Journal of Combinatorial Optimization manuscript No.
(will be inserted by the editor)

Single Machine Scheduling with Precedence Constraints and Bounded Maximum Delay Value

Maher Mallem · Claire Hanen · Alix Munier-Kordon

Received: date / Accepted: date

Abstract We consider single machine scheduling problems minimizing the makespan when tasks are subject to precedence constraints with minimum, exact or maximum delays. We investigate the parameterized complexity of these problems with respect to the maximum delay value ℓ_{max}, and we complement the landscape of results in this field.

We prove that several special cases of the problem with either exact of minimum delays are para-NP-complete or XNLP-hard with respect to ℓ_{max}, in contrast with the maximum delays case which seems easier. In particular, in the case of precedence chains and unit processing times, we establish that the problem is para-NP-complete with exact delays and XNLP-hard with minimum delays, while it is polynomial-time solvable with maximum delays.

We then investigate the combination of ℓ_{max} with the width of the precedence graph, and aim for drawing the line of fixed-parameter tractability for the three types of delays. We show that for those combined parameters, the maximum delays case is polynomial-time solvable for chains, whereas for minimum delays it is proved XNLP-complete even with unit processing times and only two available delay threshold values. If a general precedence graph is assumed, we also establish that the problem with minimum delays is XNLP-complete, even for unit processing time and equal delays, whereas it is in FPT for exact delays.

Maher Mallem[0000−0001−5654−1090] (corresponding author)
Inria, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP, UMR 5668, 69342, Lyon cedex 07, France
E-mail: maher.mallem@ens-lyon.fr

Claire Hanen[0000−0003−2482−5042]
Université Paris Nanterre, F-92000 Nanterre, France
Sorbonne Université, CNRS, LIP6, F-75005 Paris, France

Alix Munier-Kordon[0000−0002−2170−6366]
Sorbonne Université, CNRS, LIP6, F-75005 Paris, France

Finally we consider problems with time windows, and propose a fixed-parameter tractable algorithm for problems with unbounded processing times, a general precedence graph and all types of delays, with respect to parameters ℓ_{max} combined with the maximum slack of a task.

Keywords Scheduling · parameterized complexity · precedence delays

Mathematics Subject Classification (2020) MSC 03D15 · MSC 65Y20 · MSC 68M20

1 Introduction

In this paper we investigate the inclusion of precedence delays in several single machine scheduling problems with precedence constraints. Such delays specify a time value on top of existing precedence relations. Given two jobs i, j and a precedence constraint $i \rightarrow j$ forcing j to be started after i is completed, we can add a minimum - resp. exact, maximum - delay $\ell_{i,j}$ to ask that j must be started at least - resp. exactly, at most - $\ell_{i,j}$ time units after i is completed. Such delays can model a variety of scheduling problems in practice, e.g. product shelf life in the case of maximum delays or instruction scheduling on pipelined processors in the case of minimum delays (Bruno et al., 1980).

Precedence delays are a special case of the time lags defined by Brucker et al. (1999), which are not necessarily tied to a precedence relation. Other connex notions include communications delays - for which the delay is only applied when i and j are scheduled on different machines - and sequence-dependent setup times, the delay values of which depend on the order of the jobs (see for example Baart et al. (2021) or Mnich and van Bevern (2018), Section 5.3).

To describe the studied scheduling problems, we extend Graham's standard three field notation (Graham et al., 1979). The presence of precedence delays is denoted in the second field by the symbol ℓ associated with precedence constraints, with a superscript ex, min and/or max to describe the nature of delays. So problem notation $1|chains(\ell^{min}), \ell_{i,j} = \ell, p_j \in \{1, 2\}|C_{max} \leq D$ denotes the single machine problem of scheduling chains of tasks with minimum delays, assuming that the same minimum delay ℓ is associated to every precedence constraint, job processing times are either 1 or 2, and deciding whether the minimum makespan is not greater than D. With "$chains(\ell^{min}, \ell^{max})$" in the second field instead, both minimum and maximum precedence delays would be allowed. In this case, note that we can recreate an exact delay on precedence relations $i \rightarrow j$ by having $\ell_{i,j}^{min}$ equal to $\ell_{i,j}^{max}$.

This paper focuses on the computational complexity of $1|prec(\ell^X)|C_{max}$ and its subproblems with $X \in \{ex, min, max\}$. As we will see in the following paragraphs, most subproblems are strongly NP-complete and thus require to bound some property - like the maximum processing time or the maximum delay value - in order to obtain tractable algorithms. Given a problem $\mathcal{P}$ and a parameter $k : \mathcal{P} \rightarrow \mathbb{N}_0$, $\mathcal{P}$ is said to be fixed-parameter tractable parameterized

by k if there exists a deterministic algorithm which solves $\mathcal{P}$ in $f(k) \cdot \text{poly}(|\mathcal{I}|)$ time where f is some computable function and $|\mathcal{I}|$ is the input size. Then the couple $(\mathcal{P}, k)$ is said to belong to the FPT class. When a problem is believed to not be in FPT with respect to k, it is proved complete for either the para-NP class, the XNLP class or some class in the W-hierarchy $(\text{W}[t])_{t\in\mathbb{N}}$. In particular $\mathcal{P}$ is in para-NP (resp. XNLP) when there exists a nondeterministic algorithm which solves it in $f(k) \cdot \text{poly}(|\mathcal{I}|)$ time (resp. $f(k) \cdot \text{poly}(|\mathcal{I}|)$ time and $f(k) \cdot \log(|\mathcal{I}|)$ space (Elberfeld et al., 2015)). And $\mathcal{P}$ can be proved para-NP-hard with respect to k by proving it NP-hard for a fixed value of k (Flum and Grohe, 1998). Clearly when it comes to precedence delays, the most straightforward parameter is the maximum delay threshold value ℓ_{max} which appears in the input precedence graph. In the following paragraphs we detail the results obtained in the literature for each delay type.

Among the three precedence delay types considered in this work the exact ones have been the most studied, especially in the context of coupled task scheduling - i.e. when *prec* is only composed of isolated edges linking two tasks. A coupled task i is usually encoded as a triplet of integers (a_i, ℓ_i, b_i) where a_i, b_i are the processing times of the two tasks and ℓ_i is the length of the precedence delay in between them. A comprehensive overview of the classical complexity results in the literature was given by Khatami et al. (2020) and Chen and Zhang (2021). Baptiste (2010) showed that $1|(p, \ell, p)|C_{max}$, with identical processing times and identical precedence delays can be solved in $\mathcal{O}(f(\ell) \cdot \log(n))$ time, lowering the dependency in the number of jobs at the expense of an exponential blowout with respect to ℓ. Then Khatami et al. (2023) extended the method to show that $Pm|(a, \ell, b)|C_{max}$ parameterized by ℓ_{max} is in FPT. Bessy and Giroudeau (2019) explored the inclusion of a compatibility graph G_c which dictates when two coupled tasks are allowed to interleave. They showed that $1|(a_j, \ell_j, b_j), G_c|C_{max} \leq D$ is NP-hard even when G_c is a star, and that it is in FPT when parameterized by $max_j(a_j + \ell_j + b_j)$ plus the vertex cover of G_c. When minimizing the number of tardy jobs, although they proved that $1|(1, \ell, 1), G_c| \sum U_j$ is W[1]-hard parameterized by $\sum U_j$, they showed that it is in FPT when ℓ_{max} is fixed.

Minimum precedence delays typically allow more problems to be solved efficiently. While scheduling unit-time jobs on a single machine was proved strongly NP-hard on general precedence relations with a single delay threshold value by Leung et al. (1984), Bruno et al. (1980) showed that the problem can be solved in polynomial time when restricting *prec* to an inforest and adding deadlines (or restricting *prec* to an outforest and adding release dates). Still Finta and Liu (1996) showed that $1|prec(\ell^{min}), p_j = 1|C_{max} \leq D$ is strongly NP-hard. Engels (2000) considered restrictions to chains with a limited range of delay values and showed that $1|chains(\ell^{min}), \ell_{i,j} = \ell, p_j = 1|C_{max} \leq D$ remains strongly NP-complete. With jobs of length in $\{1, 2\}$, he even proved para-NP-completeness with respect to ℓ_{max} (with $\ell_{max} = 2$). Finally note that minimum delays can be used to simulate the parallel identical machine setting with regular precedence constraints. A well-known setting, the param-

eterized complexity of which has been extensively studied (Bodlaender et al., 2022; Nederlof and Swennenhuis, 2022; Swennenhuis, 2022).

Maximum precedence delays have been the least studied precedence delay type. However it is worth noting that $1|prec(\ell^{max}), \ell_{i,j} = \ell, p_j = 1|C_{max} \leq D$ is equivalent to the DIRECTED BANDWIDTH graph problem which was first studied by Garey et al. (1978). They showed that the problem is strongly NP-hard with trees of indegree one and outdegree at most two. In the case of polytrees - i.e. directed acyclic graphs whose underlying undirected graph is a tree - Bodlaender (2021) showed that the problem is W[t]-hard parameterized by the bandwidth threshold for all t positive integer when the underlying undirected graph is a caterpillar with hair length at most one. While parameterized graph problem k-BANDWIDTH was recently proved XNLP-complete (Bodlaender et al., 2022), it is open whether k-DIRECTED BANDWIDTH is also XNLP-complete.

On general precedence, it is worth noting that minimum precedence delays can be used to simulate release dates and deadlines by setting a global initial task and a global final task. As such, several references investigated the inclusion of job time windows to problems with chains of precedence. With $X \in \{min, ex\}$, Mallem et al. (2022) showed that $1|chains(\ell^X), p_j = 1, r_j, d_j|\star$ is strongly NP-hard even with fixed pathwidth μ - i.e. fixed maximum number of overlapping time windows at any time - and fixed slack σ - i.e. fixed maximum difference between the processing time of a job and the length of its time window. However in the minimum delays case the problem is in FPT when pathwidth μ is combined with maximum delay threshold value ℓ_{max} as the parameter, even on parallel identical machines with general precedence. Bodlaender and van der Wegen (2020) obtained several interesting results when time windows are defined on the precedence chains instead - denoted $[r_C, d_C)$ for a chain C. With $X = min$ (resp. ex) and delays given in unary, they showed that $1|chains(\ell^X), p_j = 1, r_C, d_C|\star$ is W[1]-hard (resp. W[1]-complete) parameterized by the number of chains. In the parallel identical machine setting with delays given in unary they also gave an XP algorithm parameterized by the thickness - i.e. the maximum number of chain time windows which can include a time unit -, essentially a pathwidth-like parameter on chain time windows.

The vast majority of these results suggests that, although restricting the values of the precedence delays is often not enough in itself to make the problem easier, in some cases bounding an extra property could yield a fixed-parameter tractable algorithm. The results presented in this paper reinforce this intuition.

Contribution

Upon studying the parameterized complexity of single machine scheduling with precedence delays, we make significant progress with respect to three parameters:

1. the maximum delay value ℓ_{max} which appears in the input,

2. ℓ_{max} plus the width w of the precedence graph,
3. ℓ_{max} plus the slack σ.

For problem $1|prec(\ell^X)|C_{max}$ with $X \in \{ex, min, max\}$ and bounded precedence delays, we show that the accurate parameterized characterization of each subproblem depends on the nature of the delays, the structure of the precedence graph, the range of available delay threshold and processing time values. We complete the results from the literature and confirm that in most cases our single machine problem with bounded delay value remains difficult. In particular, we complement the parameterized landscape of problem $1|chains(\ell^X), p_j = 1, r_j, d_j|\star$ with $X \in \{ex, min\}$ initiated by Mallem et al. (2022) - where it was shown para-NP-hard with respect to μ and in FPT with respect to $(\ell_{max} + \mu)$. With respect to ℓ_{max}, our results infer that this problem is XNLP-hard with minimum delays and para-NP-complete with exact delays. Table 1 summarizes our contributions for parameter ℓ_{max} in light of results of the literature.

<table>
<tr><th>$1|chains(\ell^X)|C_{max}$</th><th>$X = ex$</th><th>$X = min$</th><th>$X = max$</th></tr>
<tr><td>$\ell^X_{i,j} = \ell, p_j = 1$</td><td>unary W[1]-hard[b]</td><td>$\mathcal{O}(n \cdot \log(n))$ with unary chains[f]</td><td rowspan="5">$\mathcal{O}(|\mathcal{I}|)$[c]</td></tr>
<tr><td>$\ell^X_{i,j} \in \{0, \ell\}, p_j = 1$</td><td rowspan="4">para-NP-complete[a]</td><td rowspan="2">XNLP-hard[e]</td></tr>
<tr><td>Arbitrary $\ell^X_{i,j}$, $p_j = 1$</td></tr>
<tr><td>$\ell^X_{i,j} = \ell, p_j \in \{1, 2\}$</td><td rowspan="2">para-NP-complete[g]</td></tr>
<tr><td>Arbitrary $\ell^X_{i,j}$ and p_j</td></tr>
</table>

<table>
<tr><th>$1|prec(\ell^X)|C_{max}$</th><th>$X = ex$</th><th>$X = min$</th><th>$X = max$</th></tr>
<tr><td>$\ell^X_{i,j} = \ell, p_j = 1$</td><td>W[1]-hard[b]</td><td rowspan="2">XNLP-hard[d]</td><td rowspan="4">W[t]-hard for all t[h]</td></tr>
<tr><td>$\ell^X_{i,j} \in \{0, \ell\}, p_j = 1$</td><td rowspan="3">para-NP-complete[a]</td></tr>
<tr><td>Arbitrary $\ell^X_{i,j}$, $p_j = 1$</td><td rowspan="2">para-NP-complete[g]</td></tr>
<tr><td>$\ell^X_{i,j} = \ell, p_j \in \{1, 2\}$</td></tr>
<tr><td>Arbitrary $\ell^X_{i,j}$ and p_j</td></tr>
</table>

Table 1 Results with parameter ℓ_{max}. Contributions are in bold. [a]See Section 3, Proposition 1. [b]See Section 3, Proposition 2. [c]See Section 3, Proposition 3. [d]See Section 5, Theorem 1. [e]See Section 6, Theorem 2. [f]See Bruno et al. (1980). [g]See Engels (2000). [h]See Bodlaender (2021).

Furthermore, on chains of unit-time jobs with minimum delays, we improve the W[1]-hardness result with respect to w by Bodlaender and van der Wegen (2020) by proving XNLP-completeness with respect to $(\ell_{max} + w)$, even in the absence of chain time windows. Since this implies W[t]-hardness for all t (Bodlaender et al., 2022), under the assumption that W[1] $\neq$ W[t] for some $t \geq 2$, this differentiates it from the case with exact delays - which they proved W[1]-

complete with respect to w. In contrast, this problem with maximum delays is polynomial-time solvable. We also investigate the general case with an arbitrary precedence graph, still with unit processing times, and show that for equal exact delays and the problem is in FPT, whereas it is XNLP-complete for minimum delays. Our contributions for the combined parameters $(\ell_{max} + w)$ are summarized in Table 2.

$1\|chains(\ell^X)\|C_{max}$	$X = ex$	$X = min$	$X = max$
$\ell^X_{i,j} = \ell, p_j = 1$	**in FPT**[c]	$\mathcal{O}(n \cdot \log(n))$ with unary chains[g]	$\mathcal{O}(\|\mathcal{I}\|)$[a]
$\ell^X_{i,j} \in \{0, \ell\}, p_j = 1$	in W[1][h]	**XNLP-complete**[f]	
Arbitrary $\ell^X_{i,j}$, $p_j = 1$			
Arbitrary $\ell^X_{i,j}$ and p_j	**in XNLP**[b]		

$1\|prec(\ell^X)\|C_{max}$	$X = ex$	$X = min$	$X = max$
$\ell^X_{i,j} = \ell, p_j = 1$	**in FPT**[d]	**XNLP-complete**[e]	**in XNLP**[b]
$\ell^X_{i,j} \in \{0, \ell\}, p_j = 1$	**in W[1]**[d]		
Arbitrary $\ell^X_{i,j}$, $p_j = 1$			
Arbitrary $\ell^X_{i,j}$ and p_j	**in XNLP**[b]		

Table 2 Results with parameter $(\ell_{max} + w)$. Contributions are in bold. [a]See Section 3, Proposition 3. [b]See Section 3, Proposition 6. [c]See Section 4, Proposition 4. [d]See Section 4, Proposition 5. [e]See Section 5, Theorem 1. [f]See Section 6, Theorem 2. [g]See Bruno et al. (1980). [h]See Bodlaender and van der Wegen (2020).

We then investigate the case where tasks have time windows, and where the three type of delays are combined : $1|prec(\ell^{min}, \ell^{max}), r_j, d_j|\star$. We provide a fixed-parameter tractable algorithm for this general problem for parameter ℓ_{max} combined with the maximal slack σ of a task . To the best of our knowledge, this is the first fixed-parameter tractable algorithm on a scheduling problem with precedence delays and unbounded processing times.

This paper is organized as follows. In Section 3 we take maximum delay value ℓ_{max} as a parameter and prove the results of Table 1. In the next three sections we combine ℓ_{max} with the width w of the precedence graph. In Section 4 we consider subproblems which become easier to solve. On the contrary, Sections 5 and 6 show XNLP-completeness of two special cases with minimum delays and unit-time jobs, respectively on general precedence with equal delay values, and chains of precedence with two possible delay values. Section 7 shows the fixed-parameter tractable algorithm for the problem with time windows, by combining parameter ℓ_{max} with the maximal slack σ. Finally in Section 8 we give our concluding thoughts.

2 Preliminaries

Definition 1 Let $X \in \{ex, min, max\}$. An instance I of decision problem $1|prec(\ell^X)|C_{max} \leq D$ is a tuple $\langle \mathcal{J}, prec, \ell^X_{i,j}, p_j, D\rangle$ where:

- $\mathcal{J}$ is the set of jobs,
- $prec$ is a directed acyclic graph representing a strict partial order $\rightarrow$ on $\mathcal{J}$,
- optionally for each precedence relation $i \rightarrow j$, $\ell^X_{i,j} \in \mathbb{N}_0$ is a precedence delay value,
- for each $j \in \mathcal{J}$, $p_j \in \mathbb{N}^*$ is the processing time of job j,
- D is the makespan - i.e. maximum completion time - threshold.

A schedule $S : \mathcal{J} \rightarrow \mathbb{N}_0$ on such an instance I is said to be feasible if and only if:

1. at most one job is executed at a time:
$$\forall i, j \in \mathcal{J}, (i \neq j) \implies [S(j) + p_j \leq S(i)] \vee [S(i) + p_i \leq S(j)],$$
2. the makespan of schedule S is at most D:
$$\forall j \in \mathcal{J}, S(i) + p_i \leq D,$$
3. all precedence constraints are met:
$$\forall i, j \in \mathcal{J}, (i \rightarrow j) \implies (S(i) + p_i \leq S(j)),$$
4. each precedence delay constraint $\ell^X_{i,j}$ is met:
   - if $X = min$: $S(i) + p_i + \ell^{min}_{i,j} \leq S(j)$,
   - if $X = ex$: $S(i) + p_i + \ell^{ex}_{i,j} = S(j)$,
   - if $X = max$: $S(j) \leq S(i) + p_i + \ell^{max}_{i,j}$.

If more than one precedence delay type is made available, then in our extended Graham notation such delay types are listed in the parentheses following $prec$. For example an instance of $1|prec(\ell^{min}, \ell^{max})|C_{max} \leq D$ is a tuple $\langle \mathcal{J}, prec, \ell^{min}_{i,j}, \ell^{max}_{i,j}, p_j, D\rangle$ where each precedence relation $i \rightarrow j$ may have both a minimum and a maximum delay value.

Definition 2 Given a scheduling instance I with precedence constraints, we define width w as the size of the largest antichain in $\rightarrow$.

If I also has precedence delays, we define ℓ_{max} as the maximum delay threshold value which appears in I.

In the case where $prec$ is a set of disjoint chains, note that w is equal to the number of chains.

Definition 3 In a scheduling instance I with job time windows, each job $j \in \mathcal{J}$ is given a release date $r_j \in \mathbb{N}_0$ and a deadline $\bar{d}_j \in \mathbb{N}_0$. Then any feasible schedule S on I must verify:

$$\forall j \in \mathcal{J}, (r_j \leq S(j)) \wedge (S(j) + p_j \leq \bar{d}_j).$$

On scheduling problems with job time windows, whether there exists a feasible schedule can be asked without mentioning a makespan threshold - or, equivalently, by setting said threshold as the maximum deadline. In this case, we denote $\star$ in the third field of our extended Graham notation.

Definition 4 Given a scheduling instance with job time windows we define:

- pathwidth μ as the maximum number of overlapping time windows at any time.
- slack σ as $\max_{j\in\mathcal{J}}(\bar{d}_j - r_j - p_j)$.

Lemma 1 *Any YES-instance of* $1|prec(\ell^{min},\ell^{max}), r_j, \bar{d}_j|\star$ *has pathwidth at most* $2\sigma + 1$.

Proof We use the same argument as van Bevern et al. (2017) - see their Lemma 5.3. Let t be a time unit and let j be a job. If t is part of interval $[r_j, \bar{d}_j)$ then $r_j \leq t < \bar{d}_j$. Plus the definition of slack σ: $\bar{d}_j - p_j - \sigma \leq r_j$ and $\bar{d}_j \leq r_j + p_j + \sigma$. So: $r_j > t - p_j - \sigma$ and $\bar{d}_j \leq t + p_j + \sigma$. This means that whenever job j is scheduled, p_j consecutive time units will be taken by j in time interval $[t - \sigma - p_j + 1, t + \sigma + p_j)$. Thus at least one time unit will be taken by j in time interval $[t - \sigma, t + \sigma + 1)$. Since we only have one machine, this means that at most $2\sigma + 1$ jobs j can have t be a part of their time window $[r_j, \bar{d}_j)$ in any feasible schedule of this instance. This yields the wanted inequality. □

Condition "$\mu \leq 2\sigma + 1$" can be checked by computing both parameters in $\mathcal{O}(n \cdot \log(n))$ time. So w.l.o.g. one can assume that this condition holds in any instance considered in Section 7.

3 Results with Bounded Maximum Delay Value

In this section we show a number of results with exact and maximum delays. Most of them are inferred from the literature in a straightforward way.

In the case of exact delays the parameterized problem remains difficult even with fixed ℓ_{max}.

Proposition 1 *The following problems are para-NP-complete parameterized by* ℓ_{max} *(with* $\ell_{max} = 1$*):*

(i) $1|chains(\ell^{ex}), \ell_{i,j} \in \{0, \ell\}, p_j = 1|C_{max} \leq D$,
(ii) $1|chains(\ell^{ex}), \ell_{i,j} = \ell, p_j \in \{1, 2\}|C_{max} \leq D$.

Proof Both problems are trivially in NP. To prove hardness, we reduce from UNARY 3-PARTITION in both cases. Given the number of partitions B, the target sum T and positive integers $(a_j)_{1\leq j\leq 3B}$ with $\frac{T}{4} < a_j < \frac{T}{2}$ all given in unary, we are asked whether it is possible to partition the integers into B subsets of three elements of sum T.

(i) We set $D = B(2T-1)+1$ and we define a chain $(f_j)_{0\leq j\leq B(T+1)}$ of fill jobs the following way: for all $0 \leq k < B$:

$$f_{(T+1)k} \xrightarrow{=1} \dots \xrightarrow{=1} f_{(T+1)k+T} \xrightarrow{=0} f_{(T+1)(k+1)}$$

By direct induction on k, it is clear that this chain spans over exactly D time units and thus can only start at time 0 in any feasible schedule. As a result, this leaves B sequences of available time units spaced by two time units, where in each sequence the T available time units are spaced by one time unit. So, if each integer a_j is represented by a chain of a_j unit-time jobs with exact delay one everywhere, then the equivalence between the available time units of this scheduling instance and the UNARY 3-PARTITION instance is straightforward.

(ii) We use a very similar reduction. The only difference is that, for all $0 \leq k < B$, consecutive fill jobs $f_{(T+1)k+T}$ and $f_{(T+1)(k+1)}$ are fused into a single job of processing time two. Then the rest of the reduction and proof unfolds in the same manner. □

Proposition 2 *$1|chains(\ell^{ex}), \ell_{i,j} = \ell, p_j = 1|C_{max} \leq D$ is W[1]-hard parameterized by ℓ_{max} even with chains given in unary.*

Proof We reduce from the k-UNARY BIN PACKING problem, which was proved W[1]-hard by Jansen et al. (2013). Given bin capacity T, items with sizes positive integers $(a_j)_{1\leq j\leq n}$ given in unary and parameter k the question is whether it is possible to pack the items into k bins. To get our reduction we set $D = kT$ and $\ell = k-1$. Each integer a_j is represented by a chain of a_j unit-time jobs with delay ℓ everywhere. Then all jobs from a chain have the same time position modulo k. So the i^{th} bin is represented by all time positions modulo k within interval $[0, D-1]$. The fill jobs leave exactly B intervals of length T to schedule the other jobs. Then the equivalence between our scheduling instance and the k-BIN PACKING instance is straightforward. □

Note that this is not the case with maximum delays. In fact having chains with arbitrary delay values does not hinder polynomiality.

Proposition 3 *Any instance I of $1|chains(\ell^{max})|C_{max} \leq D$ can be solved in $\mathcal{O}(|I|)$ time.*

Proof Chains can be scheduled one after the other with delay 0 everywhere in the schedule. So one can simply add up the chain lengths then check if this sum is lower than or equal to D. □

4 Algorithmic Results with Bounded Maximum Delay Value and Precedence Width

In this section, we combine maximum delay value ℓ_{max} with the width w of the precedence graph as a parameter. In the case of precedence chains, note that w is equal to the number of chains.

Proposition 4 *Any instance $\mathcal{I}$ of $1|chains(\ell^{ex}), \ell_{i,j} = \ell, p_j = 1|C_{max}$ can be solved in $\mathcal{O}(|\mathcal{I}| \cdot (\ell+1)^w)$ time.*

Proof Once the starting time of the first job of a chain is known, the starting times of all the other jobs in the chain are known. Now, notice that if two chains start at the same time modulo $(\ell+1)$, then they can only be in sequence. So once the (starting time modulo $(\ell+1)$) of each chain is known, the makespan is the maximum over $s \in \{0, \ldots, \ell\}$ of the sum of the length of chains, the *starting time modulo* $(\ell+1)$ of which is s. This gives the above complexity: we can enumerate all tuples of *starting times modulo* $(\ell+1)$ in $\mathcal{O}((\ell+1)^w)$ time and then, for each tuple, compute the makespan. □

Proposition 5 *(i) Problem $1|prec(\ell^{ex}), p_j = 1|C_{max}$ polynomially reduces to $1|chains(\ell^{ex}), p_j = 1|C_{max}$ without increasing the values of ℓ_{max} and w.*

(ii) Problem $1|prec(\ell^{ex}), \ell_{i,j} = \ell, p_j = 1|C_{max}$ polynomially reduces to $1|chains(\ell^{ex}), \ell_{i,j} = \ell, p_j = 1|C_{max}$ without increasing the values of ℓ_{max} and w.

Proof *(i)* Consider an instance of $1|prec(\ell^{ex}), p_j = 1|C_{max}$. Let $U = [i_1, \ldots, i_x]$ be a sequence of jobs corresponding to an undirected chain of the precedence graph: for each $1 \leq y \leq x-1$ there is either a forward precedence constraint $a_y = (i_y, i_{y+1})$ or a backward precedence constraint $a_y = (i_{y+1}, i_y)$. We define the *length* of $L(U)$ by induction as follows: If $x = 2$ and a_1 is forward then $L(U) = \ell_{i_1,i_2} + 1$, and if a_1 is backward $L(U) = -\ell_{i_2,i_1} - 1$. If $x > 2$ let U' be undirected chain $[i_1, \ldots, i_{x-1}]$. Then, if a_{x-1} is a forward arc with respect to U, then $L(U) = L(U') + 1 + \ell_{i_{x-1},i_x}$. Otherwise $L(U) = L(U') - \ell_{i_x,i_{x-1}} - 1$.

Since we have exact delays it is straightforward to see that for any undirected chain $U = [i_1, \ldots, i_x]$ and any feasible schedule S:

$$s_{i_x} - s_{i_1} = L(U)$$

This implies that if there are two undirected chains from job i to job j, then they should have the same length. Otherwise it is a NO instance. This can be checked in polynomial time by computing the all pairs shortest length and then check whether there is an arc which is not the shortest length.

Moreover, it is clear that once the starting time of a job on a chain is known, then all the starting times along the chain are defined. So if we consider a connected component of the precedence graph, all pairs of nodes are linked by undirected chains, so that once the schedule of one job of the component is fixed, all the other jobs starting time are defined.

Given any job i of a connected component, we can define for any other job j of the same component the length L_{ij} of an undirected chain from i to j, with the property that in any feasible schedule S, $s_j = s_i + L_{ij}$.

As we have a single machine, for all $j, j', j \neq j'$ in the connected component of i, L_{ij} should be different from $L_{ij'}$. We can then sort them by increasing order (assuming $L_{ii} = 0$). Let us consider $j_1, \ldots, j_y$ such that $L_{i,j_1} < L_{i,j_2} < \ldots < L_{i,j_y}$ and observe that there is an equivalent set of

precedence constraints that leads to the same property of feasible schedules: a path $(j_1, j_2), (j_2, j_3), \ldots, (j_{y-1}, j_y)$ with precedence delays such that :

$$\ell'_{j_z, j_{z+1}} = L_{i,j_{z+1}} - L_{i,j_z} - 1 \geq 0.$$

Indeed in both cases, for any schedule and any $z \in \{1, \ldots, y-1\}$ we would have $s_{j_{z+1}} - s_{j_z} = \ell'_{j_z, j_{z+1}} + 1$.

So each connected component of the precedence graph is equivalent to a directed chain, and thus the instance can be reduced to a set of independent directed chains.

(*ii*) If all delays are equal with unit processing time tasks then, in any feasible instance, each node must have at most one predecessor and at most one successor. So that it must be a set of disjoint chains. □

We prove the membership of $1|prec(\ell^X), p_j = 1|C_{max} \leq D$ parameterized by the maximum delay value ℓ_{max} to the XNLP Class for any type of delays $(X = min, max, ex)$.

Proposition 6 *For any delay type X in $\{min, ex, max\}$, decision problem $1|prec(\ell^X)|C_{max} \leq D$ parameterized by $\ell_{max} + w$ is in XNLP.*

Proof To prove the XNLP membership, we need to produce a nondeterministic algorithm running in space logarithmic in the input size times some computable function of parameters ℓ_{max} and w. This means that the nondeterministic algorithm works on an encoded version of the instance, so that it can check whether a precedence arc exists between two tasks in polynomial time. And, if it needs extra space to do some calculation, this extra space should be of logarithmic size. Notice that the instance size is $\mathcal{O}(n^2 \cdot \log(p_{max}))$ where n is the number of jobs and p_{max} the maximum processing time.

Let us consider an instance $\mathcal{I}$ of $1|prec(\ell^X)|C_{max} \leq D$ with n jobs, maximum delay value ℓ_{max} and width w of the precedence graph. Let G be the graph of precedence relations.

The proof is organized as follows. We first observe that the existence of path between nodes can be checked by a nondeterministic polynomial-time algorithm with logarithmic space. We then show that we can check in polynomial time and logarithmic space whether the makespan constraint can be ignored and show that, if it cannot be ignored, then the maximum idle time of a schedule can be computed in polynomial time and encoded in logarithmic space. Provided these pieces of information, we describe the nondeterministic algorithm which builds a schedule (if there exists one). The logarithmic space is ensured by only storing job starting and completion time offsets not greater than ℓ_{max} with respect to the completion time of the current job, and not the schedule in itself. We finally prove that the algorithm returns a YES instance if and only if there exists a feasible schedule with makespan at most D.

Let us first remark that considering two nodes x, y of a directed graph, determining whether there is a path from x to y can be done in $\mathcal{O}(\log(n))$ space and linear time by a nondeterministic algorithm - i.e. problem STCON

is in class NL (Savitch, 1970). At each step this algorithm stores the index of the last node of the current path and the number of nodes of the current path, and an oracle gives the next node of the path. The algorithm just checks whether the number of nodes is still less than n and that the final node is indeed y.

Another remark is that, to satisfy the constraint $C_{max} \leq D$, we only have to check that the idle time of the schedule is not greater than $D - \sum_{J_i \in \mathcal{J}} p_i$. If we have maximum delays, assuming $D \geq \sum_{J_i \in \mathcal{J}} p_i$, this can be always satisfied since the schedules without idle times dominate the other schedules. In the minimum delay or exact delay cases, if $D - \sum_{J_i \in \mathcal{J}} p_i > n\ell_{max}$ and there exists a schedule S which fulfills the precedence delays, then $C_{max}(S) \leq D$. So for minimum or exact delays we first show that a XNLP algorithm can be built to check whether $D - \sum_{J_i \in \mathcal{J}} p_i > n\ell_{max}$.

To this purpose let us define $Q = \sum_{J_i \in \mathcal{J}} p_i + n\ell_{max} - D$. We show that Q being negative can be checked in polynomial time and logarithmic space. Indeed, let us remark that in order to sum (or subtract) $\mathcal{O}(n)$ signed binary encoded numbers (using two's complement), we just need a buffer memory of $\mathcal{O}(\log(n))$ length to encode the carries. Indeed, iteratively, for $r = 0, \ldots, 1 + \log(n(p_{max} + \ell_{max}) + D)$, we sum the $(n+2)$ r^{th} bits plus the corresponding $O(\log(n))$ bits of the carries issued by the previous iterations. Q being negative simply results by the existence of a last and most significant bit with value 1. Notice that if $Q \geq 0$, then $Q' = D - \sum_{J_i \in \mathcal{J}} p_i$ can be encoded in $\mathcal{O}(\log(n) + \log(\ell_{max}))$ space. From now, in the case of minimum or exact delays, if $Q \geq 0$ then we assume that Q' has been computed.

The nondeterministic algorithm builds a schedule from left to right in n steps, scheduling a new job at each step. We call *observation window* the time interval $[t - \ell_{max}, t]$ where t is the completion time of the current partial schedule. At each step the algorithm stores the following:

- For each job J_i completed in the current observation interval $[t - \ell_{max}, t]$, the index of the task and the difference between t and the completion time $t_i = t - C_i$ of J_i. The space needed is thus $\mathcal{O}(\ell_{max}(\log(n) + \log(\ell_{max}))$.
- The set M of maximal scheduled jobs (i.e. with no successor scheduled yet), and for each $J_j \in M$ its index and a boolean b_j indicating whether it completes more than ℓ_{max} time units before t (e.g. if J_j completes in the observation interval then $b_j = 0$). There are at most w jobs in M, which form an antichain of the precedence graph. So the space needed to store M is $\mathcal{O}(w \log(n))$.
- In case of minimum or exact delays with $Q \geq 0$, an idleness time s representing the sum of idle times between consecutively scheduled jobs.

Now the algorithm uses an oracle that provides at each step:

- A job J_i to be scheduled after the already scheduled tasks.
- In the case of minimum or exact delays, a non negative value $e_i \leq \ell_{max}$ such that J_i is scheduled at time $t + e_i$, where t is the completion time of the current partial schedule. We assume that for the first scheduled task

we have $e_i = 0$. Notice that in the case of maximum delays, we also assume $e_i = 0$ for all J_i.
- If the processing time of J_i is greater than ℓ_{max}, the binary piece of information information denoted by "$p_i > \ell_{max}$" is given, and otherwise the processing time p_i of J_i.

Notice that the space requirement for the oracle information is $\mathcal{O}(\log(n) + \log(\ell_{max}))$, so it fits the requirements of class XNLP.

The algorithm then checks whether the selected job J_i has not already been scheduled (i.e. not an ancestor of a job in M and not a member of M) and all its predecessors have been scheduled (i.e. they are an ancestor of a job in M or a member of M). This can be done using the previous NL algorithm for STCON $\mathcal{O}(n^2)$ times.

The algorithm must then check whether J_i can indeed be scheduled at time $t + e_i$ without contradicting the precedence delays:

- For each J_j predecessor of J_i completed in the observation interval we check whether $t_j + e_i$ is:
  - greater than or equal to $\ell_{j,i}$ in the case of minimum delays,
  - lower than or equal to $\ell_{j,i}$ in the case of maximum delays,
  - equal to $\ell_{j,i}$ in the case of exact delays.

  If any of these conditions is false, the algorithm returns NO.
- In case of maximum or exact delays:
  - For each job J_j in M such that $b_j = 1$ (e.g. not completed in the observation interval), if J_j is a predecessor of J_i, then the delay constraint cannot be met and the algorithm returns NO.
  - For each job J_j in the observation interval, we check whether there is a successor $J_{j'}$ of J_j which collides with J_i - or, in other words, whether $\ell_{j,j'} < t_j + e_i + \min(\ell_{max}, p_i))$. In this case the precedence delay between $J_{j'}$ and J_j asks to start $J_{j'}$ before J_i is completed, while it would only start at the time J_i is completed or after if we kept going. So the algorithm returns NO.

Finally, if no infeasibility is detected, we must update information on the current state.

- We update the schedule of the observation interval. If $p_i > \ell_{max}$ then the next observation interval only contains J_i with $t_i = 0$. Otherwise, for each J_j currently in the observation interval, if $t_j + e_i + p_i > \ell_{max}$ then J_j is removed from the observation interval. If not then t_j is updated: $t_j \leftarrow t_j + e_i + p_i$.
- To update M, we need to remove the predecessors of J_i in M and add J_i.
- To update the booleans, we first set $b_i = 0, t_i = 0$. If $p_i > \ell_{max}$ then all booleans of jobs J_j of M, $j \neq i$ are set to $b_j = 1$. Otherwise, if J_j is completed in the updated observation interval then $b_j = 0$ else $b_j = 1$.
- If $Q \geq 0$, idleness time s is updated to $e_i + s$.

Once all tasks are scheduled:

- in the case of maximum delays, or minimum or exact delays when $Q < 0$, the algorithm returns YES,
- in the case of minimum or exact delays when $Q \geq 0$, it returns YES if and only if $s \leq Q'$.

Obviously, if there exists a feasible schedule with makespan not greater than D, and if the oracle provides the jobs in this order with the right idle time between then, then the algorithm will return YES. Conversely, if the algorithm ends with a YES instance, let us show that a feasible schedule has been built.

The single machine constraint is fulfilled since each job J_i is scheduled e_i time units after the completion of the previous jobs on the machine, with $e_i \in [0, \ell_{max}]$. We can also notice that the makespan constraint is satisfied (provided that $D \geq \sum_{i=1}^{n} p_i$). Indeed if we have maximum delays $e_i = 0$ for all i. For minimum or exact delays when $Q < 0$ then any feasible schedule with $e_i \leq \ell_{max}$ satisfies the constraint. When $Q \geq 0$ we know that $\sum_{i=1}^{n} e_i \leq Q'$, so the makespan is not greater than D.

So we only have to check that all precedence constraints and their corresponding delays are met. Assume by contradiction that there is a constraint $J_i \rightarrow J_j$ which is not satisfied in the schedule build by the algorithm. Notice that, according to the path verification step of the algorithm, the oracle cannot schedule J_j before J_i. Let us denote by S_i (resp. S_j) the starting time of J_i (resp. J_j) according to the algorithm.

Suppose we have maximum delays. We then assume that $S_i + p_i + \ell_{i,j} < S_j$. Let $J_{j'}$ be the task scheduled after J_i and either in progress or completed at time $S_i + p_i + \ell_{i,j}$ (it exists since there is no idle time in case of maximum delays). Then, at the iteration where $J_{j'}$ was scheduled, J_i was in the observation interval. So the algorithm checked and realized that J_j would collide with J_i, and thus it should have returned NO.

Assume now we have minimum or exact delays. If $S_i + p_i + \ell_{i,j} > S_j$ this contradicts the fact that at, the iteration where J_j was scheduled, J_i being in the observation interval at this step, the algorithm would have checked and realized that the precedence delay was not satisfied. Finally, if we have exact delays and $S_i + p_i + \ell_{i,j} < S_j$, this contradicts the fact that, at the iteration J_j was scheduled, J_i being out of the observation interval at this step, the algorithm would have returned NO. □

5 XNLP Hardness on General Precedence with Minimum Delays

In this section we consider single machine scheduling of unit-time jobs with general precedence and equal delays of length ℓ. We prove the following hardness result:

Theorem 1 *Decision problem* $1|prec(\ell^{min}), \ell_{i,j} = \ell, p_j = 1|C_{max} \leq D$ *is XNLP-complete parameterized by* $\ell_{max} + w$.

We propose a reduction from problem $P|prec, p_j = 1|C_{max} \leq D$ parameterized by $m+w$, which was proved XNLP-complete by Bodlaender et al. (2022). Let $I = \langle \mathcal{J}, prec, m, D \rangle$ be an instance of problem $P|prec, p_j = 1|C_{max} \leq D$ with $\mathcal{J} = (J_j)_{1 \leq j \leq |\mathcal{J}|}$. We build an instance $I' = \langle \mathcal{J}', prec', D' \rangle$ of problem $1|prec(\ell^{min}), \ell_{i,j} = \ell, p_j = 1|C_{max} \leq D$ which will be feasible if and only if I is feasible.

We propose to linearize the parallel machine instance. Each time unit in the parallel machine setting is represented as a time segment with length the number of available machines. By setting minimum delay value ℓ as the number of machines, this guarantees that two jobs J_i, J_j related by precedence cannot be scheduled in the same time segment of length ℓ - i.e. not at the same time in the parallel identical machine setting.

Now this alone would not represent a parallel machine setting faithfully. Indeed if $J_i \rightarrow J_j$ in the parallel machine setting then J_j can be scheduled on any machine in the time unit after the completion of job J_i. If the time segments are too close from each other in our single machine representation then the precedence delays could forbid some time units in the next time segment. One way to fix this is to separate the time segments by at least m time units and add fill jobs in between.

We define our reduction the following way:

Definition 5
Given $I = \langle \mathcal{J}, prec, m, D \rangle$ an instance of $P|prec, p_j = 1|C_{max} \leq D$, we define $I' = \langle \mathcal{J}', prec', D' \rangle$ instance of $1|prec(\ell^{min}), \ell_{i,j} = \ell, p_j = 1|C_{max} \leq D$. We set $\ell = m$ and $D' = (2D-1) \cdot (m+1) + 1$. We define the set of jobs $\mathcal{J}'$ as $\mathcal{J}$ plus the following jobs:

- skeleton jobs a_i, $0 \leq i \leq 2D-1$,
- fill jobs $b_{i,j}$, $0 \leq i \leq D-2$, $0 \leq j \leq m-1$.

We set precedence graph $prec'$ as $prec$ with minimum delay ℓ on every precedence relation, plus the following relations:

- $a_i \xrightarrow{\geq \ell} a_{i+1}$, $0 \leq i \leq 2D-1$,
- $a_{2i} \xrightarrow{\geq \ell} b_{i,j} \xrightarrow{\geq \ell} a_{2i+3}$, $0 \leq i \leq D-2$, $0 \leq j \leq m-1$.

Note that among the added jobs you cannot find more than $(m+1)$ of them which are not related to each other by $prec'$. So the width w' of precedence graph $prec'$ is bounded by the width w of $prec$ plus $(m+1)$.

Lemma 2 *In any feasible schedule S' of I':*

(*i*) $S'(a_i) = i \cdot (m+1)$ *for all* $0 \leq i \leq 2D-1$,
(*ii*) $(2i+1) \cdot (m+1) < S'(b_{i,j}) < (2i+2) \cdot (m+1)$ *for all* $0 \leq i \leq D-2$ *and* $0 \leq j \leq m-1$.

Proof (*i*) Let $0 \leq i \leq 2D-1$. If job a_i is scheduled earlier than $i \cdot (m+1)$ in S' then job a_0 would have to be scheduled at a time earlier than 0, which is not possible. And if a_i is scheduled later than $i \cdot (m+1)$ then job a_{2D-1} would

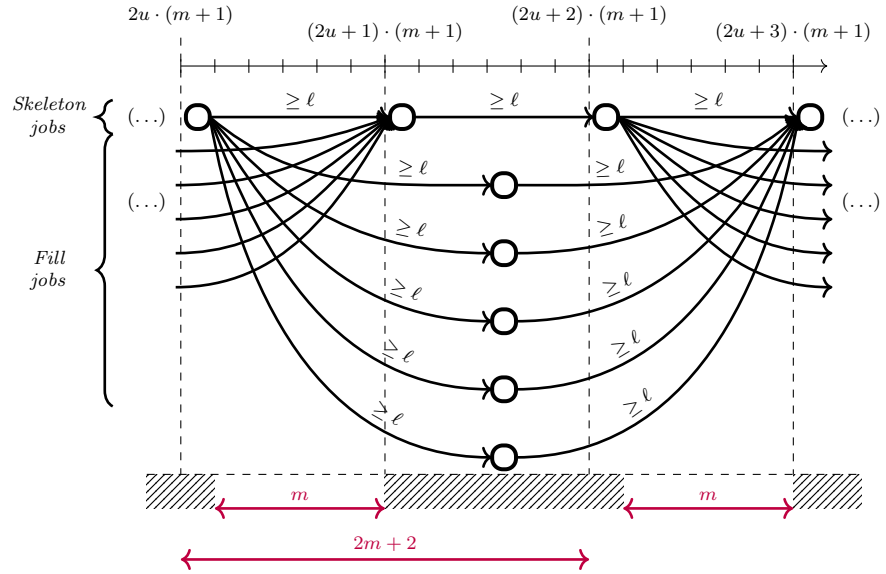

Fig. 1 Illustration of skeleton jobs and fill jobs with general precedence

have to be scheduled at a time later than $2D \cdot (m+1) - m$, i.e. at deadline D' or later. So necessarily $S'(a_i) = i \cdot (m+1)$.

(ii) Let $0 \leq i \leq D-2$ and consider fill job $b_{i,j}$ for any j. The result is directly inferred from point (i) and the fact that a_{2i} (resp. a_{2i+3}) is a predecessor (resp. successor) of $b_{i,j}$. □

Proposition 7 *Instance I is feasible if and only if instance I' is feasible.*

Proof Let us name the machines from 1 to m arbitrarily.

($\Longleftarrow$) Let S' be a feasible schedule of I'. By Lemma 2 skeleton jobs and fill jobs fully occupy times 0, $D'-1$ and time intervals of the form $[(2i+1) \cdot (m+1), (2i+2) \cdot (m+1)]$ with $0 \leq i \leq D-2$. So given a job J_j from $\mathcal{J}$, $S'(j)$ is necessarily of the form $2i \cdot (m+1) + k$ with $0 \leq i \leq D-1$ and $1 \leq k \leq m$. We propose schedule S on instance I where job J_j would then be scheduled at time i on machine k. Since S' is feasible this guarantees that there is at most one job per couple (time unit, machine). Plus if we had $J_i \xrightarrow{\geq \ell} J_j$ in I' then, in schedule S', J_j was scheduled at a later time segment of length m than J_i. So, in schedule S, J_j is scheduled at a later date than J_i and precedence relation $J_i \rightarrow J_j$ is met. Thus S is feasible.

($\Longrightarrow$) Let S be a feasible schedule of I. We propose schedule S' defined the following way:

- for every job J_j in $\mathcal{J}$ if it was scheduled on machine i then we set $S'(j) = 2S(j) \cdot (m+1) + i$,
- $S'(a_i) = i \cdot (m+1)$ for all $0 \leq i \leq 2D-1$,
- $S'(b_{i,j}) = (2i+1) \cdot (m+1) + (j+1)$ for all $0 \leq i \leq D-2$ and $0 \leq j \leq m-1$.

We show that S' is feasible. First note that skeleton jobs and fill jobs do not interfere with each other or with the jobs from $\mathcal{J}$. Now since S is feasible there is at most one job J_j per couple (time unit, machine). Plus every job J_j is scheduled earlier than D in S, so it is scheduled earlier than time $2(D-1)\cdot(m+1)+m+1 = D'-1$ in S'.

Now only precedence relations are left. The concerned skeletons jobs and fill jobs are placed at least $m+1 = \ell+1$ time units from each other, so no issue with them. Now given two jobs J_i, J_j from $\mathcal{J}$ such that $J_i \to J_j$ they are respectively scheduled in time segments $[2S(i)\cdot(m+1)+1, 2S(i)\cdot(m+1)+m]$ and $[2S(j)\cdot(m+1)+1, 2S(j)\cdot(m+1)+m]$. So their relative position is preserved and they are at least $m+2 = \ell+2$ time units from each other. So the precedence constraint - minimum delay included - is fulfilled. Thus S' is feasible. □

This concludes the XNLP-hardness proof of the corresponding parameterized problem.

6 XNLP Hardness on Chains with Minimum Delays

In this section we restrict to chains of precedence but we consider now two possible values of delays: 0 and ℓ. We show that the parameterized problem with respect to $\ell_{max}+w$ remains hard:

Theorem 2 *Decision problem* $1|chains(\ell^{min}), \ell_{i,j} \in \{0,\ell\}, p_j = 1|C_{max} \leq D$ *is XNLP-complete parameterized by* $\ell_{max}+w$.

Membership follows from Proposition 6. To prove XNLP-hardness, we reduce from the instances of problem $P|prec, p_j = 1|C_{max} \leq D$ which were used by Bodlaender et al. (2022) to prove that the problem is XNLP-complete parameterized by $m+w$.

The proof by Bodlaender et al. (2022) used a reduction from another XNLP-complete problem called ACCEPTING NNCCM, which stands for *Nondeterministic Nondecreasing Checking Counter Machine*. We recall that instance χ of the latter problem is a 3-tuple (k,n,s) where k is the number of counters - and the parameter -, n is the positive integer upper bound on all counter ranges, and s is a sequence of 4-tuples (i_1,i_2,n_1,n_2) called checks, where positive integers i_1,i_2 are counter indexes and nonnegative integers n_1,n_2 are counter values. The Counter Machine accepts the instance if, considering the machine rules (which are out of the scope of our paper), for each check (i_1,i_2,n_1,n_2) the two counters i_1,i_2 cannot have the values n_1,n_2 simultaneously. They proposed an instance I of $P|prec, p_j = 1|C_{max} \leq D$ which is feasible if and only χ is a YES instance of ACCEPTING NNCCM. Below is the definition of instance I. We also recall the main arguments that led to this instance, in order to transpose them in the context of precedence delays:

Definition 6 (Bodlaender et al., 2022) Given $\chi = (k, n, s)$ instance of ACCEPTING NNCCM, we define $I = \langle \mathcal{J}, prec, m, D \rangle$ an instance of problem $P|prec, p_j = 1|C_{max} \leq D$ the following way. Let r be the length of sequence s (number of checks) and $c = (kn + 1)(n + 1)$. We set the number of machines $m = 2k + 1$ and deadline $D = rc + n + 1$.

Instance I contains:

- a chain of time sequence jobs $a_0 \to a_1 \to \ldots \to a_{D-1}$
  As the deadline is D, this implies that in any feasible schedule, a_t is scheduled at time t, for $0 \leq t \leq D - 1$.
- We define critical time units t^* such that $t^* \mod (n + 1) = n$.
- For each critical time unit t^*, there are $k - 1$ time indicator tasks $b_{t^*}^{(x)}$, $1 \leq x \leq k - 1$, and chained so that $a_{t^*-1} \to b_{t^*}^{(x)} \to a_{t^*+1}$.
  This implies that in any feasible schedule, at each critical time t^* k machines are used for a_{t^*} and all indicator tasks $b_{t^*}^{(x)}$.
- k chains of counter sequence jobs $c_t^{(i)}$, $1 \leq i \leq k$, $0 \leq t \leq D - n - 1$, with $c_0^{(i)} \to c_1^{(i)} \to \ldots \to c_{D-n-1}^{(i)}$.
  The counter sequence jobs have more flexibility than time sequence jobs: along a counter sequence chain $(c_t^{(i)})_t$ with fixed i, there are n time units in $[0, D)$ where no task of the chain is performed. One of the key arguments is that if the time interval is decomposed in u consecutive intervals, then if $u > kn$, there will be at least one interval in which at each time t of this interval a task of each counter sequence is performed.
- for the j^{th} check (i_1, i_2, n_1, n_2) in sequence s, $1 \leq j \leq r$: check jobs $d_{t^*-n_1}^{(i_1)}$ and $d_{t^*-n_2}^{(i_2)}$ for any critical t^*, with the precedence relations:
  $c_{t^*-1-n_1}^{(i_1)} \to d_{t^*-n_1}^{(i_1)} \to c_{t^*+1-n_1}^{(i_1)}$ and $c_{t^*-1-n_2}^{(i_2)} \to d_{t^*-n_2}^{(i_2)} \to c_{t^*+1-n_2}^{(i_2)}$
  Consider a time interval where at each time a job of each counter sequence jobs is performed. The precedence relations implies that if t^* is a critical time unit of this interval for which a check job $d_t^{(i)}$ is performed then it is performed at the same time as $c_t^{(i)}$. So, at time t^* are performed $k - 1$ indicator tasks $b_{t^*}^{(x)}$, one time sequence job a_{t^*} and k counter jobs. This fills $2k$ machines. As there are only $2k + 1$ machines, there is no check (i_1, i_2, n_1, n_2) such that both check jobs $d_{t^*-n_1}^{(i_1)}$ and $d_{t^*-n_2}^{(i_2)}$ are performed at time t^*.

The reduction was proved correct by Bodlaender et al.:

Proposition 8 *(Bodlaender et al., 2022) I is feasible if and only if χ is a YES instance of ACCEPTING NNCCM.*

Note that in instance I, without the time indicators and the check jobs, the precedence graph would be a union of chains. So we show how to replicate such gadgets with chains and two distinct delay values - 0 and $\ell = m$.

We build a reduction from instance I to an instance I' of our problem $1|chains(\ell^{min}), \ell_{i,j} \in \{0, \ell\}, p_j = 1|C_{max} \leq D'$. Instance I' contains $k + 2$ chains described as skeleton chain, fill chain, and k counter-check chains.

We first start by defining the chains associated to the time indicators and time sequence jobs. The purpose of time indicators in I is to have $m-k$ available machines at critical time units instead of $m-1$ everywhere else. So we propose to replicate this effect with two chains. First as in Section 5 a chain of "skeleton" jobs will have no leeway and help delimit the time intervals. Then a second chain will cover every other time interval with fill jobs.

We now define more accurately skeleton jobs and fill jobs:

Definition 7 Given an instance $I = \langle \mathcal{J}, prec, m, D\rangle$ as described in Definition 6, we define skeleton jobs and fill jobs in instance I'. We set $\ell = m = 2k+1$ and $D' = (2m+2)\cdot D + r(kn+1)(k-1)+1$.

- for any $0 \leq \beta \leq D-1$, we define skeleton jobs $e_{2\beta}, e_{2\beta+1}$, plus $k-1$ extra jobs $e^{(x)}_{2\beta+1}$ if β is critical ($1 \leq x \leq k-1$), and a final skeleton job e_{2D},
- skeleton jobs are chained as follows:
  - $\left\{ \begin{array}{ll} e_{2\beta} \xrightarrow{\geq \ell} e^{(1)}_{2\beta+1} \xrightarrow{\geq 0} (\ldots) \xrightarrow{\geq 0} e^{(k-1)}_{2\beta+1} \xrightarrow{\geq 0} e_{2\beta+1} & \text{if } \beta \text{ is critical,} \\ e_{2\beta} \xrightarrow{\geq \ell} e_{2\beta+1} & \text{otherwise} \end{array} \right\}$, $0 \leq \beta \leq D-1$,
  - $e_{2\beta+1} \xrightarrow{\geq \ell} e_{2\beta+2}$, $0 \leq \beta \leq D-1$,
- we define fill jobs $f_{\beta,j}$, $0 \leq \beta \leq D-1$, $0 \leq j \leq m$ plus $k-1$ extra jobs $f^{(x)}_{\beta}$ if β is critical ($1 \leq x \leq k-1$),
- fill jobs are chained as follows:
  - $f_{\beta,j} \xrightarrow{\geq 0} f_{\beta,j+1}$, $0 \leq \beta \leq D-1$, $0 \leq j \leq m-2$,
  - $\left\{ \begin{array}{ll} f_{\beta,m-1} \xrightarrow{\geq \ell} f^{(1)}_{\beta} \xrightarrow{\geq 0} (\ldots) \xrightarrow{\geq 0} f^{(k-1)}_{\beta} \xrightarrow{\geq 0} f_{\beta,m} & \text{if } \beta \text{ is critical,} \\ f_{\beta,m-1} \xrightarrow{\geq \ell} f_{\beta,m} & \text{otherwise} \end{array} \right\}$, $0 \leq \beta \leq D-1$,
  - $f_{\beta,m} \xrightarrow{\geq 0} f_{\beta+1,0}$, $0 \leq \beta \leq D-2$.

We now define a sequence of particular time units,$(u_\beta)_{0\leq\beta\leq D-1}$ that will be useful to analyze the behavior of skeleton jobs and fill jobs together:

Definition 8 Let $\beta \in [0, D]$. We define $u_0 = 0$,

$$u_{\beta+1} = \left\{ \begin{array}{ll} u_\beta + 2m + 2 + k - 1 \; if \; \beta \text{ is critical} \\ u_\beta + 2m + 2 \qquad\qquad otherwise \end{array} \right. \tag{1}$$

So u_β is equal to $(2m+2)\cdot\beta$ plus $(k-1)$ times the number of critical time units prior to β.

Figure 2 (resp. 3) illustrates how a noncritical (resp. critical) time unit in instance I is simulated in instance I'. Now we show that skeleton jobs and fill jobs work as intended:

Lemma 3 *In any feasible schedule of instance I' skeleton job e_{2D-2} must be scheduled at time $D'-1$. The other skeleton jobs and the fill jobs must be scheduled at - and fully cover - the time units in:*
$\left\{ \begin{array}{ll} [u_\beta, u_\beta + m + k] \cup [u_{\beta+1} - k, u_{\beta+1} - 1] & \textit{if } \beta \textit{ is critical,} \\ [u_\beta, u_\beta + m + 1] \cup \{u_{\beta+1} - 1\} & \textit{otherwise} \end{array} \right\}$, $0 \leq \beta \leq D$.

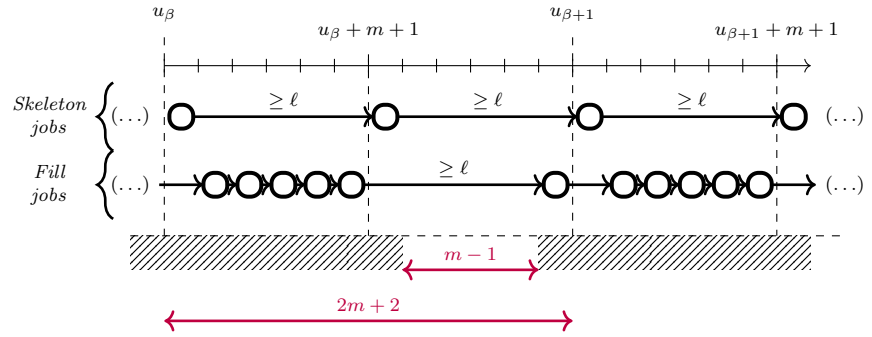

Fig. 2 Illustration of skeleton jobs and fill jobs with precedence chains in the case of a **noncritical** time unit β. The precedence relations with no label have minimum delay 0

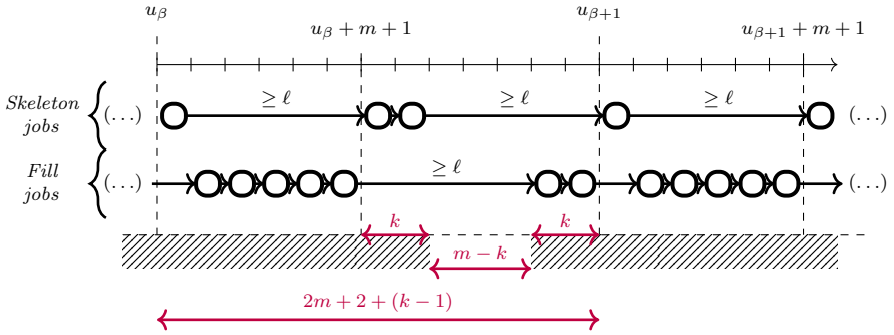

Fig. 3 Illustration of skeleton jobs and fill jobs with precedence chains in the case of a **critical** time unit β. The precedence relations with no label have minimum delay 0

Proof Let S' be a feasible schedule of I'. First we show that for all $0 \leq \beta \leq D$ skeleton job $e_{2\beta}$ is scheduled at time u_β. The definition of $prec'$ infers that $S'(e_{2\beta+2}) - S'(e_{2\beta})$ is at least $(2m+2)$ if β is not critical and $(2m+2+(k-1))$ otherwise. Then by direct induction on β it means that $e_{2\beta}$ must be scheduled at time u_β or later. By contradiction if one of these skeleton jobs is scheduled strictly later, by direct induction on β job $e_{2\beta}$ would be scheduled later than $u_D = (2m+2) \cdot D + r(kn+1)(k-1) = D' - 1$ - contradicting the feasibility of S'. Thus skeleton jobs of the form $e_{2\beta}$ are scheduled at time u_β.

Next we infer the starting time of the remaining skeleton jobs. Let $0 \leq \beta \leq D - 1$. If β is critical then, as it is shown in Figure 3, the precedence delays in $prec'$ imply that for all $1 \leq i \leq k-1$, $S'(e^{(i)}_{2\beta+1}) = u_\beta + m + i$, and $S'(e_{2\beta+1}) = u_\beta + m + k$. Otherwise, they imply that $S'(e_{2\beta+1}) = u_\beta + m + 1$ - as we can see in Figure 2.

Finally we infer the starting time of all fill jobs in a similar fashion: we show that the wanted times are the minimal ones, and that scheduling any of our jobs strictly later would make the last job in the chain to be scheduled later than $D' - 1$. We get that jobs $f_{\beta,0}$ to $f_{\beta,m-1}$ cover time interval $[u_\beta + 1, u_\beta + m]$. If β

is critical then, as it is shown in Figure 3, for all $1 \leq i \leq k-1$, $S'(f_\beta^{(i)}) = u_\beta + 2m + i = u_{\beta+1} - (k+1-i)$, and $S'(f_{\beta,m}) = u_\beta + 2m + k = u_{\beta+1} - 1$. Otherwise $S'(f_{\beta,m}) = u_\beta + 2m + k = u_{\beta+1} - 1$ - see Figure 2. □

This means that within each time interval $[u_\beta, u_{\beta+1})$ only the time units in $[u_\beta + m + k + 1, u_\beta + 2m]$ are available for other jobs if β is critical and in $[u_\beta + m + 2, u_\beta + 2m]$ otherwise. This successfully leaves exactly $m - k$ consecutive available time units if β is critical and $m - 1$ of them otherwise.

We now define in instance I' the chain of check jobs and counter sequence jobs:

Definition 9 To skeleton chain and fill jobs chain are added in I':

- counter sequence jobs $\tilde{c}_t^{(i)}$ and check jobs $\tilde{d}_t^{(i)}$ replicas of the jobs $c_t^{(i)}, d_t^{(i)}$ given in Definition 6.
- k chains with counter sequence jobs and check jobs:
  - $\begin{array}{ll} \tilde{c}_t^{(i)} \xrightarrow{\geq 0} \tilde{d}_t^{(i)} \xrightarrow{\geq \ell} \tilde{c}_{t+1}^{(i)} & \text{if } \tilde{d}_t^{(i)} \text{ exists,} \\ \tilde{c}_t^{(i)} \xrightarrow{\geq \ell} \tilde{c}_{t+1}^{(i)} & \text{otherwise} \end{array}, 1 \leq i \leq k, 0 \leq t \leq D - n - 2$

We check that our parameters ℓ_{max} and w are respectively bounded by a function of m in instance I'. ℓ is equal to m. Plus there are exactly $k+2 = \frac{m+3}{2}$ chains in I': k chains with counter sequence job and check job replicas, one with the skeleton jobs and one with the fill jobs. So both parameters are bounded by a function of m.

To complete the proof of Theorem 2, we prove that the definition of I' is a valid reduction:

Proposition 9 *Instance I is feasible if and only if instance I' is feasible.*

Proof Name the machines from 1 to m arbitrarily.

($\Longleftarrow$) Let S' be a feasible schedule of I'. By Lemma 3 all counter sequence job (resp. check job) replicas $\tilde{c}_t^{(i)}$ (resp. $\tilde{d}_t^{(i)}$) are scheduled in time intervals of the form $[u_\beta + m + k + 1, u_\beta + 2m]$ when $\beta \in [0, D-1]$ is critical and in $[u_\beta + m + 2, u_\beta + 2m]$ otherwise. So their starting time is of the form $u_{\beta'} + 2m - j'$ for some $\beta' \in [0, D-1]$, $j' \in [1, m-k]$ if β' is critical and $j' \in [1, m-1]$ otherwise.

We propose schedule S of instance I where every original job $c_t^{(i)}$ or $d_t^{(i)}$ is scheduled at time β' on machine j'. And all time sequence jobs and time indicators are scheduled at their only acceptable time unit. Since S' is feasible there are at most $m-k$ jobs scheduled at any critical time and at most $m-1$ jobs at any other time unit in S. Plus *prec* restricted to selector paths and gadgets is included in $prec'$, so all precedence constraints are met in S. We conclude that S is feasible.

($\Longrightarrow$) Let S be a feasible schedule of I. Without loss of generality we can assume that, in schedule S, a check job $d_t^{(i)}$ is always performed by a machine of lower index than the corresponding counter sequence job $c_t^{(i)}$. We propose

schedule S' of instance I' where if job $c_t^{(i)}$ (resp. $d_t^{(i)}$) is scheduled at time β on machine j' in S then replica $\tilde{c}_t^{(i)}$ (resp. $\tilde{d}_t^{(i)}$) is scheduled at time $u_\beta + 2m - j'$. And all skeleton/fill jobs are scheduled at their only acceptable time unit accordingly to the proof of Lemma 3.

Since S is feasible all replicas $\tilde{c}_t^{(i)}, \tilde{d}_t^{(i)}$ are scheduled at different times and they do not conflict with skeleton/fill jobs. Then according to *prec* only the precedence constraints of the form $\tilde{c}_t^{(i)} \xrightarrow{\geq 0} \tilde{d}_t^{(i)}$ remain to be checked. Such precedence relations were taken care of by our hypothesis on schedule S at the beginning. We conclude that S' is feasible. □

By combining Propositions 8 and 9 this concludes the XNLP-hardness proof of the corresponding parameterized problem.

7 A Fixed-parameter Tractable Algorithm Parameterized by the Maximum Delay Value Combined with the Slack

In this section we consider problem $1|prec(\ell^{min}, \ell^{max}), r_j, d_j|\star$ where all three precedence delay types are available and each job has a time window $[r_j, d_j)$. This problem is para-NP-hard with respect to slack σ - see e.g. Mallem et al. (2022) - or maximum delay value ℓ_{max} - see Engels (2000) in the case of minimum delays and Section 3 in the case of exact delays. We show that this problem becomes fixed-parameter tractable when both parameters are combined.

Theorem 3 *$1|prec(\ell^{min}, \ell^{max}), r_j, d_j|\star$ (i.e. with all three precedence delay types combined) parameterized by $\sigma + \ell_{max}$ is in FPT.*

We consider here an instance I of the decision problem $1|prec(\ell_{i,j}), r_j, d_j|\star$ assuming a general precedence graph, precedence delays of all three types in the instance and any processing time of jobs. Stage α of the dynamic programming scheme corresponds to the α^{th} task to be scheduled in order from left to right. In a state u of stage α the following items are stored:

(a) the α^{th} task to be scheduled, denoted by $j(u)$;
(b) its completion time $c(u) = C_{j(u)}$;
(c) the set $A(u)$ of scheduled tasks i for which $d_i > c(u)$;
(d) a partial schedule $S(u)$ of time interval $[c(u) - \ell_{max}, c(u))$, describing all jobs that complete in this interval and their completion times.

Definition 10 A state u is valid if there exists a feasible schedule of all jobs of $A(u) \cup \{j(u)\} \cup \{i, d_i \leq c(u)\}$ that coincides in its last time interval with $S(u)$. We denote by $\mathcal{V}_\alpha$ the set of valid states of stage α and by N_α the subset of jobs $j(u)$ for which there exists a valid state u of stage α.

We now define the transitions between consecutive stages, which correspond to the edges of a multi-stage graph G_I. Let u be a valid state of stage $\alpha - 1$. To build a valid successor v of stage α we:

- choose $j(v)$ such that $j(v)$ is unscheduled, has no predecessor, or any predecessor k of $j(v)$ is completed before $c(u)$ in u: $k \in A(u)$ or $d_k \leq c(u)$;
- choose a completion time $c(v)$ for $j(v)$ within the time interval of $j(v)$ satisfying precedence constraints and resource requirements;
- check that a job k with $d_k \leq c(v)$ does not remain unscheduled (otherwise v is discarded);
- then $A(v)$ and $S(v)$ can be deduced from $j(v)$, $A(u)$ and $S(u)$.

Definition 11 Our **dynamic programming algorithm** builds the graph stage by stage, starting from an initial state with an empty schedule. At each stage $\alpha - 1$ we consider each state u and generate only valid successors of u in stage α. Once a successor v is generated by appending a task, the algorithm checks whether the state has already been created, and if not it appends v to the list of states of stage α. The instance is feasible if and only if there is a valid state of stage n.

Let us now analyze the complexity of this algorithm, by bounding the number of nodes and arcs and the construction of successors of a state. We first assume that a preprocessing is done in $\mathcal{O}(n \cdot \log(n))$ that sorts the release times and deadlines by nondecreasing order. Then for each date the set of available tasks is computed. There are $\mathcal{O}(\sigma \cdot n)$ dates with a nonempty set of tasks. All these sets can be computed in time $\mathcal{O}(\sigma \cdot n)$ by going through the previously computed list of ordered release date and deadline values. Finally we also assume that for each deadline t, the set of the at most $\mu + 1$ tasks that crosses time unit $t - 1$ has been preprocessed in time $\mathcal{O}(\mu \cdot n)$.

7.1 Number of Tasks Crossing a Time Interval

We first prove the following lemma which bounds the number of tasks that are relevant to a time interval in a feasible instance.

Lemma 4 *Consider a feasible instance of* $1|prec(\ell_{i,j}), r_j, d_j|\star$. *Given a time interval of length* T, *the number of tasks whose time window intersect with this interval is bounded by* $2\mu + T$.

Proof Let $[t, t + T)$ be this interval. If $T = 1$ then the result holds by the definition of pathwidth μ. Now suppose $T \geq 2$. Given a task j whose time window $[r_j, d_j)$ intersects $[t, t + T)$, at least one time unit in $[t, t + T)$ must be included in $[r_j, d_j)$.

- If time t or time $t + T - 1$is included in $[r_j, d_j)$: by the definition of pathwidth μ the number of such tasks is bounded by $\mu + 1$.
- Otherwise, if a time $t' \in (t, t + T - 1)$ is included in $[r_j, d_j)$: the whole time window of task j must be included in time interval $[t + 1, t + T - 2]$. By the pigeonhole principle the number of such tasks is bounded by the length of the interval, which is $T - 2$, if the instance is feasible.

In total the number of such tasks is bounded by $(\mu+1)+(\mu+1)+(T-2)$, which sums up to $2\mu+T$. □

We now show that in a feasible instance the number of valid couples (α, j) is in $\mathcal{O}((\mu+\sigma)\cdot n)$.

Lemma 5 $\sum_{1\leq\alpha\leq n}|N_\alpha| \leq [2\mu+\sigma+1]\cdot n$.

Proof We show that each task appears in at most $2\mu+1+\sigma$ stages. We do so by bounding the indices α where a task j can appear.

- Lower bound: let Y_j be the set of tasks with a deadline not greater than r_j. Then in any feasible schedule all these tasks must be scheduled before j. This means that α must be greater than $|Y_j|$.
- Upper bound: let i be a task scheduled before j in some feasible schedule. Then i must be completed before the latest starting time of j, which is bounded by $r_j+\sigma$. Now, if $i \notin Y_j$ then d_i must be greater than r_j. This means that the time window of task i intersects with time interval $[r_j, r_j+\sigma)$ of length σ. By Lemma 4 the number of such task i not included in Y_j is bounded by $2\mu+\sigma$. Thus in any feasible schedule task j must appear in a stage α not greater than $|Y_j|+2\mu+\sigma+1$.

Hence a task j can appear in at most $2\mu+\sigma+1$ different stages α. This means that each task is included in at most $2\mu+\sigma+1$ different sets N_α. This bounds the sum of their sizes by the wanted value. □

Then we use Lemma 5 to bound the total number of states:

Proposition 10 *The number of valid states of stage α satisfies:*

$$|\mathcal{V}_\alpha| \leq |N_\alpha|\cdot f(\sigma, \ell_{max})$$

where $f(\sigma,\ell_{max}) = (\sigma+1)\cdot 2^{2\sigma+1}\cdot(4\sigma+\ell_{max})^{\ell_{max}+1}$. The total number of states is bounded by $(5\sigma+1)\cdot f(\sigma,\ell_{max})\cdot n$.

Proof First we prove the first part of the proposition part by part:

(a) By definition of N_α the number of possible tasks $j(u)$ of a state u of stage α is bounded by $|N_\alpha|$.
(b) For a given $j(u)$, by definition of slack σ there are at most $\sigma+1$ possible completion times for $j(u)$.
(c) If a job i is in $A(u)$, it was scheduled before $j(u)$ but time slot $c(u)$ is part of $[r_i, d_i)$.By the definition of pathwidth μ there can only be at most $\mu+1$ of such jobs. This bounds the number of possible sets $A(u)$ by $2^{\mu+1}$.
(d) A partial schedule of length ℓ_{max} is stored. Thus once $c(u)$ is set (from parts (a) and (b)), by Lemma 4 there are at most $2\mu+\ell_{max}$ different tasks which can appear in these partial schedules. For each of these tasks there are at most $\ell_{max}+1$ choices: either not be part of the partial schedule or have its completion time at one of the ℓ_{max} times in this interval. Thus the number of such partial schedules is bounded by $(2\mu+\ell_{max})^{\ell_{max}+1}$.

Now if we add them up with all stages α, we get the same expression, except with $(\sum_{1\leq\alpha\leq n}|N_\alpha|)$ as a factor instead of $|N_\alpha|$. By Lemma 5 this sum is bounded by $(2\mu+\sigma+1)\cdot n$. Thus the total number of states is bounded by $[2\mu+\sigma+1]\cdot(\sigma+1)\cdot 2^{\mu+1}\cdot(2\mu+\ell_{max})^{\ell_{max}+1}\cdot n$. Finally applying Result 1 to the three occurrences of μ in this expression concludes the proof. □

7.2 Successor Generation

We show that the outdegree of any node in G_I is bounded and establish the complexity of generating successors of a state.

Lemma 6 *If u is a valid state of stage $\alpha-1$.*

- *There are at most $\mu+1$ jobs that can be chosen as $j(v)$ for a valid successor v of u. The set of these jobs can be computed in $\mathcal{O}(\mu\cdot\log(\mu))$ time.*
- *There are at most $\sigma+1$ possible values of $c(v)$ once $j(v)$ is chosen.*
- *Checking whether the choice $(j(v),c(v))$ is consistent with time windows and precedence constraints induced by u and that v is a valid state can be done in $\mathcal{O}((\ell_{max}+\sigma)\log(\sigma))$ time.*
- *Comparing v to already created states can be done in $\mathcal{O}(log(|\mathcal{V}_\alpha|))$ time.*

So, the generation of all valid successors of a valid state u of stage $\alpha-1$ is in $\mathcal{O}\left(log(|\mathcal{V}_\alpha|)\cdot\sigma^2(\ell_{max}+\sigma)\cdot\log(\sigma)\right)$ time.

Proof Let us consider the least deadline d_{j_0} of an unscheduled job j_0 . To compute d_{j_0} it is sufficient to search within the sorted deadlines of jobs starting from the one next to $c(u)$ (which can be stored with u). We can search sequentially the next deadlines from $c(u)$ and for each such task k decide whether it is unscheduled (checking if $r_k\geq c(u)$ (it is unscheduled) or if it is in $A(u)$). Checking if a job is in $A(u)$ can be done in $\mathcal{O}(log(\mu))$ with appropriate data structure. At most $\mu+1$ jobs will be checked before finding an unscheduled job if any (since all scheduled jobs have a time window crossing $c(u)$).

Once j_0 is found, if another candidate i was chosen instead to extend u, then it would be completed before j_0 in a valid schedule. Thus $r_i<d_{j_0}$. Plus by the definition of j_0 we also have $d_{j_0}\leq d_i$. This means that time $d_{j_0}-1$ is included in the time window of all candidates i. By the definition of pathwidth μ this bounds the number of candidates to extend u by $\mu+1$. Our pre-processing then gives the set of candidates.

Assume we have chosen a job $j(v)$ among these candidates. There are then at most $\sigma+1$ possible starting times for $j(v)$. Let us choose $c(v)$ among these possibilities. Notice that we can scan the possible interval by storing at each step the next and previous deadline of $c(v)$. We now have to check the remaining wanted properties.

1. $j(v)$ is unscheduled. As mentioned previously this costs $\mathcal{O}(\log(\mu))$ time.
2. $d_{j(v)}-p_{j(v)}\geq\max(r_{j(v)},c(v))$.

3. All jobs i with $d_i \leq c(v)$ have been scheduled: we just have to check the deadlines from $c(v)$ by decreasing order to $c(u)$, and check whether all such jobs i are in $A(u)$. At most $\mu + 1$ such jobs have been already scheduled since then $c(u)$ is in their time window, so the check will take at most $\mathcal{O}(\mu \log(\mu))$ time.
4. We must check that all predecessors of $j(v)$ have been scheduled. If a predecessor k of $j(v)$ was unscheduled then according to the previous check it would satisfy $d_k > c(v)$. We can assume that deadlines and release times are consistent with the precedence relations - i.e. if $a \rightarrow b$ then $r_a \leq r_b$ and $d_a \leq d_b$. So we would have $r_k \leq r_{j(v)} < c(v) < d_k \leq d_{j(v)}$. Scanning the deadlines from $c(v)$ to $d_{j(v)}$ we can check whether the corresponding jobs are predecessors (with minimal and exact delays) of $j(v)$ and belong to $A(v)$. according to Lemma 4 there are at most $2(\mu + 1) + \sigma$ such tasks. The time complexity of this check is $\mathcal{O}((2(\mu + 1) + \sigma) log(\mu))$.
5. We must satisfy precedence constraints on $j(v)$. Notice that if k completes before $c(u) - \ell_{max}$, either the constraints cannot be satisfied for exact and maximum delay, or they do not impact the starting time of $j(v)$ for minimum delay.
   So we can consider the list of jobs k in $S(u)$, check (in $\mathcal{O}(1)$ time) whether k is an ancestor of j and compute $t_k = C_k + p(k) + \ell_{k,j(v)}$. If the delay is exact (resp. minimum, maximum) we check whether $c(v) = t_k$ (resp. $c(v) \geq t_k$, $c(v) \leq t_k$). Finally we check that all ancestors of $j(v)$ associated with a maximum or exact delay have been visited - otherwise this would lead to a feasible schedule. All this can be done in polynomial time.
6. Finally, checking whether a state v has already been created suppose that all states in $\mathcal{V}_\alpha$ are stored in an appropriate data structure, using an encoding of the state. Then this check could be done in $\mathcal{O}(log(|\mathcal{V}_\alpha|))$ time.

Then bounding μ with 2σ using Result 1 concludes the proof. □

We now establish that the algorithm is fixed parameter tractable.

Proposition 11 *The proposed dynamic programming algorithm is correct and runs in fixed-parameter tractable time with respect to parameters σ and ℓ_{max}. Its time complexity is:*

$$\mathcal{O}^*((\sigma + \ell_{max})^6 \cdot log^2(\sigma + \ell_{max}) \cdot 2^{2\sigma+1} \cdot (4\sigma + \ell_{max})^{\ell_{max}+1}).$$

Proof Correctness is ensured by the definition of valid states and by Lemma 6. Now let us consider the time complexity of our algorithm. From Proposition 10 we know that $|\mathcal{V}_\alpha| \leq |N_\alpha| f(\sigma, \ell_{max})$. And the algorithm generates successors of all sates of a stage. We can then, according to Lemma 6, state that the time complexity of the dynamic programming algorithm is in:

$$\mathcal{O}^* \left(\sigma^2 \cdot (\ell_{max} + \sigma) \cdot \log(\sigma) \cdot (\sum_{\alpha=1}^{n} |\mathcal{V}_{\alpha-1}| log(|\mathcal{V}_\alpha|)) \right).$$

But $log(|\mathcal{V}_\alpha|) \leq \log(|N_\alpha|) + log(f(\sigma, \ell_{max}))$. We deduce that:

$$\sum_{\alpha=1}^{n} |\mathcal{V}_{\alpha-1}| log(|\mathcal{V}_\alpha|) \leq$$
$$f(\sigma, \ell_{max}) \cdot \left((\sum_{\alpha=1}^{n} |N_{\alpha-1}| log(|N_\alpha|)) + log(f(\sigma, \ell_{max})) \cdot (\sum_{\alpha=1}^{n} |N_{\alpha-1}|) \right)$$

Now, $\sum_{\alpha=1}^{n} |N_{\alpha-1}| log(|N_\alpha|) \leq (\sum_{\alpha=1}^{n} |N_\alpha|)^2$ And we know from Lemma 5 that this is bounded by $(5\sigma+1)^2 \cdot n^2$. So we get the overall time complexity $\mathcal{O}(h(\sigma, \ell_{max}) \cdot n^2)$ by setting $h(\sigma, \ell_{max}) =$

$$\big(\sigma^2 \cdot (\ell_{max} + \sigma) \cdot \log(\sigma)\big) \cdot f(\sigma, \ell_{max}) \cdot [(5\sigma+1)^2 + (5\sigma+1) \cdot \log(f(\sigma, \ell_{max}))].$$

Now $log(f(\sigma, \ell_{max}))$ is in $\mathcal{O}((\sigma + \ell_{max}) \cdot \log(\sigma + \ell_{max}))$, so that:

$$h(\sigma, \ell_{max}) \leq \Delta \cdot (\sigma + \ell_{max})^6 \cdot log^2(\sigma + \ell_{max}) \cdot 2^{2\sigma+1} \cdot (4\sigma + \ell_{max})^{\ell_{max}+1}$$

where Δ is a constant. Notice that the bounding done here is quite raw, since the aim is not to have the most accurate complexity but only to prove that the algorithm runs in fixed-parameter tractable time.

□

8 Conclusion

In this paper we investigated the parameterized complexity of single machine scheduling problems with bounded precedence delay thresholds. We studied three delay types - exact, maximum and minimum - both in the cases of a general precedence graph and of chains of precedence. We considered the maximum delay threshold value ℓ_{max} as a parameter, both as is and combined with either the width w of the precedence graph or the slack σ of the tasks in the input instance. With parameter ℓ_{max}, in the general precedence graph case, we showed the problem to be difficult for all three delay types even with unit-time tasks. In the case of precedence chains, we unveiled key differences between the delay types on some subproblems. Then, with parameter (ℓ_{max}+w), we established that most subproblems with exact delays become easier, while with minimum delays they remain XNLP-complete. Finally, with parameter ($\ell_{max} + \sigma$) and the addition of job time windows to the problem, we proposed a fixed-parameter tractable algorithm with all three delay types combined.

Although significant progress has been made in the parameterized landscape of our problem, some key parts of it remain open. First, with parameter ℓ_{max} and minimum delays, it is unclear for which classes the cases with unit-time jobs are complete. In fact, it is a longstanding open question whether these subproblems are para-NP-complete with respect to ℓ_{max} - see e.g. Engels (2000). Although we made a breakthrough by showing XNLP-hardness, this is yet to be answered.

Second, when it comes to parameter $(\ell_{max} + w)$, the analysis could be refined for exact and maximum delays - namely we only obtained membership results. With exact delays, we believe that the cases shown to be in W[1] are actually W[1]-complete. This would make them equivalent to the problem studied by Bodlaender and van der Wegen (2020), essentially showing that adding chain time windows and allowing unbounded delays in unary to our problem would not inherently change its parameterized complexity with respect to w. With maximum delays, recalling its proximity with k-DIRECTED BANDWIDTH and the fact that k-BANDWIDTH was recently proved XNLP-complete (Bodlaender et al., 2022), we expect our problem to be XNLP-complete too.

Lastly, although we managed to yield a fixed-parameter tractable algorithm with parameter $(\ell_{max}+\sigma)$, it is unclear whether a similar approach would work with parameter $(\ell_{max} + \mu)$. Indeed the main challenge would be to circumvent the fact that the number of possible starting times per job is no longer bounded - a key hypothesis to make the algorithm fixed-parameter tractable. If successful, this would complement the fixed-parameter tractable algorithms found on the same problem but with no precedence delays with respect to μ (Hanen et al., 2022) and on parallel identical machines with unit-time jobs with respect to $(\ell_{max} + \mu)$ (Mallem et al., 2022).

Conflict of Interest

The authors have no competing interests to declare that are relevant to the content of this article. No funding was received for conducting this study.

Data Availability Statement

This manuscript has no associated data.

References

Baart R, de Weerdt M, He L (2021) Single-machine scheduling with release times, deadlines, setup times, and rejection. European Journal of Operational Research 291(2):629–639, number: 2 Publisher: Elsevier

Baptiste P (2010) A note on scheduling identical coupled tasks in logarithmic time. Discrete Applied Mathematics 158(5):583–587

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

van Bevern R, Niedermeier R, Suchý O (2017) A parameterized complexity view on non-preemptively scheduling interval-constrained jobs: few machines, small looseness, and small slack. Journal of Scheduling 20:255–265, DOI https://doi.org/10.1007/s10951-016-0478-9

Bodlaender HL (2021) Parameterized complexity of bandwidth of caterpillars and weighted path emulation. In: Graph-Theoretic Concepts in Computer Science: 47th International Workshop, WG 2021, Warsaw, Poland, June 23–25, 2021, Revised Selected Papers 47, Springer, pp 15–27

Bodlaender HL, van der Wegen M (2020) Parameterized complexity of scheduling chains of jobs with delays. In: 15th International Symposium on Parameterized and Exact Computation (IPEC)

Bodlaender HL, Groenland C, Nederlof J, Swennenhuis CM (2022) Parameterized problems complete for nondeterministic FPT time and logarithmic space. In: 2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS), IEEE, pp 193–204

Brucker P, Hilbig T, Hurink J (1999) A branch and bound algorithm for a single-machine scheduling problem with positive and negative time-lags. Discrete Applied Mathematics 94(1-3):77–99

Bruno, Jones, So K (1980) Deterministic scheduling with pipelined processors. IEEE Transactions on Computers C-29(4):308–316, conference Name: IEEE Transactions on Computers

Chen B, Zhang X (2021) Scheduling coupled tasks with exact delays for minimum total job completion time. J Sched 24(2):209–221

Elberfeld M, Stockhusen C, Tantau T (2015) On the space and circuit complexity of parameterized problems: classes and completeness. Algorithmica 71:661–701

Engels DW (2000) Scheduling for Hardware/Software Partitioning in Embedded System Design. PhD thesis, Massachusetts Institute of Technology

Finta L, Liu Z (1996) Single machine scheduling subject to precedence delays. Discrete Applied Mathematics 70(3):247–266

Flum J, Grohe M (1998) Parameterized Complexity Theory. Springer

Garey MR, Graham RL, Johnson DS, Knuth DE (1978) Complexity results for bandwidth minimization. SIAM Journal on Applied Mathematics 34(3):477–495

Graham RL, Lawler EL, Lenstra JK, Kan AR (1979) Optimization and approximation in deterministic sequencing and scheduling: a survey. In: Annals of Discrete Mathematics, vol 5, Elsevier, pp 287–326

Hanen C, Mallem M, Munier-Kordon A (2022) Parameterized complexity of single-machine scheduling with precedence, release dates and deadlines. In: 15th Workshop on Models and Algorithms for Planning and Scheduling Problems (MAPSP)

Jansen K, Kratsch S, Marx D, Schlotter I (2013) Bin packing with fixed number of bins revisited. Journal of Computer and System Sciences 79(1):39–49

Khatami M, Salehipour A, Cheng T (2020) Coupled task scheduling with exact delays: literature review and models. European Journal of Operational Research 282(1):19–39

Khatami M, Oron D, Salehipour A (2023) Scheduling coupled tasks on parallel identical machines. Optimization Letters pp 1–13

Leung JT, Vornberger O, Witthoff J (1984) On some variants of the bandwidth minimization problem. SIAM J Comput 13(3):650–667

Mallem M, Hanen C, Munier-Kordon A (2022) Parameterized complexity of a parallel machine scheduling problem. In: 17th International Symposium on Parameterized and Exact Computation (IPEC)

Mnich M, van Bevern R (2018) Parameterized complexity of machine scheduling: 15 open problems. Computers & Operations Research 100:254–261

Nederlof J, Swennenhuis CMF (2022) On the fine-grained parameterized complexity of partial scheduling to minimize the makespan. Algorithmica 84(8):2309–2334, DOI 10.1007/S00453-022-00970-8, URL https://doi.org/10.1007/s00453-022-00970-8

Savitch WJ (1970) Relationships between nondeterministic and deterministic tape complexities. Journal of Computer and System Sciences 4(2):177–192, DOI https://doi.org/10.1016/S0022-0000(70)80006-X, URL https://www.sciencedirect.com/science/article/pii/S002200007080006X

Swennenhuis CMF (2022) Fine-grained parameterized complexity of scheduling and sequencing problems. PhD thesis, Eindhoven University of Technology, URL https://pure.tue.nl/ws/portalfiles/portal/228924206/20221202_Swennenhuis_hf.pdf