \leq a - 1$. We now have a system of linear inequalities which can be solved in polynomial time with text book (shortest paths) methods, see e.g., [7, Section 24.4]. If at least one of the subproblems has a solution, then this solution gives starting times for the chains that gives a valid schedule; otherwise, there is no valid schedule.

We have $O(n^{O(c^2)})$ branches, each taking polynomial time, and this gives a running time of $O(n^{O(c^2)})$. ◀

6   Conclusions

In this paper, we have shown a number of results on the parameterized complexity of CHAIN SCHEDULING WITH EXACT DELAYS and CHAIN SCHEDULING WITH MINIMUM DELAYS. In a few cases, we obtained $W[1]$-completeness or $W[2]$-completeness; in the other cases, we only showed hardness results, often together with XP-membership. We expect that the problems, parameterized by the thickness do not belong to $W[P]$ – for the same "compositionality" reason as why one can believe that GRAPH BANDWIDTH does not belong to $W[P]$: see the discussion in [12, Section 4]. The machinery to prove such results currently is not available, but we conjecture that also the variants with minimum delays inhibit some form of compositionality and do not belong to $W[P]$.

We end this paper with mentioning some open problems. In this paper, we proved for the case that delays are given in binary, for only one of the cases membership in XP. What is the complexity of the other cases when delays are given in binary? Also, an interesting question is to study the variant where we have maximum delays, with all of its subcases.

Another open problem is whether the results where we prove $W[t]$-hardness for all t can be improved to $W[SAT]$-hardness. Our current constructions give instances that grow exponentially with the nesting depth of conjunctions and disjunctions. We currently do not

know how to avoid this exponential blowup, so it appears that possible $W[SAT]$-hardness proofs for Chain Scheduling with Exact Delays parameterized by thickness need different techniques.

References

1 Martin Aigner and Günter M. Ziegler. Bertrand's postulate. In *Proofs from THE BOOK*, pages 7–12. Springer, 2001. `doi:10.1007/978-3-662-04315-8_2`.

2 S. Bessy and R. Giroudeau. Parameterized complexity of a coupled-task scheduling problem. *Journal of Scheduling*, 22(3):305–313, 2019. `doi:10.1007/s10951-018-0581-1`.

3 Hans L. Bodlaender and Michael R. Fellows. $W[2]$-hardness of precedence constrained K-processor scheduling. *Operations Research Letters*, 18(2):93–97, 1995. `doi:10.1016/0167-6377(95)00031-9`.

4 Hans L. Bodlaender and Marieke van der Wegen. Parameterized complexity of scheduling chains of jobs with delays, 2020. arXiv preprint. `arXiv:2007.09023`.

5 Peter Brucker, Johann Hurink, and Wieslaw Kubiak. Scheduling identical jobs with chain precedence constraints on two uniform machines. *Mathematical Methods of Operations Research*, 49:211—219, 1999. `doi:10.1007/PL00020913`.

6 Mark Cieliebak, Thomas Erlebach, Fabian Hennecke, Birgitta Weber, and Peter Widmayer. Scheduling with release times and deadlines on a minimum number of machines. In *Exploring New Frontiers of Theoretical Informatics. IFIP International Federation for Information Processing*, volume 155, pages 209–222, 2004. `doi:10.1007/1-4020-8141-3_18`.

7 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009. URL: `http://mitpress.mit.edu/books/introduction-algorithms-third-edition`.

8 Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM Journal on Computing*, 24:873–921, 1995. `doi:10.1137/S0097539792228228`.

9 Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness II: On completeness for $W[1]$. *Theoretical Computer Science*, 141(1-2):109–131, 1995. `doi:10.1016/0304-3975(94)00097-3`.

10 P. Erdös and P. Turán. On a problem of Sidon in additive number theory, and on some related problems. *Journal of the London Mathematical Society*, s1-16(4):212–215, 1941. `doi:10.1112/jlms/s1-16.4.212`.

11 Michael R. Fellows and Catherine McCartin. On the parametric complexity of schedules to minimize tardy tasks. *Theoretical Computer Science*, 298(2):317–324, 2003. `doi:10.1016/S0304-3975(02)00811-3`.

12 Michael R. Fellows and Frances A. Rosamond. Collaborating with Hans: Some remaining wonderments. In Fedor V. Fomin, Stefan Kratsch, and Erik Jan van Leeuwen, editors, *Treewidth, Kernels, and Algorithms — Essays Dedicated to Hans L. Bodlaender on the Occasion of His 60th Birthday*, volume 12160 of *Lecture Notes in Computer Science*, pages 7–17. Springer, 2020. `doi:10.1007/978-3-030-42071-0_2`.

13 Matthias Mnich and René van Bevern. Parameterized complexity of machine scheduling: 15 open problems. *Comput. Oper. Res.*, 100:254–261, 2018. `doi:10.1016/j.cor.2018.07.020`.

14 Matthias Mnich and Andreas Wiese. Scheduling and fixed-parameter tractability. *Mathematical Programming*, 154(1):533–562, 2015. `doi:10.1007/s10107-014-0830-9`.

15 René van Bevern, Christian Komusiewicz, and Manuel Sorge. A parameterized approximation algorithm for the mixed and windy capacitated arc routing problem: Theory and experiments. *Networks*, 70(3):262–278, 2017. `doi:10.1002/net.21742`.

16 Erick D. Wikum, Donna C. Llewellyn, and George L. Nemhauser. One-machine generalized precedence constrained scheduling problems. *Operations Research Letters*, 16(2):87–99, 1994. `doi:10.1016/0167-6377(94)90064-7`.

17 Gerhard J. Woeginger. A comment on scheduling on uniform machines under chain-type precedence constraints. *Operations Research Letters*, 26(3):107–109, 2000. `doi:10.1016/S0167-6377(99)00076-0`.