Parameterized problems complete for nondeterministic FPT time and logarithmic space

Bodlaender, Hans L.; Groenland, Carla; Nederlof, Jesper; Swennenhuis, Céline

DOI
10.1016/j.ic.2024.105195

Publication date
2024

Document Version
Final published version

Published in
Information and Computation

Citation (APA)
Bodlaender, H. L., Groenland, C., Nederlof, J., & Swennenhuis, C. (2024). Parameterized problems complete for nondeterministic FPT time and logarithmic space. *Information and Computation*, *300*, Article 105195. https://doi.org/10.1016/j.ic.2024.105195

Information and Computation 300 (2024) 105195

Contents lists available at ScienceDirect

Information and Computation

journal homepage: www.elsevier.com/locate/yinco

Parameterized problems complete for nondeterministic FPT time and logarithmic space ☆

Hans L. Bodlaender [a,*], Carla Groenland [b,1], Jesper Nederlof [a,2], Céline Swennenhuis [c,3]

[a] Department of Information and Computing Sciences, Utrecht University, the Netherlands
[b] Faculty of Electrical Engineering, Mathematics and Computer Science, Technical University Delft, the Netherlands
[c] Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, the Netherlands

ARTICLE INFO

Article history:
Received 6 November 2023
Received in revised form 8 July 2024
Accepted 16 July 2024
Available online 19 July 2024

ABSTRACT

Let XNLP be the class of parameterized problems such that an instance of size n with parameter k can be solved nondeterministically in time $f(k)n^{O(1)}$ and space $f(k)\log(n)$ (for some computable function f). We give a wide variety of XNLP-complete problems, such as List Coloring and Precoloring Extension with pathwidth as parameter, Scheduling of Jobs with Precedence Constraints, with both number of machines and partial order width as parameter, Bandwidth and variants of Weighted CNF-Satisfiability. In particular, this implies that all these problems are W[t]-hard for all t.

1. Introduction

Already since the 1970s, an important paradigm in classical complexity theory has been that an increased number of alternations of existential and universal quantifiers increases the complexity of search problems: This led to the central definition of the polynomial hierarchy [51], whose study resulted in cornerstone results in complexity theory such as Toda's theorem and lower bounds for time/space tradeoffs for SAT [3]. In their foundational work in the early 1990s, Downey and Fellows introduced an analogue of this hierarchy for parameterized complexity theory, called the W-*hierarchy*. This hierarchy comprises the complexity classes FPT, the parameterized analogue of P, W[1], the parameterized analogue of NP, and the classes W[2], …, W[P], XP (see e.g. [26–28]).

While in the polynomial hierarchy only the classes with no quantifier alternation (i.e. P, NP and coNP) are prominent, many natural parameterized problems are known to be hard or even complete for W[i] for some $i \geq 1$. Thus, the W-hierarchy substantially differentiates the complexity of hard parameterized problems. And such a differentiation has applications outside parameterized complexity as well: For example, for problems in W[1] we can typically improve over

☆ This paper contains the results reported in [18], with the exception of results on reconfiguration, and one result from [6] (the reduction in the proof of Theorem 24).

* Corresponding author.
E-mail addresses: h.l.bodlaender@uu.nl (H.L. Bodlaender), c.e.groenland@tudelft.nl (C. Groenland), j.nederlof@uu.nl (J. Nederlof), c.m.f.swennenhuis@tue.nl (C. Swennenhuis).

[1] This research was done when Carla Groenland was associated with Utrecht University.

[2] The work of Jesper Nederlof was supported by the project CRACKNP that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 853234).

[3] The work of Céline Swennenhuis was supported by the Netherlands Organisation for Scientific Research under project no. 613.009.031b.

https://doi.org/10.1016/j.ic.2024.105195

Table 1
An overview of XNLP-complete problems is given. For problems marked with +, the source also gives an XNLP-hardness or completeness proofs for variants of the stated problem. We use the abbreviations CL = Clique, IS = Independent Set, DS = Dominating Set, pw = parameterized by pathwidth.

Problem	Source
Longest Common Subsequence +	[32]
Timed Non-determ. Cellular Automaton +	[32]; Subsection 2.5
Chained CNF-Satisfiability +	Subsection 3.1
Chained Multicolored Clique	Subsection 3.2
Binary CSP pw +	Subsection 3.3
Accepting NNCCM	Subsection 3.4
List Coloring pw +	Subsection 4.1
Log-Pathwidth DS, IS	Subsection 4.2
Scheduling with precedence constaints	Subsection 4.3
Uniform Emulation of Weighted Paths	Subsection 4.4
Bandwidth	Subsection 4.5
Acyclic Finite State Automata Intersection	[53]; Subsection 4.6

brute-force enumeration algorithms, while for W[2]-complete problems we can prove lower bounds under the Strong Exponential Time Hypothesis excluding such improvements (see e.g. the discussion in [1]).[4]

For many problems, completeness for a class is known, e.g., Clique is W[1]-complete [27] and Dominating Set is W[2]-complete [26]. However, there are also several problems known to be hard for W[1], W[2], or even for W[t] for all positive integers t, but which are not known to be in the class W[P]; in many cases, only membership in XP was known. For such problems, it is an intriguing question to establish their exact position within the W-hierarchy as it can be expected to shed light on their complexity similarly as it did for the previous problems.

One example of such a problem is the Bandwidth problem. It has been known to be hard for all classes W[t] since 1994 [12]. Already in the midst of the 1990s, Hallett argued that it is unlikely that Bandwidth belongs to W[P], see the discussion by Fellows and Rosamond in [35]. The argument intuitively boils down to the following: Bandwidth "seems" to need certificates with $\Omega(n)$ bits, while problems in W[P] have certificates with $O(f(k)\log n)$ bits. A similar situation applies to several other W[1]-hard problems.

A (largely overlooked) breakthrough was made a few years ago by Elberfeld, Stockhusen and Tantau [32], who studied several classes of parameterized problems, including a class which they called N[f poly, f log]. This class is defined as the set of parameterized problems that can be solved with a non-deterministic algorithm with simultaneously, the running time bounded by $f(k)n^c$ and the space usage bounded by $f(k)\log n$, with k the parameter, n the input size, c a constant, and f a computable function. Hardness for this class is proved using parameterized logspace reductions, instead of the fixed parameter tractable reductions used in the W-hierarchy (see Section 2.3). For easier future reference, we denote this class by XNLP. Elberfeld et al. [32] showed that a number of problems are complete for this class, including the Longest Common Substring problem. Since 1995, Longest Common Substring is known to be hard for all W[t] [10], but its precise parameterized complexity was unknown until the result by Elberfeld et al. [32].

Our contribution We show that the class XNLP (i.e., N[f poly, f log]) can play an important role in establishing the parameterized complexity of a large collection of well studied problems, ranging from abstract problems on different types of automata (see e.g. [32] or later in this paper), logic, graph theory, scheduling, and more.

In this paper, we give a number of different examples of problems that are complete for XNLP. These include Bandwidth, thus indirectly and partially answering the question the authors of [12] asked thirty years ago about the parameterized complexity of Bandwidth; see also the discussion in [35].

Indeed, we establish what the "natural home" is Bandwidth. Although we do not know what the relation is between W[P] and XNLP, we expect that XNLP is in some sense "orthogonal" to the W-hierarchy.

In Table 1, we list the problems shown to be XNLP-complete in either this paper or by Elberfeld et al. [32].

Fig. 1 shows for the problems from which problem the reduction starts to show XNLP-hardness.

Often, membership in XNLP can be seen by looking at the algorithm that establishes membership in XP. Many problems in XNLP typically have a dynamic programming algorithm that sequentially builds tables, with each individual table entry expressible with $O(f(k)\log n)$ bits. We then get membership in XNLP by instead of tabulating all entries of a table, guessing one entry of the next table — this step resembles the text-book transformation between a deterministic and non-deterministic finite automaton.

Interestingly, hardness for the class XNLP also has consequences for the use of memory of deterministic parameterized algorithms. Pilipczuk and Wrochna [48] conjecture that Longest Common Subsequence has no XP algorithm that runs in

[4] For example the naïve algorithm $O(n^{k+1})$ time algorithm for finding cliques on k vertices on n-vertex graphs can be improved to run in $n^{0.8k}$ time, but similar run times for Dominating Set refute the Strong Exponential Time Hypothesis.

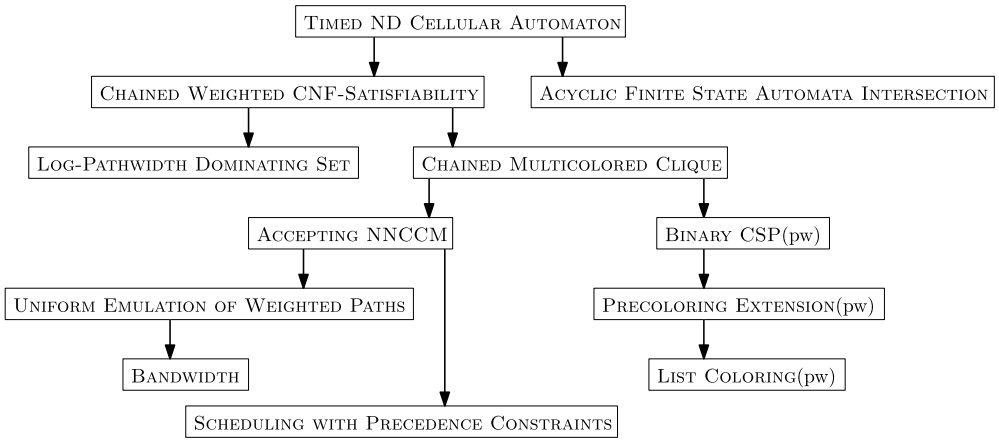

Fig. 1. Reductions between XNLP-hard problems from this paper. Several variants of problems are not shown.

$f(k)n^c$ space, for a computable function f and constant c; if this conjecture holds, then no XNLP-hard problem has such an algorithm. See Section 5 for more details.

When a problem is XNLP-hard, it is also hard for each class W[t] (see Lemma 2). Thus, XNLP-hardness proofs are also a tool to show hardness for W[t] for all t. In this sense, our results strengthen existing results from the literature: for example, List Coloring and Precoloring Extension parameterized by pathwidth (or treewidth) were known to be W[1]-hard [33], and Precedence Constraint K-processor Scheduling parameterized by the number of processors K was known to be W[2]-hard [11]. Our XNLP-hardness proofs imply hardness for W[t] for all t. Moreover, our XNLP-hardness proofs are often simpler than the existing proofs that problems are hard for W[t] for all t.

Related to the class XNLP is the class XNL: the parameterized problems that can be solved by a nondeterministic algorithm that uses $f(k)\log n$ space. There is no explicit time bound, but we can freely add a time bound of $2^{f(k)\log n}$, and thus XNL is a subset of XP. XNL can be seen as the parameterized counterpart of NL. Amongst others, XNL was studied by Chen, Flum and Grohe [23], who showed that Compact Turing Machine Computation is complete for XNL.

Hardness for a class is always defined with respect to a class of reductions. In our proofs, we use parameterized logspace reductions (or, in short, pl-reductions). A brief discussion of other reductions can be found in Subsection 5.2.

Subsequent work After this paper appeared, several other problems were shown to be XNLP-complete, and related complexity classes were defined, with their own complete problems. We briefly discuss these results here.

Recent XNLP-completeness results that build upon our work include the following:

- several graph problems with a linear structure, including problems with pathwidth, linear cliquewidth, and linear mim-width as parameter [16];
- b-coloring with pathwidth as parameter [39];
- several problems related to flow, with pathwidth as parameter [7];
- integral 2-commodity flow with pathwidth as parameter [21].

In [20], it was shown that reconfiguration of dominating sets and of independent sets, with the sizes of these sets as parameter is XNLP-complete, when the number of reconfiguration steps is given in unary. In contrast, when the number of steps is respectively a parameter, given in binary, or unspecified, the problems become complete for W[2] or W[1], for XNL, and for XL.

The class XALP was introduced by Bodlaender, Groenland, Jacob, Pilipczuk and Pilipzcuk [17] as an analogue to XNLP for "tree-structured" problems. Natural problems with a "tree structure", such as List Coloring parameterized by treewidth or Max Cut parameterized by clique-width were shown to be XALP-complete. Determining the tree partition width (or strong treewidth) of a graph was also shown to be XALP-complete in [15] and the Perfect Phylogeny problem was shown to be XALP-complete by de Vlas [25]. A probabilistic variant of XNLP with a complete problem related to Bayesian networks was introduced in [9]. Bodlaender, Groenland, and Pilipczuk [19] introduced the class XSLP, and showed some natural problems with treedepth as parameter to be complete for that class.

Paper overview In Section 2, we give a number of preliminary definitions and results. In Section 3 we introduce three new problems that are XNLP-complete. In Section 4 we then use these problems as building blocks, to prove other problems to be either XNLP-complete or XNLP-hard. For each of the problems, its background and a short literature review specific to it will be given inside its relevant subsection. Final comments and open problems are given in Section 5.

2. Preliminaries

In this section we formally define the class XNLP and give some preliminary results.

The section is organized as follows: first we introduce some basic notions in Subsection 2.1, next we formally define the class XNLP in Subsection 2.2. In Subsection 2.3 we then introduce the type of reductions that will be used in this paper and in Subsection 2.4 we list some preliminary results. Subsection 2.5 ends the section with a discussion of cellular automata, for which Elberfeld et al. [32] already established it was XNLP-complete. From this problem we will (indirectly) derive the XNLP-hardness for all other XNLP-hard problems in this paper; containment in XNLP will always be argued more directly.

2.1. Basic notions

We assume the reader to be familiar with a number of notions from complexity theory, parameterized algorithms, and graph theory. A few of these are reviewed below, along with some new and less well-known notions.

We use interval notation for sets of integers, i.e., $[a, b] = \{i \in \mathbb{Z} \mid a \leq i \leq b\}$. All logarithms in this paper have base 2. $\mathbf{N}$ denotes the set of the natural numbers $\{0, 1, 2, \ldots\}$, and $\mathbf{Z}^+$ denotes the set of the positive natural numbers $\{1, 2, \ldots\}$.

2.2. Definition of the class XNLP

In this paper, we study parameterized decision problems, which are subsets of $\Sigma^* \times \mathbf{N}$, for a finite alphabet Σ. The following notation is used, also by e.g. [32], to denote classes of (non-)deterministic parameterized decision problems with a bound on the used time and space. We use n for the input size; poly for a polynomial function in the input size (i.e. $n^{O(1)}$); log for a logarithmic function in the input size (i.e. $O(\log n)$); f for a computable function of the parameter; ∞ if there is no a priory bound for the resource.

On top of the running time of an algorithm we will also study the *space usage*. Informally, an algorithm has (random) read-only access to an input tape and random write-only access to an output tape. Additionally, it has both random read and write access to a working tape, but the length of the working tape equals the space usage.

Let $\mathrm{D}[t, s]$ denote the class of parameterized decision problems that can be solved by a deterministic algorithm in t time and s space and let $\mathrm{N}[t, s]$ be analogously defined for non-deterministic algorithms. Thus, FPT can be denoted by $\mathrm{D}[f\,\mathrm{poly}, \infty]$; we can denote XP by $\mathrm{D}[n^f, \infty]$, NP by $\mathrm{N}[\mathrm{poly}, \infty]$, L by $\mathrm{D}[\infty, \log]$, etcetera.

A special role in this paper is played by the class $\mathrm{N}[f\,\mathrm{poly}, f\,\log]$: the parameterized decision problems that can be solved by a non-deterministic algorithm that simultaneously uses at most $f(k)n^c$ time and at most $f(k)\log n$ space, on an input (x, k), where x can be encoded with n bits, f a computable function, and c a constant. Because of the special role of this class, we use the shorter notation XNLP.

XNLP is a subclass of the class XNL, which was studied by Chen et al. [23]. XNL is the class of problems solvable with a non-deterministic algorithm in $f(k)\log n$ space (f, k, n as above), i.e., XNL is the class $\mathrm{N}[\infty, f\,\log]$.

We assume the reader to be familiar with notions from parameterized complexity, such as XP, W[1], W[2], ..., W[P] (see e.g. [28]). For classes of parameterized problems, we can often make a distinction between non-uniform (a separate algorithm for each parameter value), and uniform. Throughout this paper, we look at the uniform variant of the classes, but we also will assume that functions f of the parameter in time and resource bounds are computable — this is called *strongly uniform* by Downey and Fellows, see [28, p. 25].

2.3. Reductions

Hardness for a class is defined in terms of reductions. We mainly use parameterized logspace reductions (defined below), which are a special case of fixed parameter tractable (FPT) reductions. Hardness for classes in the W-hierarchy is considered with respect to FPT reductions. The reductions are defined below based upon the formulations in [32]. Two other types of reductions are briefly discussed in the conclusion (Section 5.2.)

- A *parameterized reduction* from a parameterized problem $Q_1 \subseteq \Sigma_1^* \times \mathbf{N}$ to a parameterized problem $Q_2 \subseteq \Sigma_2^* \times \mathbf{N}$ is a function $f : \Sigma_1^* \times \mathbf{N} \to \Sigma_2^* \times \mathbf{N}$, such that the following holds.
  1. For all $(x, k) \in \Sigma_1^* \times \mathbf{N}$, $(x, k) \in Q_1$ if and only if $f((x, k)) \in Q_2$.
  2. There is a computable function g, such that for all $(x, k) \in \Sigma_1^* \times \mathbf{N}$, if $f((x, k)) = (y, k')$, then $k' \leq g(k)$.
- A *parameterized logspace reduction* or *pl-reduction* is a parameterized reduction for which there is a deterministic algorithm that computes $f((x, k))$ in space $O(g(k) + \log n)$, with g a computable function and $n = |x|$ the number of bits to denote x.
- A *fixed parameter tractable reduction* or *fpt-reduction* is a parameterized reduction for which there is a deterministic algorithm that computes $f((x, k))$ in time $O(g(k)n^c)$, with g a computable function, $n = |x|$ the number of bits to denote x and c a constant.

A deterministic algorithm that uses $O(g(k) + \log n)$ space necessarily runs in $2^{O(g(k)+\log n)}$ time which is FPT in k. Thus, if there is a pl-reduction from problem A to problem B and B is in XNLP, then problem A is in XNLP as well.

In the remainder of the paper, completeness for XNLP is with respect to pl-reductions.[5]

2.4. Preliminary results on XNLP

We give some easy observations that relate XNLP to other notions from parameterized complexity. The following easy observation can be seen as a special case of the fact that $\mathrm{N}[\infty, S(n)] \subseteq \mathrm{D}[2^{O(S(n))}, \infty]$, see [3, Theorem 4.3].

Lemma 1. XNLP *is a subset of* XP.

Proof. Using standard techniques, we can transform the non-deterministic algorithm to a deterministic algorithm that employs dynamic programming: tabulate all reachable configurations of the machine — a configuration is a tuple, consisting of the contents of the work tape, the state of the machine, and the position of the two headers. From a configuration, we can compute all configurations that can be reached in one step, and thus we can check if a configuration that has an accepting state can be reached.

The number of such configurations is bounded by the product of a single exponential of the size of the work tape (i.e., at most $2^{f(k)\log n} = n^{f(k)}$ for some computable function f), the constant number of states of the machine, and the $O(f(k)\log n) \cdot n$ number of possible pairs of locations of the heads, and thus bounded by a function of the form $n^{g(k)}$ with g a computable function. □

Lemma 2. *If a parameterized problem Q is* XNLP*-hard, then it is hard for each class* W[t] *for all $t \in \mathbf{Z}^+$.*

Proof. In Weighted t-Normalized Satisfiability, we have a Boolean formula with parenthesis-depth t and ask if we can satisfy it by setting exactly k variables to true and all others to false; we can non-deterministically guess which of the k Boolean variables are true; verifying whether this setting satisfies the formula can be done with $O(t + k\log n)$ bits of space, see e.g. [28]. Observe that the W[t]-complete problem Weighted t-Normalized Satisfiability belongs to XNLP.

Each problem in W[t] has an fpt-reduction to Weighted t-Normalized Satisfiability, and the latter has a pl-reduction (which is also an fpt-reduction) to any XNLP-hard problem Q. The transitivity of fpt-reductions implies that Q is then also hard for W[t]. □

Lemma 3 *(Chen et al. [23]). If* NL ≠ P*, then there are parameterized problems in* FPT *that do not belong to* XNL *(and hence also not to* XNLP*).*

Proof. Take a problem Q that belongs to P, but not to NL. Consider the parameterized problem Q' with $(x, k) \in Q'$ if and only if $x \in Q$. (We just ignore the parameter part of the input.) Then Q' belongs to FPT, since the polynomial time algorithm for Q also solves Q'. If Q' is in XNL, then there is an algorithm that solves Q in (non-deterministic) logarithmic space, a contradiction. So Q' belongs to FPT but not to XNL. □

Chen et al. [23] introduce the following problem.

CNTMC (Compact Nondeterministic Turing Machine Computation)
Input: the encoding of a non-deterministic Turing Machine M; the encoding of a string x over the alphabet of the machine.
Parameter: k.
Question: Is there an accepting computation of M on input x that visits at most k cells of the work tape?

Theorem 4 *(Theorem 21(2) from Chen et al. [23]).* CNTMC *is* XNL*-complete under pl-reductions.*

We expect that it is possible to show XNLP-completeness for a "timed" variant of this problem.

Timed CNTMC
Input: the encoding of a non-deterministic Turing Machine M; the encoding of a string x over the alphabet of the machine; an integer T given in unary.
Parameter: k.
Question: Is there an accepting computation of M on input x that visits at most k cells of the work tape and uses at most T time?

[5] Note we could also have defined completeness for XNLP with respect to slightly more general reductions allowing algorithms using $O(g(k)\log n)$ space and time FPT in k, since such reductions still preserve containment in XNLP. We use the more strict variant since all our reductions use only $O(g(k) + \log n)$ space.

The fact that the time that the machine uses is given in unary, is needed to show membership in XNLP. We expect XNLP-hardness can be shown for Timed CNTMC in a similar fashion to the proof of Theorem 4. Since we do not build upon the result, and the proof would be tedious, we omit the details. We instead start with a problem on cellular automata which was shown to be complete for XNLP by Elberfeld et al. [32]. We discuss this problem in the next subsection. Elberfeld et al. [32] show a number of other problems to be XNLP-complete, including a timed version of the acceptance of multihead automata, and the Longest Common Subsequence problem, parameterized by the number of strings. The latter result is discussed in the Conclusion, Section 5.1.

2.5. Cellular automata

In this subsection, we discuss one of the results by Elberfeld et al. [32]. Amongst the problems that are shown to be complete for XNLP by Elberfeld et al. [32], of central importance to us is the Timed Non-deterministic Cellular Automaton problem. We use the hardness of this problem to show the hardness of Chained CNF-Satisfiability in Subsection 3.1.

In this subsection, we describe the Timed Non-deterministic Cellular Automaton problem, and a variant. We are given a linear cellular automaton, a time bound t given in unary, and a starting configuration for the automaton, and ask if after t time steps, at least one cell is in an accepting state.

More precisely, we have a set of states S, and subset of accepting states $A \subseteq S$. We assume there are two special states s_L and s_R which are used for the leftmost and rightmost cell. A *configuration* is a function $c : \{1, \ldots, q\} \to S$, with $c(1) = s_L$, $c(q) = s_R$ and for $i \in [2, q-1]$, $c(i) \in S \setminus \{s_L, s_R\}$. We say that we have q cells, and in configuration c, cell i has state $c(i)$. The machine is further described by a collection of 4-tuples $\mathcal{T}$ in $S \times (S \setminus \{s_L, s_R\}) \times S \times (S \setminus \{s_L, s_R\})$. At each time step, each cell $i \in [2, q-1]$ reads the 3-tuple (s_1, s_2, s_3) of states given by the current states of the cells $i-1$, i and $i+1$ (in that order). If there is a cell $i \in [2, q-1]$ with $c(i) \in A$, then the machine accepts. Otherwise, if there is a cell $i \in [2, q-1]$ for which there is no 4-tuple of the form $(s_1, s_2, s_3, s_4) \in \mathcal{T}$ with $s_4 \in S$, then the machine halts and rejects. Otherwise, each of the cells $i \in [2, q-1]$ selects an $s_4 \in S$ $(s_1, s_2, s_3, s_4) \in \mathcal{T}$ (with respect to their 'own' 3-tuple (s_1, s_2, s_3)) and moves in this time step to state s_4. (In a non-deterministic machine, there can be multiple such states s_4 and a non-deterministic step is done. For a deterministic cellular automaton, for each 3-tuple (s_1, s_2, s_3) there is at most one 4-tuple $(s_1, s_2, s_3, s_4) \in \mathcal{T}$.) Note that the leftmost and rightmost cell never change state: their states are used to mark the ends of the tape of the automaton.

Timed Non-deterministic Cellular Automaton
Input: Cellular automaton with set of states S and set of transitions $\mathcal{T}$; configuration c on q cells; integer in unary t; subset $A \subseteq S$ of accepting states.
Parameter: q.
Question: Is there an execution of the machine for exactly t time steps with initial configuration c, such that at time step t at least one cell of the automaton is in A?

We will build on the following result.

Theorem 5 *(Elberfeld et al. [32]).* Timed Non-deterministic Cellular Automaton *is* XNLP*-complete.*

We recall that the class, denoted by XNLP in the current paper, is called N[f poly, f log] in [32].

Elberfeld et al. [32] state that asking if all cells are in an accepting state does not make a difference, i.e., if we modify the Timed Non-deterministic Cellular Automaton problem by asking if all cells are in an accepting state at time t, then we also have an XNLP-complete problem.

We also discuss a variant that can possibly be useful as another starting point for reductions.

Timed Non-halting Non-deterministic Cellular Automaton
Input: Cellular automaton with set of states S and set of transitions $\mathcal{T}$; configuration c on q cells; integer in unary t; subset $A \subseteq S$ of accepting states.
Parameter: q.
Question: Is there an execution of the machine for exactly t steps with initial configuration c, such that the machine does not halt before time t?

Corollary 6. Timed Non-halting Non-deterministic Cellular Automaton *is* XNLP*-complete.*

Proof. Membership in XNLP follows in the same way as for Timed Non-deterministic Cellular Automaton, see [32]; observe that we can store a configuration using $\lceil q \log |S| \rceil$ bits.

For hardness, we start from a cellular automaton from Timed Non-deterministic Cellular Automaton and adjust it as follows. We take an automaton that accepts, if and only if at time t all cells are in an accepting state. Now, we enlarge the set of states as follows: for each time step $t' \in [0, t]$, and each state $s \in S \setminus \{s_L, s_R\}$, we create a state $s^{t'}$. The initial

configuration c is modified to c' by setting $c'(i) = s^0$ for $i \in [2, q-1]$ when $c(i) = s$. We enlarge the set of transitions as follows. For each $t' \in [0, t-1]$ and $(s_1, s_2, s_3, s_4) \in \mathcal{T}$, we create a transition $(s_1^{t'}, s_2^{t'}, s_3^{t'}, s_4^{t'+1})$ in the new set of transitions. In this way, each state of the machine also codes the time: at time t' all cells except the first and last will have a state of the form $s^{t'}$.

We run the machine for one additional step, i.e., we increase t by one. We create one additional accepting state s_a. For each accepting state $s \in A$, we make transitions (x, s^t, y, s_a) for all possible values x and y can take. When $s \notin A$, then there are no x, y, z for which there is a transition of the form (x, s^t, y, z). This ensures that a cell has a possible transition at time t if and only if it is in an accepting state. In particular, when all states are accepting, all cells have a possible transition at time t; if there is a state that is not accepting at time t, then the machine halts. □

2.6. Pathwidth, bandwidth, and cutwidth

A *path decomposition* of a graph $G = (V, E)$ is a sequence $(X_1, X_2, \ldots, X_r)$ of subsets of V with the following properties.

1. $\bigcup_{1 \le i \le r} X_i = V$.
2. For all $\{v, w\} \in E$, there is an $i \in I$ with $v, w \in X_i$.
3. For all $1 \le i_0 < i_1 < i_2 \le r$, $X_{i_0} \cap X_{i_2} \subseteq X_{i_1}$.

The *width* of a path decomposition $(X_1, X_2, \ldots, X_r)$ equals $\max_{1 \le i \le r} |X_i| - 1$, and the *pathwidth* of a graph G is the minimum width of a path decomposition of G.

When considering the parameter pathwidth, we will assume that a path decomposition of width at most k is given as part of the input. It currently is an open problem whether such a path decomposition can be found with a non-deterministic algorithm using logarithmic space and "fpt" time. Kintali and Munteanu [43] show that for each fixed k, determining if the pathwidth is at most k, and if so, finding a path decomposition of width at most k belongs to L. As a subroutine, this uses a related earlier result from Elberfeld et al. [31], who showed that for each fixed k, determining if the treewidth is at most k, and if so, finding a tree decomposition of width at most k belongs to L.

A *linear ordering* of a graph $G = (V, E)$ is a bijection $f : V \to [1, |V|]$. The *bandwidth* of a linear ordering f of $G = (V, E)$ is $\max_{\{v,w\} \in E} |f(v) - f(w)|$. The *cutwidth* of a linear ordering f of $G = (V, E)$ is given by $\max_{i \in [1, |V|-1]} |\{\{v, w\} \in E \mid f(v) \le i < f(w)\}|$. The *bandwidth*, respectively *cutwidth* of a graph G is the minimum bandwidth, respectively cutwidth over all linear orderings of G.

3. Building blocks

In this section, we introduce three new problems and prove that they are XNLP-complete, namely Chained CNF-Satisfiability, Chained Multicolored Clique and Accepting NNCCM. These problems are called building blocks, as their main use is proving XNLP-hardness for many other problems (see Fig. 1).

3.1. Chained CNF-Satisfiability

In this subsection we give a useful starting point for our transformations: a variation of Satisfiability which we call Chained Weighted CNF-Satisfiability. The problem can be seen as a generalization of the W[1]-hard problem Weighted CNF-Satisfiability [26].

Chained Weighted CNF-Satisfiability
Input: r disjoint sets of Boolean variables $X_1, X_2, \ldots X_r$, each of size q; integer $k \in \mathbf{N}$; Boolean formulas F_1, F_2, …, F_{r-1}, where each F_i is an expression in conjunctive normal form on variables $X_i \cup X_{i+1}$.
Parameter: k.
Question: Is it possible to satisfy the formula $F_1 \wedge F_2 \wedge \cdots \wedge F_{r-1}$ by setting exactly k variables to true from each set X_i and all others to false?

Our main result in this subsection is the following. We will also prove a number of variations to be XNLP-complete later in this subsection.

Theorem 7. Chained Weighted CNF-Satisfiability *is* XNLP*-complete.*

Proof. Membership in XNLP is easy to see. Indeed, for i from 1 to r, we non-deterministically guess which variables in each X_i are true, and keep the indexes of the true variables in memory for the two sets X_i, X_{i+1}. Verifying $F_i(X_i, X_{i+1})$ can easily be done in $O(k + \log n)$ space and linear time.

To show hardness, we transform from Timed Non-deterministic Cellular Automaton.

For each time step $t' \in [1, t]$, each cell $r \in [1, q]$, and each state $s \in S$, we have a Boolean variable $x_{t',r,s}$ with $x_{t',r,s}$ denoting whether the rth cell of the automaton at time t' is in state s. For each time step $t' \in [1, t-1]$, each cell $r \in [2, q-1]$, and each transition $z \in \mathcal{T}$ we have a variable $y_{t',r,z}$ that expresses that cell r uses transition z at time t'.

We will build a Boolean formula of the form $\bigwedge_{i=1}^{t-1} F_i$ and partitions of the variables, such that the expression is satisfiable by setting exactly k variables to true from each set in the partition if and only if the machine can reach time step t with at least one cell in accepting state starting from the initial configuration. The partition is based on the time of the automaton: for each time step t', the set $X_{t'}$ consists of all variables of the form $x_{t',r,s}$ and $y_{t',r,s}$. For each set $X_{t'}$ we require that exactly $2q-2$ variables are set to true.

The formula has the following ingredients.

- *At each step in time, each cell has exactly one state. Moreover, it uses a transition (unless at the first or final cell).* To encode that we have at least one state, we use the expression $\bigvee_{s \in S} x_{t',r,s}$ for each $t' \in [1, t-1]$ and $r \in [1, q]$. To encode that there is at least one transition, we use the clause $\bigvee_{z \in \mathcal{T}} y_{t',r,z}$ for all $t' \in [1, t-1]$ and $r \in [2, q-1]$. Since we need to set exactly $2q-2$ variables to true from $X_{t'}$, and this is the total allowed amount, the pigeonhole principle shows that for each step in time t' and each cell r, at most (and hence, exactly) one state s exists for which variable $x_{t',r,s}$ is true. Similarly, for all cells apart from the first and last, there is exactly one transition z for which $y_{t',r,z}$ is true.
- *We start in the initial configuration.* We encode this using clauses with one literal $x_{0,r,s}$ whenever cell r has state s in the initial configuration.
- *We end in an accepting state.* This is encoded by

$$\bigvee_{s \in A} \bigvee_{r} x_{t,r,s}.$$

- *Left and right cells do not change.* We add clauses $x_{t',1,s_L}$ and x_{t',q,s_R} with one literal for all time steps t'.
- *If a cell has a value at a time $t' > 0$, then there was a transition that caused it.* This is encoded by

$$x_{t',r,s} \Rightarrow \bigvee_{(s_1,s_2,s_3,s) \in \mathcal{T}} y_{t'-1,r,(s_1,s_2,s_3,s)}.$$

  This is expressed in conjunctive normal form as

$$\neg(x_{t',r,s}) \vee \bigvee_{(s_1,s_2,s_3,s) \in \mathcal{T}} y_{t'-1,r,(s_1,s_2,s_3,s)}.$$

- *If a transition is followed, then the cell and its neighbors had the corresponding states.* For each time step $t' \in [1, t-1]$, cell $r \in [2, q-1]$ and transition $z = (s_1, s_2, s_3, s_4) \in \mathcal{T}$, we express this as

$$y_{t',r,z} \Rightarrow \left(x_{t',r-1,s_1} \wedge x_{t',r,s_2} \wedge x_{t',r+1,s_3}\right).$$

  We can rewrite this to the three clauses $\neg(y_{t',r,z}) \vee x_{t',r-1,s_1}$, and $\neg(y_{t',r,z}) \vee x_{t',r,s_2}$, and $\neg(y_{t',r,z}) \vee x_{t',r+1,s_3}$.

The last two steps ensure that the transition chosen from the y-variables agrees with the states chosen from x-variables.

It is not hard to see that we can build the formula with $f(k) + \log n$ space and polynomial time, and that the formula is of the required shape. □

A special case of the problem is when all literals that appear in the formulas F_i are positive, i.e., we have no negations. We call this special case Chained Weighted Positive CNF-Satisfiability.

Theorem 8. Chained Weighted Positive CNF-Satisfiability *is* XNLP*-complete.*

Proof. We modify the proof of the previous result. Note that we can replace each negative literal by the disjunction of all other literals from a set where exactly one is true, i.e., we may replace $\neg(x_{t',r,s})$ and $\neg(y_{t',r,z})$ by

$$\bigvee_{s' \neq s} x_{t',r,s'} \text{ and } \bigvee_{z' \neq z} y_{t',r,z'}$$

respectively. The modification can be carried out in logarithmic space and polynomial time, and gives an equivalent formula. Thus, the result follows. □

A closer look at the proof of Theorem 7 shows that $F_2 = F_3 = \cdots = F_{r-2}$, and more specifically, we have a condition on X_1, a condition on X_r, and identical conditions on all pairs $X_i \cup X_{i+1}$ with i from 1 to $r-1$. Thus, we also have XNLP-completeness of the following special case:

Preregular Chained Weighted CNF-Satisfiability

Input: r sets of Boolean variables $X_1, X_2, \ldots X_r$, each of size q; an integer $k \in \mathbf{N}$; Boolean formulas F_0, F_1, F_2 in conjunctive normal form, where F_0 and F_2 are expressions on q variables, and F_1 is an expression on $2q$ variables.

Parameter: k.

Question: Is it possible to satisfy the formula

$$F_0(X_1) \wedge \bigwedge_{1 \leq i \leq r-1} F_1(X_i, X_{i+1}) \wedge F_2(X_r)$$

by setting exactly k variables to true from each set X_i and all others to false?

Moreover, the argument in the proof of Theorem 8 can be applied, and thus Preregular Chained Weighted Positive CNF-Satisfiability (the variant of the problem above where all literals in F_0, F_1 and F_2 are positive) is XNLP-complete.

For a further simplification of our later proofs, we obtain completeness for a regular variant with only one set of constraints.

Regular Chained Weighted CNF-Satisfiability

Given: r sets of Boolean variables $X_1, X_2, \ldots X_r$, each of size q; integer $k \in N$; Boolean formula F, which is in conjunctive normal form and an expression on $2q$ variables.

Parameter: k.

Question: Is it possible to satisfy the formula

$$\bigwedge_{1 \leq i \leq r-1} F(X_i, X_{i+1})$$

by setting exactly k variables to true from each set X_i and all others to false?

Theorem 9. Regular Chained Weighted CNF-Satisfiability *is* XNLP-*complete.*

Proof. The idea of the proof is to add the constraints from add the constraints from F_0 and F_2 to F_1, resulting in a new F, but to ensure that these constraints are only "verified" at the start and at the end of the chain.

To achieve this, we add variables $t_{i,j}$ for $i \in [1, r]$ and $j \in [1, r]$, with $t_{i,j}$ part of X_i. We increase the parameter k by one. The construction is such that $t_{i,j}$ is true, if and only if $i = j$; $t_{i,1}$ implies all constraints from F_0, and $t_{i,r}$ implies all constraints from F_2. The details are as follows. We start by setting $F = F_1$, and then in a number of steps, add additional constraints to F.

1. We ensure that for all $i \in [1, r]$, exactly one $t_{i,j}$ is true. This can be done by adding a clause
$$\bigvee_{1 \leq j \leq r} t_{i,j}.$$
As the number of disjoint sets of variables that each have at least one true variable still equals k (as we increased both the number of these sets and k by one), we cannot have more than one true variable in the set.
2. For all $i \in [1, r]$ and $j \in [1, r]$, we enforce the constraint $t_{i,j} \Leftrightarrow t_{i+1,j+1}$ by adding the clauses $\neg t_{i,j} \vee t_{i+1,j+1}$ and $t_{i,j} \vee \neg t_{i+1,j+1}$.
3. We add a constraint that ensures that $t_{i,1}$ is false for all $i \in [2, r]$. This can be done by adding the clause with one literal $\neg t_{i+1,1}$ to the formula $F_1(X_i, X_{i+1})$, i.e., we have a condition on a variable that is an element of the set given as second parameter. Together with the previous set of constraints, this ensures that $t_{i,1}$ is true then $i = 1$.
4. Similarly, we add a constraint that ensures that $t_{i,r}$ is false for $i < r$. This is done by adding a clause with one literal $\neg t_{i,r}$ to $F(X_i, X_{i+1})$.
5. We add a constraint of the form $t_1 \rightarrow F_0(X)$ to F; for all i, the variable t_1 is substituted by $t_{i,1}$ and X by X_i.
6. We add a constraint of the form $t_r \rightarrow F_2(X)$ to F; for all i, the variable t_1 is substituted by $t_{i,1}$ and X by X_i.

The first four additional constraints given above ensure that for all $i \in [1, r]$ and $j \in [1, r]$, $t_{i,j}$ is true if and only if $i = j$. Thus, the fifth constraint enforces $F_0(X_1)$ (since $t_{1,1}$ has to be true); for $i > 1$, this constraint has no effect. Likewise, the sixth constraint enforces $F_2(X_r)$. Hence, the new set of constraints is equivalent to the constraints for the first version of Preregular Chained Weighted CNF-Satisfiability.

Using standard logic operations, the constraints can be transformed to conjunctive normal form; one easily can verify the time and space bounds. □

Again, with a proof identical to that of Theorem 8, we can show that the variant with only positive literals (Regular Chained Weighted Positive CNF-Satisfiability) is XNLP-complete.

From the proofs above, we note that each set of variables X_i can be partitioned into k subsets, and a solution has exactly one true variable for each subset, e.g., for each t', exactly one $x_{t',r,s}$ is true in the proof of Theorem 7. This still holds after the modification in the proofs of the later results. We define the following variant.

Partitioned Regular Chained Weighted CNF-Satisfiability
Input: r sets of Boolean variables $X_1, X_2, \dots X_r$, each of size q; an integer $k \in \mathbf{N}$; Boolean formula F, which is in conjunctive normal form and an expression on $2q$ variables; for each i, a partition of X_i into $X_{i,1}, \dots, X_{i,k}$ with for all i_1, i_2, j: $|X_{i_1,j}| = |X_{i_2,j}|$.
Parameter: k.
Question: Is it possible to satisfy the formula

$$\bigwedge_{1 \leq i \leq r-1} F(X_i, X_{i+1})$$

by setting from each set $X_{i,j}$ exactly 1 variable to true and all others to false?

We call the variant with only positive literals Partitioned Regular Chained Weighted Positive CNF-Satisfiability.

Corollary 10. Partitioned Regular Chained Weighted CNF-Satisfiability *and* Partitioned Regular Chained Weighted Positive CNF-Satisfiability *are* XNLP*-complete.*

3.2. Chained multicolored clique

The Multicolored Clique problem is an important tool to prove fixed parameter intractability of various parameterized problems. It was independently introduced by Pietrzak [47] (under the name Partitioned Clique) and by Fellows et al. [34].

In this paper, we introduce a chained variant of Multicolored Clique. In this variant, we ask to find a sequence of cliques that are overlapping with the previous and next clique in the chain.

Chained Multicolored Clique
Input: Graph $G = (V, E)$; partition of V into sets $V_1, \dots, V_r$, such that for each edge $\{v, w\} \in E$, if $v \in V_i$ and $w \in V_j$, then $|i - j| \leq 1$; function $f : V \to \{1, 2, \dots, k\}$.
Parameter: k.
Question: Is there a subset $W \subseteq V$ such that for each $i \in [1, r-1]$, $W \cap (V_i \cup V_{i+1})$ is a clique, and for each $i \in [1, r]$ and each $j \in [1, k]$, there is a vertex $w \in V_i \cap W$ with $f(w) = j$?

Thus, we have a clique with $2k$ vertices in $V_i \cup V_{i+1}$ for each $i \in [1, r-1]$, with for each color a vertex with that color in V_i and a vertex with that color in V_{i+1}. Importantly, the same vertices in V_i are chosen in the clique for $V_{i-1} \cup V_i$ as for $V_i \cup V_{i+1}$ for each $i \in [2, r-1]$. Below, we call such a set a *chained multicolored clique*.

Theorem 11. Chained Multicolored Clique *is* XNLP*-complete.*

Proof. Membership in XNLP is easy to see: iteratively guess for each V_i which vertices belong to the clique. We only need to keep the clique vertices in V_{i-1} and V_i in memory.

We now prove hardness via a transformation from Partitioned Regular Chained Weighted Positive CNF-Satisfiability. We are given:

- r sets of Boolean variables $X_1, X_2, \dots X_r$, each of size q;
- an integer $k \in \mathbf{N}$; for each i, a partition of X_i into $X_{i,1}, \dots, X_{i,k}$ with for all i_1, i_2, j: $|X_{i_1,j}| = |X_{i_2,j}|$;
- a Boolean formula F, which is in conjunctive normal form using only positive literals and an expression on $2q$ variables.

We need to decide if it is possible to satisfy the formula

$$\bigwedge_{1 \leq i \leq r-1} F(X_i, X_{i+1})$$

by setting from each set $X_{i,j}$ exactly 1 variable to true and all others to false.

We build an equivalent instance of Chained Multicolored Clique. See Fig. 2 for an example of the construction. We create a graph with the following vertices. For each $i \in [r-1]$ and for each clause c in F, we create a vertex set $V_{i,c}$. Informally, this serves to check whether the clause c is satisfied by $X_i \cup X_{i+1}$, so we need to "track" which of those variables are set to true. For each $j \in [k]$, we add the following vertices to $V_{i,c}$.

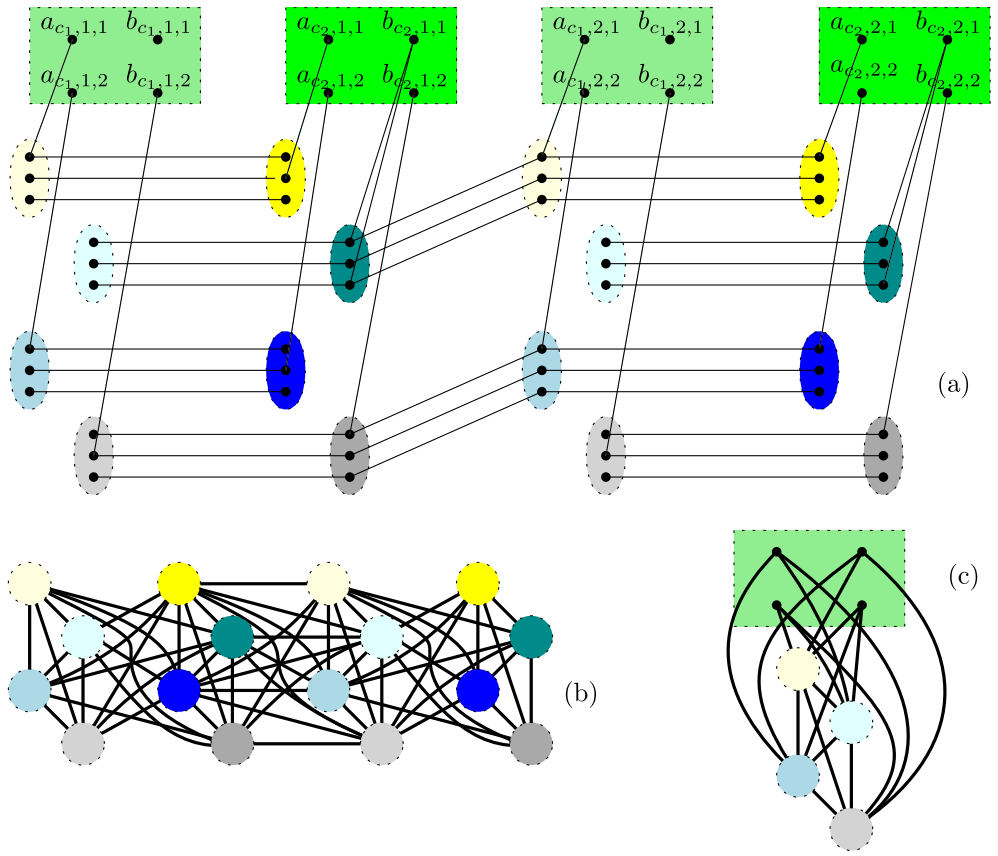

Fig. 2. The construction in the proof of Theorem 11 is illustrated for $r = 2$, $q = 6$ and $k = 2$. This means the variables sets $X_{i,j}$ are of size 3. The formula F has two clauses: the first with three (positive) literals, and the second with four positive literals. These literals correspond to the vertices in the sets marked with an ellipse that are endpoint of an edge with the other endpoint in a green squared set. For visibility, in (a) only some of the edges are shown. Each vertex of type a or b is incident to all selection vertices in the current and next part, except those corresponding the set represented by the a or b vertex. The selection vertices are incident to all selection vertices in the current and next part, with a few exceptions: there is no adjacency to vertices in the same set, and when edges are shown in part (a) to other vertices in a set. Additional adjacencies are indicated in (b) and (c); the fat edges represent complete bipartite subgraphs.

- We add two vertices $a_{c,i,j}$ and $b_{c,i,j}$. We give these the color $2k + 1$. Informally, choosing the vertex $a_{c,i,j}$ corresponds to satisfying clause c using a vertex from $X_{i,j}$; similarly, choosing the vertex $b_{c,i,j}$ corresponds to satisfying clause c using a vertex from $X_{i+1,j}$. We refer to these as *clause checking* vertices.
- For each $x \in X_{i,j}$, we add a vertex $v_{x,i,j,c}$ which we give color j. These vertices keep track of the variable x from $X_{i,j}$ which is assigned true. We refer to these as *current selection* vertices and say $v_{x,i,j,c}$ *selects* x.
- For each $y \in X_{i+1,j}$, we add a vertex $w_{y,i,j,c}$ which we give color $j + k$. These vertices keep track of the variable y from $X_{i+1,j}$ which is assigned true. We refer to these as *next selection* vertices and say $w_{y,i,j,c}$ *selects* y.

This defines the vertex set and vertex coloring (with $2k + 1$ colors, the new parameter).

We place an arbitrary order on the clauses $c_1, \ldots, c_f$ in F and the partition of the vertex set gets the order

$$(V_{1,c_1}, V_{1,c_2}, \ldots, V_{1,c_f}, V_{2,c_1}, V_{2,c_2} \ldots, \ldots, V_{r-1,c_f}).$$

Next, we define the edges, which will only be between vertices in the same or consecutive parts (according to the partition order above). The idea is that vertices that model different selection choices have no restrictions on each other; however, if we choose $v_{x,i,j,c}$ then we should also choose $v_{x,i,j,c'}$ for all future c', for example.

We first describe the adjacencies between the (current or next) selection vertices.

- *Case 1: vertices with the same* $i \in [r-1]$. If selection vertices $u \in V_{i,c}$ and $u' \in V_{i,c'}$ are placed in the same part ($c = c'$), or consecutive parts, then they are adjacent if and only if
  - they have different colors (from $[2k]$), or
  - they select the same vertex (i.e. $u' = v_{x,i,j,c}$ and $u = v_{x,i,j,c'}$ for some x, or $u = w_{y,i,j,c}$ and $u' = w_{y,i,j,c'}$ for some y).

- *Case 2: one vertex has $i \in [r-2]$ and the other $i+1$.* The vertices $u \in V_{i,c_r}$ and $u' \in V_{i+1,c_1}$ are adjacent if
  - u is a current selection vertex ("v-type"), or
  - u' is a next selection vertex ("w-type"), or
  - $u = w_{x,i,j,c_r}$ and $u' = v_{x',i,j',c_1}$ where $j \neq j'$ or $x = x'$.

Note that, as we wanted, if vertices are in a consecutive part, and both model selection from the same set $X_{i,j}$, then they are only adjacent if they select the same vertex.

Next, we define adjacencies to $a_{c,i,j}$ and $b_{c,i,j}$ in order to check whether c is satisfied. For $i \in [r-1]$, $j \in [k]$ and c a clause in F,

- the vertices $a_{c,i,j}$ and $b_{c,i,j}$ are adjacent to all vertices from the previous and next part;
- the vertex $a_{c,i,j}$ is adjacent to $v_{x,i,j,c} \in V_i$ if and only if $x \in X_{i,j}$ appears in c (as a positive literal), and $a_{c,i,j}$ is adjacent to all vertices of the form $w_{y,i,j,c}$;
- the vertex $b_{c,i,j}$ is adjacent to $w_{y,i,j,c} \in V_{i+1}$ if and only if $y \in X_{i+1,j}$ appears in c (as a positive literal), and $b_{c,i,j}$ is adjacent to all vertices of the form $v_{x,i,j,c}$.

Note that there are only edges between vertices in consecutive parts and that if p is the total number of clauses in F and $\ell = \max_{i,j} |X_{i,j}|$, our total number of created vertices is at most $rk(\ell+2)p$.

We claim the created instance admits a multicolored clique if and only if the original instance was satisfiable.

Suppose first that we are given a satisfying assignment, given by a choice of $t_{i,j} \in X_{i,j}$ for each $i \in [r]$ and $j \in [k]$ (i.e. the variables to be set to true). For $i \in [r]$, $j \in [k]$ and clause c from F, we select $v_{t_{i,j},i,j,c}$ if $i \leq r-1$ and we also select $w_{t_{i,j},i-1,j,c}$ if $i \geq 2$. These vertices are adjacent (for any choice of $t_{i,j}$) so this is a multicolored clique except for missing the color $2k+1$ in all parts. For each $i \in [r-1]$ and clause c, there is a choice of $j \in [k]$ such that either $t_{i,j}$ or $t_{i+1,j}$ appears in c, since c is satisfied. We choose $a_{c,i,j}$ or $b_{c,i,j}$ respectively. This gives the desired multicolored clique (with all $2k+1$ colors now).

Conversely, if we are given a multicolored clique, then for $i \in [r]$ and $j \in [k]$, let v_{x,i,j,c_1} be the chosen vertex of color j from V_{i,c_1}. We set $x \in X_{i,j}$ to true. This gives an assignment which sets the right number of variables to true and we need to check if it satisfies all the clauses. Let $i \in [r-1]$ and c a clause in F. There has to be a vertex u of color $2k+1$ in $V_{i,c}$, so u is a clause checking vertex.

- Case 1: u is of the form $a_{c,i,j}$. It is adjacent to the chosen vertex of color j from $V_{i,c}$, of the form $v_{x,i,j,c}$ (since we have a clique). By definition of the edges, this means $x \in X_{i,j}$ appears in c and so we just need to argue that x has been set to true. This follows from an inductional argument: it is immediate if $c = c_1$ and if $c = c_b$ for $b > 1$, then the only vertex in the previous part of color j that is adjacent to $v_{x,i,j,c}$ is $v_{x,i,j,c_{b-1}}$. So $v_{x,i,j,c_{b-1}}$ must be part of the clique, and continuing, v_{x,i,j,c_1} must be part of the clique and indeed we set x to true.
- Case 2: u is of the form $b_{c,i,j}$. This must be adjacent to a chosen vertex of the form $w_{x,i,j,c}$ of color $j+k$. Again, the clause will be satisfied if we set $x \in X_{i+1,j}$ to true, which we did if we chose the vertex $v_{x,i-1,j,c_1}$. This follows by a similar inductional argument.

All clauses are satisfied so indeed there is a satisfying assignment.

This shows the required equivalence and proves the claim. Finally, we note the reduction is easily performed in $f(k) + \log(rp\ell)$ space. □

A simple variation is the following. We are given a graph $G = (V, E)$, a partition of V into sets $V_1, \ldots, V_r$ with the property that for each edge $\{v, w\} \in E$, if $v \in V_i$ and $w \in V_j$ then $|i - j| \leq 1$, and a coloring function $f : V \to \{1, 2, \ldots, k\}$. A *chained multicolored independent set* is an independent set S with the property that for each $i \in [1, r]$ and each color $j \in [1, k]$, the set S contains exactly one vertex $v \in V_i$ of color $f(v) = j$. The Chained Multicolored Independent Set problem asks for the existence of such a chained multicolored independent set, with the number of colors k as parameter. We have the following simple corollary.

Corollary 12. Chained Multicolored Independent Set *is* XNLP*-complete.*

Proof. This follows directly from Theorem 11, by observing that the following "partial complement" of a partitioned graph $G = (V_1 \cup \cdots \cup V_r, E)$ can be constructed in logarithmic space: create an edge $\{v, w\}$ if and only if there is an i with $v \in V_i$ and $w \in V_i \cup V_{i+1}$, $v \neq w$ and $\{v, w\} \notin E$. □

3.3. CSP with binary constraints parameterized by pathwidth

In this paper, we consider the Constraint Satisfaction Problem where constraints are binary, often called the Binary CSP problem. The problem can be viewed as a generalization of graph coloring. The formal definition is given below.

An instance of Binary CSP is a triple

$$I = (G, \{D(u) \colon u \in V(G)\}, \{C(u,v) \colon \{u,v\} \in E(G)\}),$$

where

- G is an undirected graph, called the *Gaifman graph* of the instance;
- for each $u \in V(G)$, $D(u)$ is a finite set called the *domain* of u; and
- for each $\{u,v\} \in E(G)$, $C(u,v) \subseteq D(u) \times D(v)$ is a binary relation called the *constraint* at $\{u,v\}$. Throughout this paper, we apply the convention that $C(v,u) = \{(b,a) \mid (a,b) \in C(u,v)\}$.

A *satisfying assignment* for an instance I is a function η that maps every variable u to a value $\eta(u) \in D(u)$ such that for every edge $\{u,v\}$ of G, we have $(\eta(u), \eta(v)) \in C(u,v)$. The Binary CSP problem asks, for a given instance I, whether I is *satisfiable*, that is, there is a satisfying assignment for I.

For Binary CSP parameterized by pathwidth, we assume a path decomposition of width k of the Gaifman graph G is also given as part of the input and consider k as our parameter.

Theorem 13. Binary CSP *is XNLP-complete when parameterized by the following parameters:*

1. *pathwidth;*
2. *pathwidth + maximum degree;*
3. *cutwidth;*
4. *bandwidth.*

Proof. We first prove membership when parameterized by pathwidth, using a "standard" dynamic programming approach. If the given path decomposition is $(X_1, X_2, \ldots, X_\ell)$, then for $i = 1, \ldots, n$,

- we (non-deterministically) guess an element of $f_i(v) \in D(v)$ for each $v \in X_i$ (which we store in memory),
- we check constraints: check if $(f_i(u), f_i(v)) \in C(u,v)$ for all $u, v \in X_i$ with $\{u,v\} \in E(G)$ (else fail),
- we check consistency: if $v \in X_{i-1}$, we check whether $f_i(v) = f_{i-1}(v)$ (else fail),
- we free up memory (forget $f_{i-1}(v)$ for all $v \in X_{i-1}$).

The total memory used is $O(k \log n)$ bits, with k the pathwidth and n the number of vertices. The described non-deterministic algorithm also runs in fpt time. Membership for the other parameters follows analogously.

Next, we prove XNLP-hardness via a transformation from Chained Multicolored Clique.

Let $G = (V, E)$ be a k-colored graph with partition $V_1, \ldots, V_r$ of the vertex set, such that edges are only between the same or consecutive parts.

For each $i \in [r]$ and $j \in [k]$, we create a vertex $u_{i,j}$. Let $U_i = \{u_{i,j} : j \in [k]\}$. A vertex $u \in U_i$ is adjacent to $v \in U_{i'}$ if and only if $|i - i'| \leq 1$. This defines the Gaifman graph. So

$$(X_1, \ldots, X_{r-1}) := (U_1 \cup U_2, U_2 \cup U_3, \ldots, U_{r-1} \cup U_r)$$

gives a path decomposition of the Gaifman graph of width at most $2k$. Moreover, the maximum degree of the graph is at most $3k$ and the total number of vertices is rk.

For $i \in [r]$ and $j \in [k]$, the domain $D(u_{i,j})$ is given by the set of vertices of color j in V_i. Adjacent vertices $u_{i,j}$ and $u_{i',j'}$ have the following constraint set:

$$C(u_{i,j}, u_{i',j'}) = \{(v, v') \in D(u_{i,j}) \times D(u_{i',j'}) \mid \{v, v'\} \in E(G)\}.$$

In words, they may only be assigned to adjacent vertices (from V_i of color j and $V_{i'}$ of color j' respectively, by definition of the domains).

Once the construction is understood, the equivalence is direct: any satisfying assignment is a multicolored clique and vice versa.

The Gaifman graph in our reduction has bandwidth at most $2k-1$: take the linear ordering f with $f(u_{i,j}) = (i-1) \cdot k + j$. (I.e., we first number the vertices in U_1, then in U_2, etc. As each vertex is only adjacent to the vertices in its own set U_i, and the previous and next sets U_{i-1}, and U_{i+1}, the bandwidth of this ordering is $2k-1$.) This linear ordering f also has cutwidth $O(k^2)$, and the hardness for all claimed parameters now follows. □

We remark that we will later show that List Coloring is also XNLP-complete parameterized by pathwidth. The simple reduction however blows up the maximum degree, which is to be expected because the problem becomes fpt when both the pathwidth and the maximum degree are bounded. This means that we cannot expect to extend the result above to List Coloring for the other three parameters.

3.4. Non-decreasing counter machines

In this subsection, we introduce a new simple machine model, which can also capture the computational power of XNLP (see Theorem 14). This model will be a useful stepping stone when proving XNLP-hardness reductions in Section 4.

A *Nondeterministic Nondecreasing Checking Counter Machine* (or: NNCCM) is described by a 3-tuple (k, n, s), with k and n positive integers, and $s = (s_1, \ldots, s_r)$ a sequence of 4-tuples (called *checks*). For each $i \in \{1, \ldots, r\}$, the 4-tuple s_i is of the form (c_1, c_2, r_1, r_2) with $c_1, c_2 \in \{1, 2, \ldots, k\}$ positive integers and $r_1, r_2 \in \{0, 1, 2, \ldots, n\}$ non-negative integers. These model the indices of the counters and their values respectively.

An NNCCM (k, n, s) with $s = (s_1, \ldots, s_r)$ works as follows. The machine has k counters that are initially 0. For i from 1 to r, the machine first sets each of the counters to any integer that is at least its current value and at most n. After this, the machine performs the ith check $s_i = (c_1, c_2, r_1, r_2)$: if the value of the c_1th counter equals r_1 and the value of the c_2th counter equals r_2, then we say the ith check rejects and the machine halts and rejects. When the machine has not rejected after all r checks, the machine accepts.

For example, we may have $n = 2$ and $k = 3$, with checks

$$s = ((1, 2, 0, 1), (2, 3, 1, 1), (1, 3, 0, 2)).$$

We pass s_1 since $(c_1, c_2) = (0, 0) \neq (0, 1)$. We then increase c_1 to 1 and increase c_3 to 2. We succeed s_2 and s_3 since $(c_2, c_3) = (0, 2) \neq (1, 1)$ and $(c_1, c_3) = (1, 2) \neq (0, 2)$. The machine accepts since none of the checks rejected. (Many variations on this also work.) An example which always rejects is $n = 1, k = 2$ and

$$s = ((1, 2, 0, 0), (1, 2, 1, 0), (1, 2, 0, 1), (1, 2, 1, 1)).$$

The nondeterministic steps can be also described as follows. Denote the value of the cth counter when the ith check is done by $c(i)$. We define $c(0) = 0$. For each $i \in \{1, \ldots, r\}$, $c(i)$ is an integer that is nondeterministically chosen from $[c(i-1), n]$.

We consider the following computational problem.

Accepting NNCCM
Given: An NNCCM (k, n, s) with all integers given in unary.
Parameter: The number of counters k.
Question: Does the machine accept?

Theorem 14. Accepting NNCCM *is* XNLP*-complete.*

Proof. We first argue that Accepting NNCCM belongs to XNLP. We simulate the execution of the machine. At any point, we store k integers from $[0, n]$ that give the current values of our counters, as well as the index $i \in [1, r]$ of the check that we are performing. This takes only $O(k \log n + \log r)$ bits. When we perform the check $s_i = (c_1, c_2, r_1, r_2)$, we store the values c_1, c_2, r_1, r_2 in order to perform the check using a further $O(\log n)$ bits. So we can simulate the machine using $O(k \log n + \log r)$ space. The running time is upper bounded by some function of the form $f(k) \cdot \text{poly}(n, r)$.

We now prove hardness via a transformation from Chained Multicolored Clique. We are given a k-colored graph with vertex sets $V_1, \ldots, V_s$. By adding isolated vertices if needed, we may assume that, for each $i \in [1, s]$, the set V_i contains exactly m vertices of each color. We will assume that ℓ is even; the proof is very similar for odd ℓ. We set $n = ms$.

We create $4k$ counters: for each color $i \in [1, k]$, there are counters $c_{i,1,+}$, $c_{i,1,-}$, $c_{i,0,+}$, $c_{i,0,-}$. We use the counters $c_{i,1,\pm}$ for selecting vertices from sets V_j with j odd and the counters $c_{i,0,\pm}$ for selecting vertices from sets V_j with j even.

The intuition is the following. We increase the counters in stages, where in stage j we model the selection of the vertices from V_j. Say j is even. We increase the counters $c_{i,0,+}$, $c_{i,0,-}$ to values within $[jm + 1, (j+1)m]$. Since counters may only move up, there can be at most one $\ell \in [1, m]$ for which the counters at some point take the values $c_{i,0,+} = jm + \ell$ and $c_{i,0,-} = (j+1)m + 1 - \ell$. We enforce that such an ℓ exists and interpret this as placing the ℓth vertex of color i in V_j into the chained multicolored clique.

We use the short-cut (c_1, c_2, R_1, R_2) for the sequence of checks $((c_1, c_2, r_1, r_2) : r_1 \in R_1,\ r_2 \in R_2)$ performed in lexicographical order, e.g. for $R_1 = \{1, 2\}$ and $R_2 = \{1, 2, 3\}$, the order is $(1, 1), (1, 2), (1, 3), (2, 1), (2, 2), (2, 3)$.

For each $j \in [1, s]$, the jth *vertex selection check* confirms that for each $i \in [1, k]$, $(c_{i,par,+}, c_{i,par,-})$ is of the form $(jm + \ell, (j+1)m + 1 - \ell)$ for some $\ell \in [1, m]$, where *par* denotes the parity of j. For each $i \in [1, k]$, we set $c_1 = c_{i,par,+}$ and $c_2 = c_{i,par,-}$, and perform the following checks in order.

1. $(c_1, c_2, [0, jm - 1], [0, n])$;
2. $(c_1, c_2, [0, n], [0, jm - 1])$;
3. for $\ell \in [1, m]$, $(c_1, c_2, \{jm + \ell\}, [jm, (j+1)m] \setminus \{(j+1)m + 1 - \ell\})$;
4. $(c_1, c_2, [(j+1)m + 1, n], [0, n])$;
5. $(c_1, c_2, [0, n], [(j+1)m + 1, n])$.

Suppose all the checks succeed. After the second check, c_1 and c_2 are both at least jm, and before the last two checks, c_1 and c_2 are both at most $(j+1)m$. The middle set of checks ensure that there is some ℓ for which the c_1th counter and c_2th counter have been simultaneously at the values $jm+\ell$ and $(j+1)m+1-\ell$ respectively. We say the check *chooses* ℓ for c_1 and c_2. Since the counters can only move up, this ℓ is unique.

This is used as a subroutine below, where we create a collection of checks such that the corresponding NNCCM accepts if and only if the k-colored graph has a chained multicolored clique. For $j=1$ to s, we do the following:

- Let $par \equiv j \bmod 2$ denote the parity of j and let $par' = 1 - par \in \{0,1\}$ denote the opposite parity.
- We perform a jth vertex selection check.
- We verify that all selected vertices in V_j are adjacent. Let uu' be a non-edge with $u, u' \in V_j$. Let ℓ, i, ℓ', i' with $i \neq i'$ be such that u is the ℓth vertex of color i in V_j and u' is the ℓ'th vertex of color i'. We add the check $(c_{i,par,+}, c_{i',par,+}, jm+\ell, jm+\ell')$. This ensures that we do not put both u and u' in the clique of V_j.
- If $j > 1$, then we verify that all selected vertices in V_j are adjacent to all selected vertices in V_{j-1}. Let uu' be a non-edge with $u \in V_j$ and $u' \in V_{j-1}$ and let ℓ, i, ℓ', i' be as above. We add the check $(c_{i,par,+}, c_{i',par',+}, jm+\ell, jm+\ell')$ to ensure that we do not put both u and u' into the clique.
- We finish with another jth vertex selection check. If $j > 1$, we also do a $(j-1)$th vertex selection check. This ensures our counters are still "selecting vertices" from V_{j-1} and V_j.

We now argue that the set of checks created above accepts if and only if the graph contains a chained multicolored clique. Suppose first that such a set $W \subseteq V$ exists for which $W \cap (V_j \cup V_{j+1})$ forms a clique for all j and W contains at least one vertex from V_j of each color. We may assume that $W_j = W \cap V_j$ is of size k for each j. Let $j \in [1,s]$ be given of parity par and for each $i \in [k]$, let the $f(i)$th vertex of V_j of color i be in W_j. Before the first jth vertex selection check, we move the counters $(c_{i,par,+}, c_{i,par,-})$ to $(jm+f(i), (j+1)m-1-f(i))$, and these will be left there until the first $(j+2)$th vertex selection check. This will ensure that all the checks accept.

Suppose now that all the checks accept. We first make an important observation. Let two counters c_1 and c_2 be given. If in an iteration above, the first vertex selection check chooses ℓ for c_1 and c_2 and the second vertex selection check chooses ℓ', then it must be the case that $\ell = \ell'$. Indeed, we cannot increase c_1 or c_2 beyond $(j+1)m$ before the last vertex selection check, and they need to be above jm due to the first. If the first vertex selection check selects ℓ, then the c_1th counter is at least $jm+\ell$, so in order for it to be $jm+\ell'$ in the second check, we must have $\ell' \geq \ell$. Considering the value of the c_2th counter, we also find $\ell \geq \ell'$ and hence $\ell = \ell'$. In particular, the counters cannot have moved between the two vertex selection checks.

It is hence well-defined to, for $j \in [1,s]$ of parity par, let W_j be the set of vertices that are for some color $i \in [1,k]$ the ℓth vertex of color i in V_j, where ℓ denotes the unique value that is selected by a jth vertex selection check for $c_{i,par,+}$ and $c_{i,par,-}$. We claim that $W = \bigcup_{j=1}^{s} W_j$ is our desired multicolored chained clique. It contains exactly one vertex per color from V_j. Suppose $u \in W_j$ and $u' \in W_{j-1} \cup W_j$ are not adjacent and distinct. Let i, par, ℓ be such that u is the ℓth color of parity par in V_j for j of parity par, and similar for i', par', ℓ'. Then at the jth iteration, the check $(c_{i,par,+}, c_{i',par,+}, jm+\ell, jm+\ell')$ has been performed (because uu' is a non-edge). Since $u, u' \in W$ and this check is done between vertex selection checks, the counters have to be at those values. This shows that there is a check that rejects, a contradiction. So W must be a chained multicolored clique. □

The Accepting NNCCM problem appears to be a very useful tool for giving XNLP-hardness proofs. Note that one step where the k counters can be increased to values at most n can be replaced by kn steps where counters can be increased by one, or possibly a larger number, again to at most n. This modification is used in some of the proofs in the following section.

4. Applications

In this section, we consider several problem, which we prove to be XNLP-complete. Using the building blocks of the previous section, we provide XNLP-completeness results for the following problems:

- Subsection 4.1: List Coloring and Precoloring Extension parameterized by pathwidth.
- Subsection 4.2: Dominating Set and Independent Set parameterized by logarithmic pathwidth.
- Subsection 4.3: Scheduling with Precedence Constraints parameterized by the number of machines and the partial order width.
- Subsection 4.4: Uniform Emulations of Weighted Paths.
- Subsection 4.5: Bandwidth.
- Subsection 4.6: Acyclic Finite State Automata Intersection.

4.1. List coloring parameterized by pathwidth

In this section, we show that List Coloring and Precoloring Extension are XNLP-complete when parameterized by pathwidth.

In the List Coloring problem, we are given a graph $G = (V, E)$, a finite set of colors $\mathcal{C}$, and for each vertex $v \in V$, a subset of the colors $C(v) \subseteq \mathcal{C}$, and ask if there is a function $f : V \to \mathcal{C}$, such that each vertex gets assigned a color from its own set: $\forall v \in V : f(v) \in C(v)$, and the endpoints of each edge have different colors: $\forall \{v, w\} \in E : f(v) \neq f(w)$. We call such a function f a *list coloring* for G.

In the Precoloring Extension problem, we are given a graph $G = (V, E)$, a set of colors $\mathcal{C}$, a set of precolored vertices $W \subseteq V$, a function $p : W \to \mathcal{C}$, and ask for a coloring $f : V \to \mathcal{C}$ that extends p (for all $w \in W : f(w) = p(w)$), such that the endpoints of each edge have different colors: $\forall \{v, w\} \in E : f(v) \neq f(w)$.

The List Coloring and Precoloring Extension problems parameterized by treewidth or pathwidth belong to XP [41]. Fellows et al. [33] have shown that List Coloring and Precoloring Extension parameterized by treewidth are W[1]-hard. (The transformation in their paper also works for parameterization by pathwidth.) We strengthen this result by showing that List Coloring and Precoloring Extension parameterized by pathwidth are XNLP-complete. Note that this result also implies W[t]-hardness of the problems for all integers t. Note that the standard Graph Coloring problem is fixed parameter tractable with treewidth as parameter [2].

Theorem 15. List Coloring *parameterized by pathwidth is* XNLP*-complete.*

Proof. Membership follows analogously to Binary CSP (or by using that List Coloring is a special case of this).

In order to prove hardness, we reduce from Binary CSP. Let an instance $(G, (D(u))_{u \in V(G)}, (C(u, v))_{\{u,v\} \in E(G)})$ of Binary CSP be given. We create a new graph G' which will be an "enlarged" version of G. To create G', we start with a copy of G. The vertices in this copy also have the same domains: if $v' \in V(G')$ is a copy of $v \in V(G)$, then $L(v') = D(v)$.

For an edge $\{u, v\} \in E(G)$, let $u', v' \in V(G')$ denote the copies of $u, v \in V(G)$. For each pair $(c, d) \in (D(u) \times D(v)) \setminus C(u, v)$, we add a *helper* vertex $h_{uv,c,d}$ to $V(G')$ with list $\{c, d\}$ and make it adjacent to u' and v'.

A list coloring α of the copy of G within G' is exactly an assignment of the original instance (which satisfies the domains). It is easily checked that such a list coloring extends to the helper vertices of G' if and only if $(\alpha(u'), \alpha(v')) \in C(u, v)$ for all edges $\{u, v\}$ in G. This means there exists a list coloring if and only if there is a satisfying assignment for the original instance.

Finally, we note that the pathwidth of G' is at most the pathwidth of G plus 1. Indeed, the only new vertices are the "helper vertices", which are adjacent to two adjacent vertices. □

We deduce the following result from a well-known transformation from List Coloring.

Corollary 16. Precoloring Extension *parameterized by pathwidth is* XNLP*-complete.*

Proof. Membership follows immediately since Precoloring Extension can be seen as a special case of List Coloring in which the precolored vertices have lists of size one.

We prove hardness via a reduction from List Coloring. Consider an instance of List Coloring with graph $G = (V, E)$ and color lists $C(v)$ for all $v \in V$. Let $\mathcal{C} = \bigcup_{v \in V} C(v)$ be the set of all colors. For each vertex v with color list $C(v)$, for each color $\gamma \in \mathcal{C} \setminus C(v)$, add a new vertex that is only adjacent to v and is precolored with γ. The original vertices are not precolored. It is easy to see that we can extend the precoloring of the resulting graph to a proper coloring, if and only if the instance of List Coloring has a solution.

The procedure increases the pathwidth by at most one. Indeed, if we are given a path decomposition of G, then we can build the path decomposition of the resulting graph as follows. We iteratively visit all the bags X_i of the path decomposition. Let $v \in X_i$ be a vertex for which X_i is the first bag it appears in (so $i = 1$ or $v \notin X_{i-1}$). We create a new bag $X_i \cup \{w\}$ for each new neighbor w of v and place this bag somewhere in between X_i and X_{i+1}. This transformation can be easily executed with $k + O(\log n)$ bits of memory. □

From Theorem 15, and Corollary 16 we can also directly conclude that List Coloring and Precoloring Extension are XNLP-hard when parameterized by the treewidth. However, we expect that these problems are not members of XNLP; they are shown to be complete for the class XALP in [17], and we conjecture that XNLP is a proper subset of XALP.

4.2. Logarithmic pathwidth

There are several well known problems that can be solved in time $O(c^k n)$ for a constant c on graphs of pathwidth or treewidth at most k. Classic examples are Independent Set and Dominating Set (see e.g., [24, Chapter 7.3]), but there are many others, e.g., [8,52].

In this subsection, rather than bounding the pathwidth by a constant, we allow the pathwidth to be linear in the logarithm of the number of vertices of the graph. We consider the following problem.

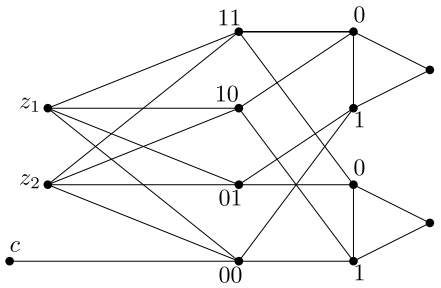

Fig. 3. An example of a clause gadget is depicted. Here, $t = 2$ and only the variable with index 00 satisfies clause c

Log-Pathwidth Dominating Set **Input:** Graph $G = (V, E)$, path decomposition of G of width ℓ, integer K.
Parameter: $\lceil \ell / \log |V| \rceil$.
Question: Does G have a dominating set of size at most K?

The independent set variant and clique variants are defined analogously.

Theorem 17. Log-Pathwidth Dominating Set *is* XNLP-*complete.*

Proof. It is easy to see that the problem is in XNLP — run the standard dynamic programming algorithm for Dominating Set on graphs with bounded pathwidth, but instead of computing full tables, guess the table entry for each bag.

Hardness for XNLP is shown with help of a reduction from Partitioned Regular Chained Weighted Positive CNF-Satisfiability.

Suppose we are given F, X_1, ..., X_r, q, k, $X_{i,j}$ $(1 \leq i \leq r,\ 1 \leq j \leq k)$ as instance of the Partitioned Regular Chained Weighted Positive CNF-Satisfiability problem.

We assume that each set $X_{i,j}$ is of size 2^t for some integer t; otherwise, add dummy variables to the sets and a clause for each $X_{i,j}$ that expresses that a non-dummy variable from the set is true. Let c be the number of clauses in F. We now describe in a number of steps the construction of G.

Variable choice gadget For each set $X_{i,j}$ we have a variable choice gadget. The gadget consists of t copies of a K_3. Each of these K_3's has one vertex marked with 0, one vertex marked with 1, and one vertex of degree two, i.e., this latter vertex has no other neighbors in the graph.

The intuition is the following. We represent each element in $X_{i,j}$ by a unique t-bit bitstring (recall that we ensured that $|X_{i,j}| = 2^t$). Each K_3 represents one bit, and together these bits describe one element of $X_{i,j}$, namely the variable that we set to be true.

Clause checking gadget Consider a clause ϕ on $X_i \cup X_{i+1}$. We perform following construction separately for each such clause, and in particular the additional vertices defined below are not shared among clauses. For each $j \in [1, k]$, we add $2^t + 2$ additional vertices for both $X_{i,j}$ and $X_{i+1,j}$, as follows.

- We create a *variable representing* vertex v_x for each variable $x \in X_{i,j} \cup X_{i+1,j}$. We connect v_x to the each vertex that represents a bit in the complement of the bit string representation of x.
- We add two new vertices for $X_{i,j}$ and also for $X_{i+1,j}$ (see z_1 and z_2 in Fig. 3). These vertices are incident to all variable representing vertices for variables from the corresponding set.

We call this construction a *variable set* gadget.

The clause checking gadget for ϕ consists of $2k$ variable set gadgets (for each $j \in [1, k]$, one for each set of the form $X_{i,j}$ and $X_{i+1,j}$), and one additional *clause vertex*. The clause vertex is incident to all vertices in the current gadget that represent a variable that satisfies that clause. In total, we add $2 \cdot k \cdot (2^t + 2) + 1$ variables per clause.

The intuition is the following. From each triangle, we place either the vertex marked with 0 or with 1 in the dominating set. This encodes a variable for each $X_{i,j}$ as follows. The vertices in the triangles dominate all but one of the variable representing vertices; the one that is not dominated is precisely the encoded variable. This vertex is also placed in the dominating set (we need to pick at least one variable representing vertex in order to dominate the vertices z_1 and z_2 private to $X_{i,j}$). The clause is satisfied exactly when the clause vertex has a neighbor in the dominating set.

The graph G is formed by all variable choice and clause checking gadgets, for all variables and clauses. Recall that c is the number of clauses in F.

Claim 17.1. *There is a dominating set in G of size $rkt + 2kc(r-1)$, if and only if the instance of* Partitioned Regular Chained Weighted Positive CNF-Satisfiability *has a solution.*

Proof. Suppose we have a solution for the latter. Select in the variable choice gadgets the bits that encode the true variables and put these in the dominating set S. As we have rk sets of the form $X_{i,j}$, and each is represented by t triangles of which we place one vertex per triangle in the dominating set, we already used rkt vertices.

For each clause, we have $2k$ sets of the form $X_{i,j}$. For each, take the vertex that represents the true variable of the set and place it in S. Note that all vertices that represent other variables are dominated by vertices from the variable choice gadget. For each set $X_{i,j}$, the chosen variable representing vertex dominates the vertices marked z_1 and z_2. As the clause is satisfied, the clause vertex is dominated by the vertex that represents the variable representing vertex for the variable that satisfies it. This shows S is a dominating set. We have $(r-1)c$ clauses in total, and for each we have $2k$ sets of the form $X_{i,j}$. We placed one variable representing vertex for each of these $(r-1)c \cdot 2k$ sets in the dominating set. In total, $|S| = rkt + 2kc(r-1)$ as desired.

For the other implication, suppose there is a dominating set S of size $rkt + 2kc(r-1)$.

We must contain one vertex from each triangle in a variable choice gadget (as the vertex with degree two must be dominated). Thus, these gadgets contain at least rkt vertices from the dominating set.

Each of the vertices marked z_1 and z_2 must be dominated. So we must choose at least one variable choice vertex per variable set gadget. This contributes at least $2kc(r-1)$ vertices.

It follows that we must place exactly those vertices and no more: the set S contains exactly one vertex from each triangle in a variable choice gadget and exactly one variable selection vertex per variable set gadget. Note that we cannot replace a variable selection vertex by a vertex marked z_1 or z_2: either it is already dominated, or we also must choose its twin, but we do not have enough vertices for this.

From each triangle in a variable choice gadget, we must choose either the vertex marked 0 or 1, since otherwise one of the variable selection vertices will not be dominated. This dominates all but one of the variable selection vertices, which must be the variable selection vertex that is in the dominating set. We claim that we satisfy the formula by setting exactly these variables to true and all others to false. (Note also that we set exactly one variable per set $X_{i,j}$ to true this way.) Indeed, the clause vertex needs to be dominated by one of the variable selection vertices, and by definition this implies that the corresponding chosen variable satisfies the clause. □

The path decomposition can be constructed as follows. Let V_i for $i \in [1, r-1]$ be the set of all vertices in the variable choice gadgets of all sets $X_{i,j}$ and $X_{i+1,j}$ with $j \in [1,k]$. For any given i, we create a path decomposition that covers all vertices of V_i and all clause gadgets connected to V_i. We do this by sorting the clauses in any order, and then one by one traversing the clause checking gadgets. Each time, we begin by adding the z-vertices and the clause vertex of the gadget to the current bag. Then, iteratively we add and remove all the variable representing vertices one by one. After this, we remove the z-vertices and clause vertex from the bag and continue to the next clause. At any point, there are at most $6k \cdot t + 4k + 2$ vertices in a bag: $6k \cdot t$ from the set V_i, $4k$ z-variables, one clause vertex and one variable representing vertex. We then continue the path decomposition by removing all vertices related to variable choice gadgets of the set X_i and adding all vertices related to variable choice gadgets of the set X_{i+2}. Note that t is at most logarithmic in the size $|V|$ of the graph and that our new parameter of interest will be the pathwidth divided by $\log |V|$.

As the construction can be executed in logarithmic space, the result follows. □

The independent set variant is defined as follows; the clique variant is defined analogously.

Log-Pathwidth Independent Set
Input: Graph $G = (V, E)$, path decomposition of G of width ℓ, integer K.
Parameter: $\lceil \ell / \log |V| \rceil$.
Question: Does G have an independent set of size at least K?

Theorem 18. Log-Pathwidth Independent Set *is XNLP-complete.*

Proof. As with Log-Pathwidth Dominating Set, the problem is in XNLP as we can use the standard dynamic programming algorithm on graphs with bounded pathwidth, but guess the table entry for each bag.

We show XNLP-hardness with a reduction from Partitioned Regular Chained Weighted Positive CNF-Satisfiability. The reduction is similar to the reduction to Log-Pathwidth Dominating Set, but uses different gadgets specific to Independent Set. The clause gadget, which will be defined later, is a direct copy of a gadget from [44].

Suppose that $F, q, X_1, \ldots, X_r, k, X_{i,j}$ $(1 \le i \le r, 1 \le j \le k)$ is the given instance of the Partitioned Regular Chained Weighted Positive CNF-Satisfiability problem. Let $F = \{C_1, \ldots, C_m\}$ and let $|C_i|$ be the number of variables in clause C_i. We assume $|C_i|$ to be even for all $i \in [1,m]$; if this number is odd, then we can add an additional copy of one of the variables to the clause. Enlarging $|X_{i,j}|$ by at most a factor of 2 (by adding dummy variables to X_i), we may assume that $t = \log |X_{i,j}|$ is an integer. (Recall that log has base 2 in this paper.)

Variable choice gadget This gadget consists of t copies of a K_2 (a single edge), with one vertex marked with a 0 and the other with a 1. Each element in $X_{i,j}$ can be represented by a unique t-bit bitstring. One K_2 describes one such bit and

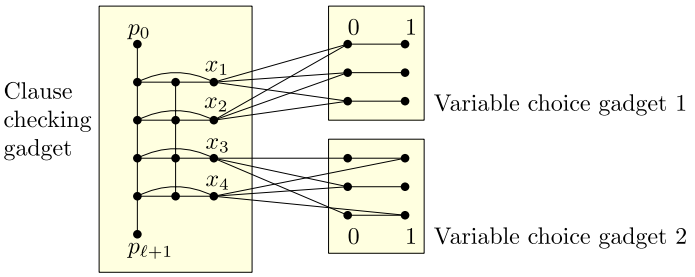

Fig. 4. An example of a clause gadget is depicted for the clause $x_1 \vee x_2 \vee x_3 \vee x_4$, together with two variable gadgets. Variables x_1 and x_2 are elements of the same set $X_{i,j}$, while x_3 and x_4 are elements of another set $X_{i',j'}$. Each set $X_{i,j}$ has 2^3 elements. The connections give which variable is represented. For example, x_1 is adjacent to 000 and the complement 111 of this is the code of x_1 (in this case, representing the last element of $X_{i,j}$). Similarly, x_4 is adjacent to 101, so has code 010, and thus represents the third variable from set $X_{i',j'}$ ($010 = 2; 3 = 2 + 1$).

together these t bits describe one element of $X_{i,j}$, which is going to be the variable that we assume to be true. We will ensure that in a maximal independent set, we need to choose exactly one vertex per copy of K_2, and hence always represent a variable.

Clause checking gadget The clause checking gadget is a direct copy of a gadget from [44] and is depicted in Fig. 4. The construction is as follows. Let $C = x_1 \vee \cdots \vee x_\ell$ be a clause on $X_i \cup X_{i+1}$ (for some $i \in [1, r]$). We create two paths $p_0, \ldots, p_{\ell+1}$ and $p'_1, \ldots, p'_\ell$ and add an edge $\{p_j, p'_j\}$ for $j \in [1, \ell]$. We then add the *variable representing* vertices $v_1, \ldots, v_\ell$. For $j \in [1, \ell]$, the vertex v_j is connected to p_j and p'_j and to the vertices representing the complement of the respective bits in the representation of x_j. This ensures that v_j can be chosen if and only if the vertices chosen from the variable choice gadget represent x_j.

For each clause, we add such a *clause checking* gadget on $3\ell + 2$ new vertices. The pathwidth of this gadget is 4.

Recall that we assume ℓ to be even for all clauses. Each clause checking gadget has the property that it admits an independent set of size $\ell+2$ (which is then maximum) if and only if at least one of the v-type vertices is in the independent set, as proved in [44]. Otherwise its maximum independent set is of size $\ell + 1$. We try to model the clause being satisfied by whether or not the independent set contains $\ell + 2$ vertices from this gadget.

The graph G is formed by combining the variable choice and clause checking gadgets.

Claim 18.1. *There is an independent set in G of size $rkt + (r-1)\sum_{i=1}^{m}(|C_i| + 2)$ if and only if the instance of* Partitioned Regular Chained Weighted Positive CNF-Satisfiability *has a solution.*

Proof. Suppose that there is a valid assignment of the variables that satisfies the clauses. We select the bits that encode the true variables in the variable choice gadgets and put these in the independent set. This forms an independent set because only one element per variable choice gadget is set to true by a valid assignment. As we have rk sets of the form $X_{i,j}$, and each is represented by t times two vertices of which we place one in the independent set, we now have rkt vertices in the independent set.

For each clause checking gadget for a clause on ℓ variables, we add a further $\ell + 2$ vertices to the independent set as follows. Let x_i be a variable satisfying the clause. We add the variable representing vertex v_i (corresponding to x_i) into the independent set. Since v_i is chosen in the independent set, an additional $\ell+1$ vertices can be added from the corresponding clause checking gadget, namely $p_0, p'_1, p_2, p'_2, \ldots$ on the one hand and $p_{\ell+1}, p'_\ell, p_{\ell-1}, p'_{\ell-2}, \ldots$ on the other hand. In total we place $rkt + (r-1)\sum_{i=1}^{m}(|C_i| + 2)$ vertices in the independent set.

To prove the other direction, let S be an independent set of size $rkt + (r-1)\sum_{i=1}^{m}(|C_i| + 2)$. Any independent set can contain at most t vertices from any variable choice gadget (one per K_2), and at most $|C| + 2$ vertices per clause checking gadget corresponding to clause C. Hence, S contains exactly t vertices per variable choice gadget and exactly $|C| + 2$ vertices per clause gadget.

Consider a variable choice gadget. Since S contains exactly t vertices (one per K_2), a variable from the set $X_{i,j}$ is encoded. It remains to show that we satisfy the formula by setting exactly these variables to true and all others to false. Consider a clause C and its related clause checking gadget. As $|C| + 2$ vertices from its gadget are in S, this means that at least one of its vertices of the form v_i is also in S. None of the neighbors of v_i can be in S, so the corresponding variable x_i must be set to true. Thus the clause is indeed satisfied. □

A path decomposition can be constructed as follows. For $i = 1, \ldots, r$, let V_i be the set of all vertices in the variable choice gadgets of the sets $X_{i,j}$ and $X_{i+1,j}$ (for all $j \in [1, k]$). We create a sequence of bags that contains V_i as well as a constant number of vertices from clause checking gadgets. We transverse the clause checking gadgets in any order. For a given clause checking gadget of a clause on ℓ variables, we first create a bag containing V_i and p_0, p_1, p'_1 and v_1. Then for $s = 2, \ldots, \ell$, we add p_s, p'_s and v_s to the bag and remove p_{s-2}, p'_{s-2} and v_{s-2} (if these exist). At the final step, we remove

$p_{\ell-1}$, $p'_{\ell-1}$ and $v_{\ell-1}$ from the bag and add $p_{\ell+1}$. We then continue to the next clause checking gadget. Each bag contains at most $4k \cdot t$ from the set V_i and at most 6 from any clause gadget.

This yields a path decomposition of width at most $4kt + 6 \leq 10k \log |X_{i,j}|$ which is of the form $g(k)\log(n)$ for n the number of vertices of G and $g(k)$ a function of the problem we reduced from, as desired. Since G can be constructed in logarithmic space, the result follows. □

We now briefly discuss the Log-Pathwidth Clique problem. We are given a graph $G = (V, E)$ with a path decomposition of width ℓ, and integer K and ask if there is a clique in G with at least K vertices. Let $k = \lceil \ell / \log |V| \rceil$.

The problem appears to be significantly easier than the corresponding versions of Dominating Set and Independent Set, mainly because of the property that for each clique W, there must be a bag of the path decomposition that contains all vertices of W (see e.g., [22].) Thus, the problem reverts to solving $O(n)$ instances of Clique on graphs with $O(k \log n)$ vertices. This problem is related to a problem called Mini-Clique where the input has a graph with a description size that is at most $k \log n$. Mini-Clique is M[1]-complete under FPT Turing reductions (see [29, Corollary 29.5.1]), and is a subproblem of our problem, and thus Log-Pathwidth Clique is M[1]-hard. However, instances of Log-Pathwidth Clique can have description sizes of $\Omega(\log^2 n)$. The result below shows that it is unlikely that Log-Pathwidth Clique is XNLP-hard.

Proposition 19. Log-Pathwidth Clique *is in* W[2].

Proof. Suppose that we are given a graph $G = (V, E)$ and path decomposition $(X_1, \ldots, X_r)$ of G of width at most $k \log n - 1$. Let K be a given integer for which we want to know whether G has a clique of size K. We give a Boolean expression F of polynomial size that can be satisfied with exactly $2k + 2$ variables set to true, if and only if G has a clique of size K. (This shows membership in W[2].)

First, we add variables $b_1, \ldots, b_r$ that are used for selecting a bag; we add the clause $\bigvee_{1 \leq i \leq r} b_i$.

For each bag X_i, we build an expression that states that $G[X_i]$ has a clique with r vertices as follows. Partition the vertices of X_i in k groups of at most $\log n$ vertices each. For each group $S \subseteq X_i$, we have a variable $c_{i,S'}$ for each subset S' of vertices of the group that induces a clique in G. (We thus have at most kn such variables per bag.) The idea is that when $c_{i,S'}$ is true, then the vertices in S' are part of the clique, and the other vertices in the group are not. The variable $c_{i,S'}$ is false if the bag X_i is not selected (we add the clause $b_i \vee \neg c_{i,S'}$). For each group S, we add a clause that is the disjunction over all cliques $S' \subseteq S$ of $c_{i,S'}$ and $\neg b_i$. This encodes that if the bag X_i is selected, then we need to choose a subset S' from the group S.

With the b- and c-variables, we specified a bag and a subset of the vertices of the bag. It remains to add clauses that verify that the total size of all selected subsets equals K, and that the union of all selected subsets is a clique. For the latter, we add a clause $\neg c_{i,S'} \vee \neg c_{i,S''}$ for each pair S', S'' of subsets of different groups for which $S' \cup S''$ does not induce a clique. For the former, we number the groups $1, 2, \ldots, k$, and create variables $t_{j,q}$ for $j \in [1, k]$ and $q \in [0, K]$. The variable $t_{j,q}$ expresses that we have chosen q vertices in the clique from the first j groups of the chosen bag. For each $j \in [1, k]$, we add the clause $\bigvee_q t_{j,q}$. To enforce that the t-variables indeed give the correct clique sizes, we add a large (but polynomial) collection of clauses. We add a variable $t_{0,0}$ that must be true (using a one-literal clause). For the jth group with vertex set $S \subseteq X_i$, each clique $S' \subseteq S$, and each $q, q' \in [0, r]$ (with $q = 0$ when $j = 1$), whenever $q + |S'| \neq q'$, we have a clause $\neg t_{j-1,q} \vee \neg c_{i,S'} \vee \neg t_{j,q'}$. Finally, we require (with a one-literal clause) that $t_{k,K}$ holds. If there is a satisfying assignment, then the union of all sets S' for which $c_{i,S'}$ is true, forms a clique of size K in the graph.

We allow $2k + 2$ variables to be set to true: one variable b_i to select a bag, one for $t_{0,0}$, one for a $c_{i,S'}$-variable in each of the k groups and one $t_{j,q}$-variable for each $j \in [1, k]$ (where for $j = k$, $q = K$ is enforced). □

4.3. Scheduling with precedence constraints

In 1995, Bodlaender and Fellows [11] showed that the problem to schedule a number of jobs of unit length with precedence constraints on K machines, minimizing the makespan, is W[2]-hard, with the number of machines as parameter. A closer inspection of their proof shows that W[2]-hardness also applies when we take the number of machines and the width of the partial order as combined parameter. In this subsection, we strengthen this result, showing that the problem is XNLP-complete (and thus also hard for all classes W[t], $t \in \mathbf{Z}^+$.) In the notation used in scheduling literature to characterize scheduling problems, the problem is known as $P|prec, p_j = 1|C_{\max}$, or, equivalently, $P|prec, p_j = p|C_{\max}$.

Scheduling with Precedence Constraints
Given: K, D positive integers; T set of tasks; $\prec$ partial order on T of width w.
Parameter: $K + w$.
Question: Is there a schedule $f : T \to [1, D]$ with $|f^{-1}\{i\}| \leq K$ for all $i \in [1, D]$ such that $t \prec t'$ implies $f(t) < f(t')$.

In other words, we parameterize $P|prec, p_j = 1|C_{max}$ by the number of machines and the width of the partial order.

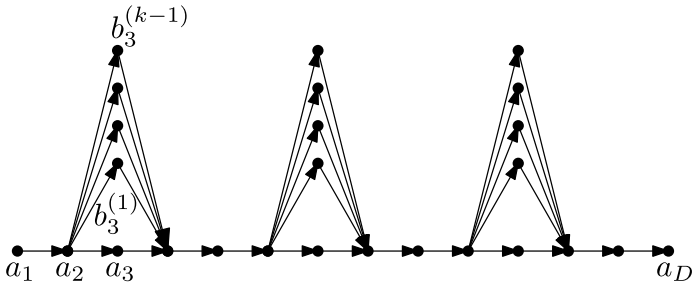

Fig. 5. Illustration for a time sequence with time indicators. E.g., in the example, $3 \in I$.

Theorem 20. Scheduling with Precedence Constraints *is* XNLP*-complete.*

Proof. To see that the problem is in XNLP, define $f^{-1}\{1\}, f^{-1}\{2\}, \ldots, f^{-1}\{D\}$ in order as follows. For $i \in [0, D-1]$, we temporarily store a set $S_i \subseteq T$ containing the maximal elements of $f^{-1}[1, i]$, initializing $S_0 = \emptyset$. Each such set S_i forms an antichain and hence has size at most w, the width of the partial order on T. Once S_{i-1} is defined for some $i \in [1, D]$, we define $f^{-1}\{i\}$ by selecting up to K tasks to schedule next, taking a subset of the minimal elements of the set of "remaining tasks"

$$R_i = \{t \in T \mid t \not\prec s \text{ for all } s \in S_{i-1}\}.$$

By definition, for $t \in R_i$, we find that $t \not\prec s$ for all $s \in f^{-1}[1, i-1]$. In order to update S_i given S_{i-1}, we add all selected elements (which must be maximal) and remove the elements $s \in S_{i-1}$ for which $s \prec t$ for some $t \in f^{-1}\{i\}$. After computing S_i and outputting $f^{-1}\{i\}$, we remove S_{i-1} and $f^{-1}\{i\}$ from our memory. At any point in time, we store at most $O((K + w)\log|T|)$ bits.

To prove hardness, we transform from the Accepting NNCCM problem, with k given counters with values in $[0, n]$ and set of checks $(s_1, \ldots, s_r)$. We adapt the proof and notation of Bodlaender and Fellows [11].

We create $K = 2k + 1$ machines and a set of tasks T with a poset structure on this of width at most $3(k + 1)$. Let $c = (kn + 1)(n + 1)$. The deadline is set to be $D = cr + n + 1$. We will construct one sequence of tasks for each of the k counters, with $kn + 1$ repetitions of $n + 1$ time slots for each of $s_1, \ldots, s_r$ (one for each value in $[0, n]$ that it might like to check), and then n extra time slots for "increasing" counters. Each $t \in [1, D]$ can be written in the form

$$t = (j-1)c + \alpha(n+1) + y$$

for some $y \in [1, n+1]$, $j \in [1, r]$, $\alpha \in [0, kn]$. We define $j(t)$ and $\alpha(t)$ to be the unique values of j and α respectively for which this is possible. Our construction has the following components.

- **Time sequence.** We create a sequence $a_1 \prec a_2 \prec \ldots \prec a_D$ of tasks that represents the time line.
- **Time indicators.** By adding a task b with $a_{t-1} \prec b \prec a_{t+1}$, we ensure that at time i one less machine is available. We place $k-1$ such tasks at special *indicator times*. That is, we set

  $$I = \{(j-1)c + \alpha(n+1) + n + 1 \mid j \in [1, r],\ \alpha \in [0, kn]\}$$

  and create a task $b_t^{(x)}$ with $a_{t-1} \prec b_t^{(x)} \prec a_{t+1}$ for each $x \in [1, k-1]$ and $t \in I$.
- **Counter sequences.** For each $i \in [k]$, we create a sequence $c_1^{(i)} \prec c_2^{(i)} \prec \ldots \prec c_{D-n}^{(i)}$ of tasks. If $c_{t-\ell}^{(i)}$ is planned in at the same moment as time vertex a_t (for some $\ell \in [0, n]$ and $t \in [n+1, D]$) then this is interpreted as counter i taking the value ℓ at that time.
- **Check tasks.** Let $t = (j-1)c + \alpha(n+1) + n + 1 \in I$ be the indicator time for $j \in [1, r]$ and $\alpha \in [0, kn]$. If $s_j = (i_1, i_2, r_1, r_2)$, then for $x \in \{1, 2\}$, we add a check task $d_{t-r_x}^{(i_x)}$ "parallel" to $c_{t-r_x}^{(i_x)}$, that is, we add the precedence constraints

  $$c_{t-r_x-1}^{(i_x)} \prec d_{t-r_x}^{(i_x)} \prec c_{t-r_x+1}^{(i_x)}.$$

Fig. 5 gives an illustration of the construction.

Intuitively, the repetitions allow us to assume that concurrent with each time task are exactly k tasks of the form $c_y^{(i)}$. For most $t \in [1, D]$, we are happy for a potential $d_y^{(i)}$ to be scheduled simultaneously (since we have $2k$ machines available). However, when $t \in I$ then we only have access to $k + 1$ machines, so we may have at most one $d_y^{(i)}$ at this time; if t "corresponds" to $s_j = (i_1, i_2, r_1, r_2)$, then only $d_{t-r_1}^{(i_1)}, d_{t-r_2}^{(i_2)}$ can potentially be planned simultaneously with a_t, which happens exactly if s_j rejects (due to the i_xth counter being at position r_x for both $x \in \{1, 2\}$).

The width w of the defined partial order is at most $3(k+1)$. We now claim that there is a schedule if and only if the given machine accepts.

Suppose that the machine accepts. Let $(p_1, \ldots, p_r)$ with $p_j = (p_j^{(1)}, \ldots, p_j^{(k)}) \in [0,n]^k$ be the positions of the k counters before s_j is checked, for $j \in [1,r]$. We schedule for $t \in [1,D],\ x \in [1,k-1],\ i \in [1,k]$, when defined,

$$f(a_t) = t,$$
$$f(b_t^x) = t,$$
$$f(c_t^{(i)}) = t + p_{j(t)}^{(i)},$$
$$f(d_t^{(i)}) = t + p_{j(t)}^{(i)}.$$

Recall that $j(t)$ is the unique value $j \in [1,r]$ for which we can write $t = (j-1)c + \alpha(n+1) + y$ for some $\alpha \in [0,kn]$ and $y \in [1,n+1]$. At each $t \in [1,D]$, $f^{-1}\{t\}$ contains exactly one a_t and, because the counters are non-decreasing, at most one $c_{t'}^{(i)}$ and at most one $d_{t'}^{(i)}$ for each $i \in [1,k]$. So when $t \notin I$ (and hence b_t^x is not defined), $|f^{-1}\{t\}| \leq 2k+1$ as desired.

Let $t \in I$ and $j = j(t) \in [1,r]$, $\alpha = \alpha(t) \in [0,kn]$. If $d_{t'}^{(i)} \in f^{-1}\{t\}$ for some $i \in [1,k]$, then $t' = t - p_j^{(i)}$ and $s_j = (i_1, i_2, r_1, r_2)$ with $i_x = i,\ r_x = p_j^{(i)}$ for some $x \in \{1,2\}$. Since the machine accepts, there can be at most one such i. Hence $f^{-1}\{t\}$ contains at most one check task when $t \in I$. We again find

$$|f^{-1}\{t\}| \leq \underbrace{1}_{a_t} + \underbrace{(k-1)}_{b_t^{(x)}} + \underbrace{k}_{c_{t'}^{(i)}} + \underbrace{1}_{d_{t'}^{(i)}} \leq 2k+1.$$

Suppose now that a schedule $f : T \to [1,D]$ exists. By the precedence constraints, we must have $f(a_t) = t$ for all $t \in [1,D]$ and $f(b_t^x) = t$ for all $t \in I$ and $x \in [1,k-1]$. Let $j \in [1,r]$ be given. Let

$$C = \{c_t^{(i)} \mid j(t) = j,\ i \in [1,k]\}.$$

For at least one $\alpha \in [0,kn]$, it must be the case that $|f^{-1}\{t\} \cap C| = k$ for all t with $\alpha(t) = \alpha$ and $j(t) = j$. This is because for each of the k counter sequences, there are at most n places where we may "skip"; this is why we repeated the same thing $kn+1$ times. We select the smallest such α. Note that for all $t \in T$ with $\alpha(t) = \alpha$ and $j(t) = j$, we also find that $f(c_t^{(i)}) = f(d_t^{(i)})$ for all $i \in [1,k]$.

There is a unique $t \in I$ with $\alpha(t) = \alpha$ and $j(t) = j$; for this choice of t, and for all $i \in [1,k]$, we set $p_j^{(i)}$ to be the value for which $c_{t-p_j^{(i)}}^{(i)} \in f^{-1}\{t\}$. This defines a vector $p_j \in [0,n]^k$ of positions which we set the counters in before s_j is checked. Note that the values of the counters are non-decreasing, and that s_j does not fail since $f^{-1}\{t\}$ contains at most one $d_{t'}^{(i)}$ (due to its size constraints), which implies that s_j accepts. □

We remark that it is unclear if the problem is in XNLP when we take only the number of machines as parameter. In fact, it is a longstanding open problem whether Scheduling with Precedence Constraints is NP-hard when there are three machines (see e.g., [37,49]).

4.4. Uniform emulation of weighted paths

In this subsection, we give a proof that the Uniform Emulation of Weighted Paths problem is XNLP-complete. The result is a stepping stone for the result that Bandwidth is XNLP-complete even for caterpillars with hair length at most three (see the discussion in Subsection 4.5).

The notion of *(uniform) emulation* of graphs on graphs was originally introduced by Fishburn and Finkel [36] as a model for the simulation of computer networks on smaller computer networks. Bodlaender [5] studied the complexity of determining for a given graph G and path P_m if there is a uniform emulation of G on P_m. In this subsection, we study a weighted variant, and show that already determining whether there is a uniform emulation of a weighted path on a path is hard.

An *emulation* of a graph $H = (V,E)$ on a graph $G = (W,F)$ is a mapping $f : V \to W$ such that for all edges $\{v,w\} \in E$, $f(v) = f(w)$ or $\{f(v), f(w)\} \in F$. We say that an emulation is *uniform* if there is an integer c, such that $|f^{-1}\{w\}| = |\{v \mid f(v) = w\}| = c$ for all $w \in W$. We call c the emulation factor.

Determining whether there is a uniform emulation of a graph H on a path P_m is NP-complete, even for emulation factor 2, if we allow H to be disconnected. The problem to determine for a given *connected* graph H if there is a uniform emulation of H on a path P_m belongs to XP with the emulation factor c as parameter [5].

Recently, Bodlaender [6] looked at the weighted variant of uniform emulation on paths, for the case that H is a path. Now, we have a path P_n and a path P_m, a weight function $w : [1,n] \to \mathbf{Z}^+$ and ask for an emulation $f : [1,n] \to [1,m]$, such that there is a constant c with $\sum_{i \in f^{-1}\{j\}} w(i) = c$ for all $j \in [1,m]$. Again, we call c the emulation factor.

It is not hard to see that the problem, given n, m, and weight function $w : [1, n] \to \mathbf{Z}^+$, to determine if there is a uniform emulation of P_n on P_m with emulation factor c is in XP, with the emulation factor c as parameter; the dynamic programming algorithm from [5] can easily be adapted.

As an intermediate step for (W[t])-hardness proof for Bandwidth, Bodlaender [6] showed that Uniform Emulation of Weighted Paths is hard for all classes W[t], $t \in \mathbf{Z}^+$. In the current subsection, we give a stronger result, and show the same problem to be XNLP-complete. Our proof is actually simpler than the proof in [6] – by using Accepting NNCCM as starting problem, we avoid a number of technicalities.

Uniform Emulation of Weighted Paths
Input: Positive integers n, m, c, weight function $w : [1, n] \to [1, c]$.
Parameter: c.
Question: Is there a function $f : [1, n] \to [1, m]$, such that f is a uniform emulation of P_n on P_m with emulation factor c, i.e., $|f(i) - f(i+1)| \leq 1$ for all $i \in [1, n-1]$ and $\sum_{i \in f^{-1}\{j\}} w(i) = c$ for all $j \in [1, m]$?

Theorem 21. Uniform Emulation of Weighted Paths *is* XNLP-*complete.*

Proof. We first briefly sketch a variant of the dynamic programming algorithm (see [5]) that shows membership in XNLP. For i from 1 to m, guess which vertices are mapped to i. Keep this set, and the sets for $i-1$ and $i-2$ in memory. We reject when neighbors of vertices mapped to $i-1$ are not mapped to $\{i-2, i-1, i\}$ (or not to $\{1, 2\}$ or $\{m-1, m\}$ for $i = 1$ or $i = m$ respectively) or when the total weight of vertices mapped to i does not equal c. We also check whether $cm = \sum_{i=1}^{n} w(i)$ to ensure that each vertex gets mapped to a single index. Since P_n is connected, and $f(i)$ is defined for some $i \in [1, n]$, we find that $f(i)$ is defined for all $i \in [1, n]$ (since our check ensures that $f(j)$ is defined for each neighbor j of i in P_n, and then also for all neighbors of j etcetera). We have $O(c)$ vertices from P_n in memory, and thus $O(c \log n)$ bits.

To show that the problem is XNLP-hard, we use a transformation from Accepting NNCCM. Suppose that we are given an NNCCM(k, n, s), with s a sequence of r checks. We build an instance of Uniform Emulation of Weighted Paths as follows.

First, we define a number of constants:

- $d_1 = 3k + 2$,
- $d_2 = k \cdot d_1 + 1$,
- $d_3 = k \cdot d_2 + 1$,
- $c = 2k \cdot d_3 + 1$,
- $n_0 = 3n + 1$,
- $M = 1 + (r+1) \cdot n_0$.

We construct a weighted path P_N for some value N whose value follows from our construction below, and ask for a uniform emulation of P_N to P_M, with emulation factor c. The path P_N has three subpaths with different functions. We concatenate these in order to obtain P_N.

- The floor. We ensure that the ith vertex of the floor must be mapped to the ith vertex of P_M. We will use this to be able to assume that the counter components always "run from left to right".
- The k counter components. Each counter component models the values for one of the counters from the NNCCM and the goal is to ensure the emulation of this part is possible if and only if all the checks succeed.
- The filler path. This is a technical addition aimed to ensure that a total weight of c gets mapped to j for all $j \in [1, M]$.

The floor consists of the following M (weighted) vertices.

- A vertex of weight $c - k \cdot d_2$.
- A path with $M-2$ vertices. The ith vertex of P_N (and hence the $(i-1)$th of this part) has weight $c - 2d_1 + 1$ if $i = n_0 \cdot j + 1$ for some j and weight $c - 3d_1$ otherwise. The positions of the higher weight of $c - 2d_1 + 1$ are called *test positions*.
- A vertex of weight $c - k \cdot d_3 - 1$.

We have k counter components, that give after the floor, the successive parts of the "large path". The qth counter component has the following successive parts.

- We start with $M-2$ vertices of weight 1.
- We then have a vertex of weight d_2; this vertex is called the *left turning point*.
- Then we have $M - 2 + n$ vertices of weight either 1 or d_1. The ith vertex has weight d_1 if and only if $i = n_0 \cdot j + \alpha$ and the check s_j verifies whether counter q is equal to α. The ith vertex has weight 1 otherwise. We call this the *main path* of the counter component.

- A vertex of weight d_3. This vertex is called the *right turning point* of the counter component.

Assuming the left and right turning points are mapped to 1 and M, the main path of the gadget will be mapped between 2 and $M-1$. These main paths will tell us what the values of the counters are at any moment. The number of shifts then stands for the number a counter gadget represents, i.e. if the ith vertex of the main path is mapped to $i-\alpha+1$, then α is the value of the counter at that moment. In this way, assuming that left and right turning points are mapped to 1 and M, the value of counters can only increase.

We add a "filler path" to ensure the total weight of all vertices of P_M is at least Mc, by adding a path of $\min\{\gamma - Mc, 0\}$ vertices of weight 1.

Claim 21.1. *The* NNCCM *described by* (k, n, s) *accepts if and only if* P_N *with the defined vertex weights has a uniform emulation on* P_M*,*

Proof. If the NNCCM accepts, then we can build a uniform emulation as follows. Fix some accepting run and let $q(j)$ be the value of counter q at the jth check in this run.

Map the ith vertex of the floor to i. Now, successively map each counter component as follows. We start by "moving back to 1", i.e., we map the $M-2$ vertices of weight 1 to $M-1$, $M-2$, …, 2 and map the left turning point to 1. To map the vertices on the main path of this counter component, we use the following procedure. For convenience, we define $q(r+1)=n$.

- Set $i=1$, $p=2$. We see $p \in [1, M]$ as a position on P_M, and $i \in [1, M+n]$ as the number of the vertex from the main path we are currently looking at. The number $i-(p-1)$ represents the value of the counter.
- Set j to be the smallest integer with $p \le n_0 \cdot j + 1$; if $j \le r$, then $n_0 \cdot j + 1$ is the first test position (defined in the floor gadget) after p.
- If $i-(p-1) < q(j)$, and p is in the interval $[n_0 \cdot (j-1) + 1 + n + 1, n_0 \cdot j - n]$, then we map both the ith and the $(i+1)$th vertex of the main path to p. We increase i by 2 and p by 1; the increase of $i-(p-1)$ by one represents the increase of the counter by one. Otherwise, we map the ith vertex of the main path to p, and increase both i and p by one.

Since $q(r+1)=n$, in the end we will map $M-2+n$ vertices to $M-2$ positions.

Note that the final step above does not decrease the value of $i-(p-1)$. The interval $[n_0 \cdot (j-1)+1+n+1, n_0 \cdot j - n]$ contains at least $n \ge q(j)$ integers, and thus we may map two vertices to the same position until we obtain $i-(p-1) \ge q(j)$. We then stop increasing the value $i-p+1$, so $i-p+1=q(j)$ at the jth test position $p = n_0 \cdot j + 1$, and map a single vertex to this.

What vertex gets mapped to the jth test position? If the qth counter does not participate in the jth check, then the vertex will have weight one. Otherwise, suppose that check s_j verifies whether counter q is equal to the value α. The vertex i mapped to the jth test position has weight d_1 if and only if $i = n_0 \cdot j + \alpha$, which is equivalent to $q(j)=\alpha$ since $i-(p-1)=q(j)$ and $p = n_0 \cdot j + 1$. As we were considering an accepting run for the NNCCM, there is at most one counter component that maps a vertex of weight d_1 (rather than 1) to the jth test position.

After we have placed the main path, we send the right turning point to M and continue with the next counter component or, if $q=k$, the filler path. At this point, 1 and M have received weight exactly c (from one floor vertex and k turning points). For all test positions, we have a floor vertex of weight $c-2d_1+1$, at most one heavy vertex of weight d_1, and at most $2k$ vertices of weight one (at most two per counter component). For a vertex on P_M that is not 1, M or a test position, we have a floor vertex of weight $c-3d_1$, at most two heavy vertices of weight d_1 and at most $3k$ vertices of weight one (per counter component, it obtains at most two vertices for the main path and one for path going left). In all cases, we have a weight less than c.

We now use the filler path to make the total weight equal to c everywhere. (By definition, the filler path has enough vertices of weight 1 available for this.)

Conversely, suppose that we have a uniform emulation f. We cannot map two floor vertices to the same vertex as each has a weight larger than $c/2$. As we have M floor vertices, one of the following two statements holds:

1. For all i, the ith floor vertex is mapped to i.
2. For all i, the ith floor vertex is mapped to $M-i+1$.

Without loss of generality, we assume that the ith floor vertex is mapped to i.

We first show that each right turning point is mapped to M and each left turning point is mapped to 1. Since M is the only vertex where the available weight (which is at most c minus the weight of the floor vertex) is at least the weight of a right turning point, all right turning points must be mapped to M. After this, there is no more weight available on this vertex, and the only vertex where a left turning point can fit is 1.

We now use the test positions to define values for the counters. Because the main path of the counter component is from a left turning point (mapped to 1) to a right turning point (mapped to M), at least one vertex of the main path is

mapped to the jth test position $n_0 \cdot j + 1$. If the ith vertex of the main path is mapped to the test position, then we set $q(j) = i - n_0 \cdot j$. (If multiple vertices are mapped to the test position, we choose arbitrarily one of those values.) We find that the values we defined for the counters are non-decreasing since the difference between $i = i(j)$ and $n_0 \cdot j$ can only become larger for larger j.

We are left to prove that this defines an accepting sequence. Assume towards a contradiction that the check $s_j = (q, q', \alpha_q, \alpha_{q'})$ fails because of $q(j) = \alpha_q$ and $q'(j) = \alpha_{q'}$. That would imply both the $(n_0 \cdot j + q(j))$th vertex from the qth counter component and the $(n_0 \cdot j + q'(j))$th vertex from the q'th counter component were mapped to test position $n_0 \cdot j + 1$. Both these vertices have weight d_1, by the construction of the main path of counter components. Since we already have a total weight of $c - 2d_1 + 1$ from the floor gadget, the total weight mapped to the jth test position is $> c$. This is a contradiction. Hence the given solution for the NNCCM instance is a valid one. □

Theorem 21 now follows from Claim 21.1 and observing the resources (logarithmic space, fpt-time) by the transformation. □

In a later proof, we need the following variant of Theorem 21.

Corollary 22. Uniform Emulation of Weighted Paths *with* $f(1) = m$ *is* XNLP*-complete.*

Proof. In the proof of Theorem 21, we showed that if there is a solution, then there is one with $f(1) = 1$. By symmetry, we can also require that $f(1) = M$ in this proof, where M is the value set for the variable called m in the problem formulation of Uniform Emulation of Weighted Paths. □

4.5. Bandwidth

In this subsection, we discuss the XNLP-completeness problem of the Bandwidth problem. The question where the parameterized complexity of Bandwidth lies was actually the starting point for the investigations whose outcome is reported in this paper; with the main result of this subsection (Theorem 24) we answer a question that was asked over a quarter of a century ago.

In the Bandwidth problem, we ask if the bandwidth of a given graph G is at most k. The problem models the question to permute rows and columns of a symmetric matrix, such that all non-zero entries are at a small "band" along the main diagonal. Already in 1976, the problem was shown to be NP-complete by Papadimitriou [46]. Later, several special cases were shown to be hard; these include caterpillars with hairs of length at most three [45]. A *caterpillar* is a tree where all vertices of degree at least three are on a common path; the *hairs* are the paths attached to this main path.

We are interested in the parameterized variant of the problem, where the target bandwidth is the parameter:

Bandwidth
Given: Integer k, undirected graph $G = (V, E)$
Parameter: k
Question: Is there a bijection $f : V \to [1, |V|]$ such that for all edges $\{v, w\} \in E$: $|f(v) - f(w)| \leq k$?

Bandwidth belongs to XP. In 1980, Saxe [50] showed that Bandwidth can be solved in $O(n^{k+1})$ time; this was later improved to $O(n^k)$ by Gurari and Sudborough [38]. In 1994, Bodlaender and et. [12] reported that Bandwidth for trees is W[t]-hard for all $t \in \mathbf{Z}^+$ — the proof of that fact was published 26 years later [6]. A sketch of the proof appears in the monograph by Downey and Fellows [28]. More recently, Dregi and Lokshtanov [30] showed that Bandwidth is W[1]-hard for trees of pathwidth at most two. In addition, they showed that there is no algorithm for Bandwidth on trees of pathwidth at most two with running time of the form $f(k)n^{o(k)}$ assuming the Exponential Time Hypothesis.

Below, we show that Bandwidth is XNLP-complete for caterpillars with hairs of length at most three.[6] This reduction uses gadgets from the NP-completeness proof by Monien [45].

Lemma 23. Bandwidth *is in* XNLP.

Proof. This can be seen in different ways: one can look at the XP algorithms for Bandwidth from [50] or [38], and observe that when instead of making full tables in the dynamic programming algorithm, we guess from each table one entry, one obtains an algorithm in XNLP.

Alternatively, we loop over all connected components W of G, compute its size and for each, guess for i from 1 to $|W|$ the ith vertex in the linear ordering, i.e., $f^{-1}(i)$. Keep the last $2k + 1$ guessed vertices in memory, and verify that all neighbors of $f^{-1}(i - k)$ belong to $f^{-1}[i - 2k, i]$. □

[6] The reduction from Uniform Emulation of Weighted Paths to this problem was reported in [6], at WG 2020, where it was used to show that Bandwidth is W[t]-hard for all t.

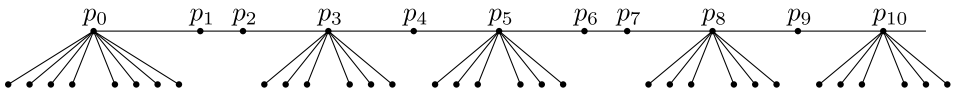

Fig. 6. First part of the caterpillar: left barrier and first part of the floor.

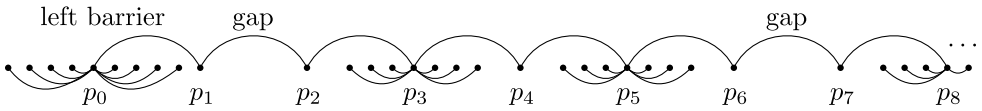

Fig. 7. A layout of the left barrier and first part of the floor. The layout creates gaps where the weighted path gadgets should fit.

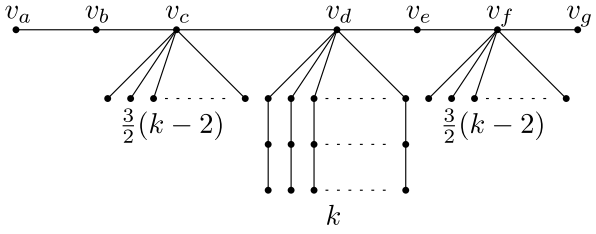

Fig. 8. The turning point, after Monien [45]

Theorem 24. BANDWIDTH *is* XNLP*-complete, when restricted to caterpillars with hair length at most three.*

Proof. To prove Theorem 24, we transform from the WEIGHTED PATH EMULATION WITH $f(1) = M$ problem (cf. 22).

Let P_N and P_M be paths with weight function $w : \{1, \ldots, N\} \to \{1, \ldots, c\}$, and let $c = \sum_{i=1}^{N} w(i)/M$ be the emulation factor. (Note that we may always restrict our weights to be between 1 and c since otherwise no emulation can exist.) Thus, $\sum_{i=1}^{N} w(i) = cM$. Recall that we parameterized this problem by the value c.

Assume that $c > 6$; otherwise, obtain an equivalent instance by multiplying all weights by 7.

Let $b = 12c + 6$. Let $k = 9bc + b$. Note that k is even. We give a caterpillar $G = (V, E)$ with hair length at most three, with the property that P_N has a uniform emulation on P_M, if and only if G has bandwidth at most k.

G is constructed in the following way:

- We have a *left barrier*: a vertex p_0 which has $2k - 1$ hairs of length one, and is neighbor to p_1.
- We have a path with $5M - 3$ vertices, $p_1, \ldots, p_{5M-3}$. As written above, p_1 is adjacent to p_0. Each vertex of the form $5i - 2$ or $5i$ $(1 \leq i \leq M - 1)$ receives $2k - 2b$ hairs of length one. See Fig. 6 and Fig. 7. We call this part the *floor*.
- Adjacent to vertex p_{5M-3}, we add the *turning point* from the proof of Monien [45]. We have vertices $v_a = p_{5M-3}$, v_b, v_c, v_d, v_e, v_f, v_g, which are successive vertices on a path. I.e., we identify one vertex of the turning point (v_a) with the last vertex of the floor p_{5M-3}. To v_c, we add $\frac{3}{2}(k-2)$ hairs of length one; to v_d, we add k hairs of length three, and to v_f we add $\frac{3}{2}(k-2)$ hairs of length one. Note that this construction is identical to the one by Monien [45]; vertex names are chosen to facilitate comparison with Moniens proof. See Fig. 8.
- Add a path with $6N - 5$ vertices, say $y_1, \ldots y_{6N-5}$, with y_1 adjacent to v_g. To each vertex of the form y_{6i-5}, add $9b \cdot w(i)$ hairs of length one. We call this part the *weighted path gadget*.
- Note that the number of vertices that we defined so far and that is not part of the turning point equals $2k + 5M - 3 + 2(M-1)(2k-2b) + 6N - 5 + 9b\sum_{i=1}^{N} w_i = 5M + 4Mk - 2k - 4Mb + 4b + 9bcM$. Let this number be α. One easily sees that $\alpha \leq (5M-2)k - 1$. Add a path with $(5M-2)k - 1 - \alpha$ vertices and make it adjacent to y_{6n}. We call this the *filler* path.

Let G be the remaining graph. Clearly, G is a caterpillar with hair length at most three. It is interesting to note that the hair lengths larger than one are only used for the turning point.

The correctness of the construction follows from the following lemma.

Lemma 25. *Suppose P_N has a uniform emulation g on P_M with emulation factor c with $f(1) = M$. Then the bandwidth of G is at most k.*

Proof. Suppose we are given a uniform emulation $f : \{1, \ldots, N\} \to \{1, \ldots, M\}$ of P_N on P_M (for weight function w). We construct a layout g of G with bandwidth at most k as follows.

For $1 \leq i \leq 5M - 3$, set $g(p_i) = (i+1)k + 1$. Each of the hairs of p_0 is mapped by g to a different value in $\{1, \ldots, 2k\} \setminus \{k+1\}$. (We later define how to map the hairs of other vertices of the floor.)

For the turning point, we set $g(v_g) = g(v_a) - 1 = (5M-2)k$. The other vertices of the turning point receive values larger than $(5M-2)k$; the remaining layout for the turning point is identical to the one described by Monien [45].

Each position of the form $(i+1)k+1+j$ for $1 \leq i \leq 5M-3$ with $j \in \{-1,+1\}$ is said to be *reserved*. Note these are the positions before and after the image of the vertices p_i. These positions are reserved for the filler path.

We call the positions between $g(p_{5i-4})$ and $g(p_{5i-3})$ the ith *gap*, for $1 \leq i \leq M$. Gaps 1 till $M-1$ have at this point $k-1$ unused integers of which two are reversed; in gap M we have $k-1$ unused integers (v_g is mapped in the gap) of which one is reserved. (The gaps are used to layout "most of" the vertices of the weighted path gadget.) See Fig. 7 for an illustration.

We now show how to map the weighted path gadget. We start with fixing the images of vertices of the form y_{6i-5}. For each i, $1 \leq i \leq N$, set $g(y_{6i-5})$ to be the first integer in the $f(i)$th gap that is so far not used and is not reserved. As there are at most c vertices of P_N mapped by f to a specific vertex of P_M, we have that $g(y_{6i-5})$ is in the interval $[f(p_{5(i-1)})+3, f(p_{5(i-1)})+c+2]$, thus $|g(y_{6i-5}) - g(y_{6i+1})| \leq 5k+c$; i.e., they can be in the same, or neighboring gaps.

As $f(1) = M$, we have that y_1 is placed in the Mth gap, as is, by construction its neighbor v_g; thus $|g(y_1) - g(v_g)| \leq k$.

The next step is to map the vertices of the form y_j with j not of the form $6i-5$. We must do this in such a way, that neighboring vertices are mapped to integers that differ by at most k, taking into account the mapping vertices of the form y_{6i-5} as defined above.

So, for $i \in \{1, \ldots, N-1\}$, the vertices on the path from y_{6i-5} to y_{6i+1} are mapped as follows. If y_{6i-5} and y_{6i+1} are mapped to the same gap, then map the five vertices on this path also to this gap, to so far unused and unreserved values. Otherwise, define a "largest possible step to the right" from an integer α as the largest integer that is at most $\alpha + k$ and is not used or reserved. If $g(y_{6i-5}) < g(y_{6i+1})$, then put $g(y_{6i-4})$ at the largest possible step to the right from $g(y_{6i-5})$, and then $g(y_{6i-3})$ at the largest possible step to the right from $g(y_{6i-4})$, and continue doing so, till we have placed $g(y_{6i})$. If $g(y_{6i+1}) < g(y_{6i-5})$, a similar construction is followed.

Claim 25.1. *For each j, $1 \leq j \leq 5M-4$, we map at most $11c$ vertices of the form y_i to integers in the interval $[g(p_j), g(p_{j+1})]$.*

Proof. If the interval is a gap, then for each such y_i, there is a vertex of the form $y_{6i'-5}$ at distance at most five from y_i mapped to this gap. As we have at most c vertices of the form $y_{6i'-5}$ in the gap, we have at most $11c$ vertices of the form y_i mapped to this gap.

If the interval is not a gap, then for each such y_i, there is a vertex of the form $y_{6i'-5}$ at distance at most five from y_i mapped to the last gap before the interval. Thus, we have at most $10c$ vertices of the form y_i mapped to this gap. (A more precise counting argument can give a smaller bound.) □

Claim 25.2. *For all i, $1 \leq i < N$, $|g(y_{6i}) - g(y_{6i+1})| \leq k$.*

Proof. The result clearly holds when y_{6i-5} and y_{6i+1} are placed in the same gap. If they are placed in different gaps, then we consider the case that $g(y_{6i-5}) < g(y_{6i+1})$; the other case is similar.

Note that each largest possible step to the right makes a jump that is at least $k - 22c - 5$: from any k consecutive integers, we have used at most $22c$ for vertices of the form y_i (by Claim 25.1, with vertices possible in two consecutive intervals), reserved four integers, and used one for a vertex of the form p_i.

Recall that $|g(y_{6i-5}) - g(y_{6i+1})| \leq 5k+c$. Five largest jumps to the right bridge at least $5(k-22c-5)$, hence $g(6i+1) - g(6i) \leq 23c + 25 < k$. □

The next step is to map an initial part of the filler part, to "move it" to the last number in the $(M-1)$th gap. Suppose $f(v_N) = i$. Now, map the first vertex of the filler path to $f(p_{5i-3}) - 1$, i.e., the last number in the ith gap (which was reserved), and then, while we did not yet place a vertex on $f(p(5(M-1)-3)) - 1$ (the last number in the $M-1$st gap), map k larger than its predecessor on the path, i.e., the second vertex goes to $g(p_{5i-3}) + k - 1$, the third to $g(p_{5i-3}) + 2k - 1$, etc. In each case, we place the vertex on the last number in the gap; these were reserved so not used by other vertices.

At this point, each open interval $(g(p_i), g(p_{i+1}))$ has at most $11c+1$ used positions, and at most four reserved positions. Thus, at least $k - 1 - (11c+5) \geq k - b = 9bc$ positions are unused and not reserved in this interval.

For each vertex of the form p_{5i-2} or p_{5i} ($1 \leq i \leq M-1$), say $p_{i'}$ we map $k-b$ of its hairs to the interval before it: i.e., to integers in $(g(p_{i'-1}), g(p_{i'}))$ that are unused and not reserved, and $k-b$ hairs to the interval after it, i.e., to integers in $(g(p_{i'}), g(p_{i'+1}))$ that are unused and not reserved.

Claim 25.3. *For each j, $1 \leq j \leq M$, the total number of hairs adjacent to vertices of the form y_{6i-5}, $1 \leq i \leq N$ with y_{6i-5} mapped to the jth gap equals $9bc$.*

Proof. Recall that f is a uniform emulation, and that $c = \sum_{i=1}^{n} w(i)/M$. This implies that for each j, $1 \leq j \leq M$. This implies that $\sum_{i: f(i)=j} w_i = c$.

A vertex y_{6i-5} is placed in the jth gap, if and only if it contributes to this sum. As the number of hairs of y_{6i-5} equals $9b \cdot w(i)$, the result follows. □

For each vertex of the form y_{6i-5}, $1 \leq i \leq N$, we map all its hairs to the gap to which it belongs. I.e., if $f(i) = j$, then all hairs of y_{6i-5} are mapped to unused and not reserved integers in $(g(p_{5j-4}, g(p_{5j-3}))$. From the analysis above, we have sufficiently many such integers to map all hairs.

Finally, we lay out the filler path to fill up all remaining unused values: go from $f(p(5(M-1)-4))-1$ to the last unused number of the Mth gap (which is smaller than $f(p(5(M-1)-4))+k-1$), and then, while there are unused numbers left, place each next vertex of the filler gap on the largest so far unused number. Thus, from this point on, the filler gap goes from right to left, skipping only used numbers. The reserved numbers of the form $(i+1)k+1$ guarantee that by doing so, we never skip more than $k-1$ numbers, thus preserving the bandwidth condition.

This finishes the construction of the linear ordering g. From the analysis above, it follows that g has bandwidth k. □

Lemma 26. *Suppose the bandwidth of G is at most k. Then P_N has a uniform emulation on P_M with emulation factor c with $f(1) = M$.*

Proof. Suppose we have a linear ordering g of G of bandwidth at most k.

First we note that all vertices outside the left barrier and the turning point have to be mapped between the left barrier and the turning point. For the left barrier, this is easy to see: p_0 has $2k$ neighbors, which will occupy all $2k$ positions with distance at most k to $g(p_0)$, and thus all other vertices must be at the same side of p_0 as p_1.

Now, we look at the turning point, and use a result by Monien [45, Lemma 1].

Lemma 27 *(Monien [45]). Let g be a linear ordering of a graph G containing the turning point as a subgraph. Then $|g(v_a) - g(v_g)| \leq 1$, and one of the following two cases holds.*

1. *All vertices in the turning point except v_a and v_g have an image under g that is larger than $\max\{g(v_a), g(v_g)\}$; all vertices in G that do not belong to the turning point have an image that is smaller than $\min\{g(v_a), g(v_g)\}$.*
2. *All vertices in the turning point except v_a and v_g have an image under g that is smaller than $\min\{g(v_a), g(v_g)\}$; all vertices in G that do not belong to the turning point have an image that is larger than $\max\{g(v_a), g(v_g)\}$.*

I.e., we have that v_a and v_g are next to each other; all other vertices of the turning point are at one side of this pair, and all vertices not in the turning point at the other side. Without loss of generality, suppose $g(p_0) < g(v_a)$.

Thus, if we look at the consecutive values $f^{-1}(1), f^{-1}(2), \ldots$, we first see k hairs of p_0, then p_0, then $k-1$ hairs of p_0, then p_1, then all 'other' vertices (i.e., all vertices, except p_0, the hairs of p_0, p_1 and all vertices in the turning point), then we have the two vertices $v_a = p_{5M-3}$ and v_g (in this order, or reversed), and the sequence ends with all other vertices of the turning point.

Claim 27.1. *$g(p_0) = k+1$, and $\{g(v_a), g(v_g)\} = \{(5M-2)k, (5K-2)k+1\}$.*

Proof. p_0 is placed after k hairs, so must have image $k+1$. The total number of vertices not in the turning point equals $(5M-2)k-1$, see the construction of the filler path. The pair $\{g(v_a), g(v_g)\}$ comes after all vertices not in the turning point, and before all other vertices from the turning point, so must go to $(5M-2)k$ and $(5M-2)k+1$. □

Claim 27.2. *For all i, $0 \leq i < 5M-3$, either $g(p_{i+1}) = g(p_i) + k$ or $g(p_{i+1}) = g(p_i) + k - 1$.*

Proof. The path from p_0 to $v_a = p_{5M-3}$ has $M-3$ edges. For each pair p_i, p_{i+1}, the difference of the images is at most k (by the bandwidth condition), but the total of the differences over all pairs on the path must be at least $(M-3) \cdot k - 1$. □

Now, call the positions between $g(p_{5i-5})$ and $g(p_{5i-2})$ the ith *enlarged gap*, for $1 \leq i \leq M-1$; and between $g(5M-5)$ and $g(p_{5M-3})$ the Mth enlarged gap.

Claim 27.3. *Each vertex of the form y_{6j-5} is mapped to an enlarged gap.*

Proof. Suppose not. As $g(y_{6j-5}) > g(p_1)$ and $g(y_{6j-5}) < g_{5M-3}$, by construction of the left barrier and turning point, we have that there is an i with $g(y_{6j-5}) \in [g(p_{5i-2}), g(p_{5i})]$. Now, $[g(p_{5i-3}), g(p_{5i+1})]$ contains $4k-4b$ hairs attached to vertices p_{5i-2} and p_{5i} and $9b \cdot w(j) \geq 9b$ hairs attached to y_{6j-5}. The interval has size at most $4k$, but has $4k+b$ hairs mapped to it; contradiction. □

Now, let $f : \{1, \ldots, N\} \to \{1, \ldots, M\}$, such that $f(j) = i$, if y_{6j-5} is mapped to the ith enlarged gap. By Claim 27.3, f is well defined.

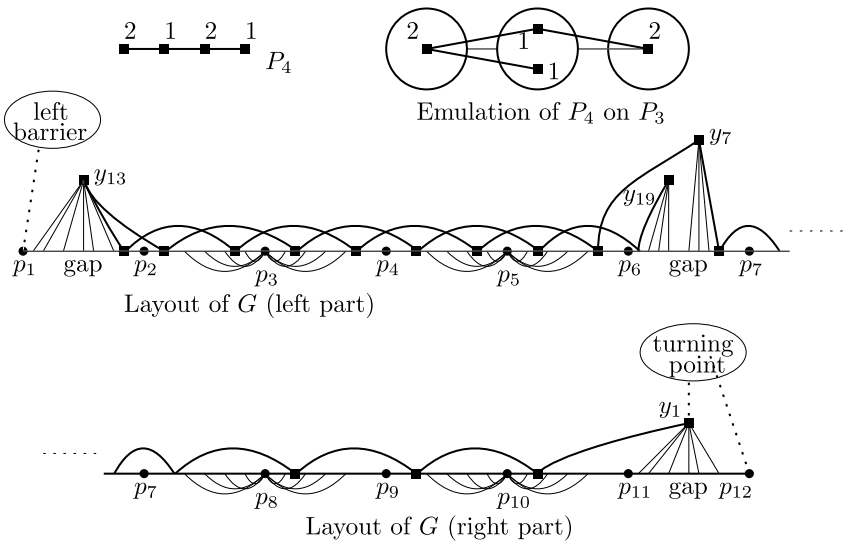

Fig. 9. Illustration of part of the construction. Shown are P_4 with successive vertex weights 2, 1, 2, 1; a uniform emulation of P_4 on P_3 with emulation factor 2; a layout of a part of G (the layout of left barrier and turning point are not explicitly shown). For readability, the layout is shown in two parts.

Claim 27.4. *f is an emulation.*

Proof. Note that there is a path with six edges from a vertex y_{6j-5} to $y_{6(j+1)-5}$. As the distance between vertices p_i in the linear ordering is either k or $k-1$, each edge in this path can jump over at most one vertex of the form p_α. To go from the ith enlarged gap to the $(i+2)$th enlarged gap, we must jump at least seven vertices of the form p_α, so y_{6j-5} and $y_{6(j+1)-5}$ are mapped to the same or neighboring enlarged gaps, thus $|f(j) - f(j+1)| \leq 1$. □

Claim 27.5. *f is uniform.*

Proof. We show that for each i, $\sum_{j:f(j)=i} w(j) \leq c$. As f is uniform, we have that for each i: $\sum_{j:f(j)=i} w(j) = c$.

We only give the proof for $1 < i < M$; the cases $i = 1$ and $i = M$ are similar (using that no vertices can be mapped to values used by the left blockade and turning point) and omitted.

Consider an i, $1 < i < M$. For each j with $f(i) = j$, all $9b \cdot w(j)$ hairs of y_{6j-5} are mapped to the interval $[g(p_{5i-6}), g(p_{5i-1})]$. This interval has size at most $5k$, contains $4k - 4b$ hairs of p_α-vertices, thus can contain at most $8k + 4b$ hairs of vertices y_β. All hairs of vertices y_{6j-5} with $f(j) = i$ must be mapped to this interval. As the number of these hairs equals $9b$ times the sum of the weights of these vertices, we have $\sum_{j:f(j)=i} w(j) \leq c$. □

Finally, we observe that $f(1) = M$: y_1 is incident to v_g, and thus y_1 must be placed in the Mth enlarged gap.

Now, with this last observation, Claims 27.3, 27.4 and 27.5 together show Lemma 26. □

An illustration of the construction of a linear ordering, given a uniform emulation is given in Fig. 9.

As the construction of the caterpillar G can be done in polynomial time and logarithmic space, given M, N and w, Theorem 24 now follows. □

4.6. Acyclic finite state automata intersection

In the Finite State Automata Intersection problem, we are given k deterministic finite state automata on an alphabet Σ and ask if there is a string $s \in \Sigma^*$ that is accepted by each of the automata.

In the overview of the parameterized complexity of various problems in [28], the problem is mentioned to be hard for all classes W[t], $t \in \mathbf{Z}^+$, when either parameterized by the number of machines k or by the combination of the number of machines k and the size of the alphabet Σ; the result is due to Hallett, but has not been published.

More recently, the problem and many variations were studied by Wehar [53]. Amongst others, he showed that Finite State Automata Intersection with the number of machines as parameter is XNL-complete. He also considered the variant where the automata are acyclic.

Acyclic Finite State Automata Intersection
Input: k deterministic finite state automata on an alphabet Σ for which the underlying graphs are acyclic (except for self-loops at an accepting or rejecting state).
Parameter: k.
Question: Is there a string $s \in \Sigma^*$ that is accepted by each of the automata?

Wehar [53, Chapter 5] showed that Acyclic Finite State Automata Intersection is equivalent under LBL-reductions (parameterized reductions that do not change the parameter) to a version of Timed CNTMC (see Section 2.4) where the given time bound T is linear in the sum of the sizes of the automaton and the input. The proof technique of Wehar [53] can also be used to show that Acyclic Finite State Automata Intersection is XNLP-complete. We use a different, simple reduction from Longest Common Subsequence (defined on the next page).

Theorem 28. Acyclic Finite State Automata Intersection *is* XNLP*-complete.*

Proof. For XNLP-membership of Acyclic Finite State Automata Intersection, the argument from [53, Proposition 5.1] can be used: guess the solution string one character at the time, and keep track of the states of the machines during these guesses. To see this does not take too much time, let n be the largest number of vertices in a graph underlying one of the machines. Since the graphs are acyclic, all machines enter an accepting or rejecting state within n transitions.

The simple transformation from Longest Common Subsequence parameterized by the number of strings k to Acyclic Finite State Automata Intersection is given below.

Suppose that we are given k strings $s^1, \ldots, s^k \in \Sigma^*$ and an integer m. The Longest Common Subsequence asks if there is a string $s \in \Sigma^*$ of length at least m which is a subsequence of $s^1, \ldots, s^k$.

We consider $k+1$ automata on alphabet Σ: one for each string s^i and one that checks the length of the solution.

The *length automaton* has $m+1$ states $q_0, \ldots, q_m$, where q_0 is the starting state and q_m is the accepting state. For each $i \in [0, m-1]$, and each $\sigma \in \Sigma$ we have a transition from q_i to q_{i+1} labeled with σ. We also have for each $\sigma \in \Sigma$ a transition (self-loop) from q_m to q_m. The following claim is easy to observe.

Claim 28.1. *The length automaton accepts a string $s \in \Sigma^*$ if and only if the length of s is at least m.*

For each $i \in [1, k]$, we have a *subsequence automaton* for string s^i. Suppose that its length is $|s^i| = t$. The subsequence automaton for s^i has the following $t+2$ states: $q_0, \ldots, q_t, q_R$. All states except for the rejecting state q_R are accepting states, and q_0 is the starting state. The automaton for s^i has the following transitions. For each $j \in [0, t-1]$ and $\sigma \in \Sigma$, if the substring $s^i_{j+1} \cdots s^i_t$ contains the symbol σ, then let $s^i_{j'}$ be the first occurrence of σ in this substring, and take a transition from q_j to $q_{j'}$ labeled with σ. (Recall that for each state q and each $\sigma \in \Sigma$, if no transition out of q labeled with σ was defined in the previous step, then we take a transition from q to q_R labeled with σ.) The name "subsequence automaton" is explained by the following claim.

Claim 28.2. *The subsequence automaton for string s^i accepts a string $s \in \Sigma^*$ if and only if s is a subsequence of s^i.*

Proof. Note that after reading a character, the automaton moves to the index of the next occurrence of this character after the current index. So, when we read a subsequence s, we are in state q_j when j is the smallest integer with s a subsequence of the substring $s^i_1 \ldots s^i_j$. When there is no such next character, then s is not a subsequence and we move to the rejecting state. □

From the claims above, it follows that a string $s \in \Sigma^*$ is a subsequence of $s^1, \ldots, s^k$ of length at least m if and only if s is accepted by both the length automaton and the subsequence automata of $s^1, \ldots, s^k$. We can compute the automata in $O(f(k) + \log n)$ space, given the strings $s^1, \ldots, s^k$ and m (with n the number of bits needed to describe these). □

It is also not difficult to obtain XNLP-completeness for the restriction to a binary alphabet. First, add dummy symbols to Σ to ensure that the size of the alphabet Σ is a power of two. Each transition labeled with a dummy symbol leads to the rejecting state q_R, i.e., we accept the same set of strings. Now, encode each element of Σ with a unique string in $\{0,1\}^{\log|\Sigma|}$. For each state q in the automata, replace the outgoing transitions by a complete binary tree of depth $\log|\Sigma|$ with the left branches labeled by 0 and the right branches labeled by 1. For a leaf of this tree, look at the string $z \in \{0,1\}^{\log|\Sigma|}$ formed by the labels when one follows the path from q to this leaf. This codes a character in Σ; now let this leaf have a transition to the state reached from q when this character is read. This straightforward transformation gives an automaton that precisely accepts the strings when we replace each character by its code in $\{0,1\}^{\log|\Sigma|}$. If we apply the same transformation to all automata, we obtain an equivalent instance but with $\Sigma = \{0,1\}$. We can conclude the following result.

Corollary 29. Acyclic Finite State Automata Intersection *is* XNLP*-complete for automata with a binary alphabet.*

5. Conclusion

We end the paper with some discussions and open problems. We start by discussing a conjecture on the space usage of XNLP-hard problems, then discuss the type of reductions we use, and then give a number of open problems.

5.1. Space efficiency of XNLP-hard problems

Pilipczuk and Wrochna [48] made the following conjecture. In the Longest Common Subsequence problem, we are given k strings $s^1, \ldots, s^k$ over an alphabet Σ and an integer r and ask if there is a string t of length r that is a subsequence of each s^i, $i \in [1, k]$.

Conjecture 30 *(Pilipczuk and Wrochna [48]). The* Longest Common Subsequence *problem has no algorithm that runs in $n^{f(k)}$ time and $f(k)n^c$ space, for a computable function f and constant c, with k the number of strings, and n the total input size.*

Interestingly, this conjecture leads to similar conjectures for a large collection of problems. As Longest Common Subsequence with the number of strings k as parameter is XNLP-complete [32], Conjecture 30 is equivalent to the following conjecture, which is currently known as the *Slice-wise Polynomial Space Conjecture*. Recall that the class XNLP is the same as the class N[f poly, f log].

Conjecture 31. N[f poly, f log] $\not\subseteq$ D[n^f, f poly].

If Conjecture 30 holds, then no XNLP-hard problem has an algorithm that uses XP time and simultaneously "FPT" space (i.e., space bounded by the product of a computable function of the parameter and a polynomial of the input size). Thus, XNLP-hardness proofs yield conjectures about the space usage of XP algorithms, and Conjecture 30 is equivalent to the same conjecture for Bandwidth, List Coloring parameterized by pathwidth, Chained CNF-Satisfiability, etc.

In particular, the conjecture suggests that there is a constant k^* for which any deterministic Turing machine needs $\omega(\log n)$ space in order to solve List Coloring for n-vertex graphs of treewidth k^*; for trees the problem can still be solved in logarithmic space [14], i.e. k^* needs to be chosen strictly greater than one, but it's unclear what space efficiency can be achieved for treewidth 2, for example.[7]

5.2. Reductions

In this paper, we mainly used parameterized logspace reductions (pl-reductions), i.e., parameterized reductions that run in $f(k) + O(\log n)$ space, with f a computable function.

Elberfeld et al. [32] use a stronger form of reductions, namely *parameterized first-order reductions* or pFO-reductions, where the reduction can be computed by a logarithmic time-uniform para AC^0-circuit family. In [32], it is shown that Timed Non-Deterministic Cellular Automaton and Longest Common Subsequence (with the number of strings as parameter) are XNLP-complete under pFO-reductions. We have chosen to use the easier to handle notion of logspace reductions throughout the paper, and not to distinguish which steps can be done with pFO-reductions and which not.

One might want to use the least restricted form of reductions, under which XNLP remains closed, and that are transitive, in order to be able to show hardness for XNLP for as many problems as possible. Instead of using $O(f(k) + \log n)$ space, one may want to use $O(f(k) \cdot \log n)$ space – thus allowing to use a number of counters and pointers that depends on the parameter, instead of being bounded by a fixed constant. However, it is not clear that XNLP is closed under parameterized reductions with a $O(f(k) \cdot \log n)$ space bound, as the reduction may use $O(n^{f(k)})$ time.

To remedy this, we can simultaneously bound the time and space of the reduction. A *parameterized tractable logspace reduction* (ptl-reductions) is a parameterized reduction that simultaneously uses $O(f(k) \cdot \log n)$ space and $O(g(k) \cdot n^c)$ time, with f and g computable functions, k the parameter, and n the input size. One can observe that the same argument ("repeatedly recomputing input bits when needed") that shows transitivity of L-reductions (see [3, Lemma 4.15]) can be used to show transitivity of parameterized tractable logspace reductions (and of parameterized logspace reductions).

We currently are unaware of a problem where we would use ptl-reductions instead of pl-reductions. However, the situation reminds of a phenomenon that also shows up for hardness proofs for classes in the W-hierarchy. Pl-reductions allow us to use time that grows faster than polynomial in the parameter value. If we have an fpt-reduction that uses $O(f(k)n^c)$ time with c a constant, and f *a polynomial function*, then this could be used in an NP-hardness proof for the unparameterized version of the problem. Most *but not all* fpt-reductions from the literature have such a polynomial time bound. However, in the published hardness proofs, the distinction is usually not made explicit.

[7] In a subtle way, the conjecture actually does not rule out that there is an $O(\log n)$ space algorithm for List Colouring for treewidth k graphs, for each k, but with the constant hidden in the O-notation growing faster than any computable function f.

5.3. Relation between XNLP and W[P]

The relation between XNLP and W[P] is currently unknown. We conjecture that these classes are incomparable. At one hand, problems in W[P] have certificates with $O(k \log n)$ bits, while problems in XNLP seem to require certificates with $\Omega(n^c)$ bits, k the parameter, n the input size and $c > 0$. At the other hand, the polynomial size circuits that are central in the definition of W[P] (see e.g. [28]) may express problems that have no logspace algorithms. See also the discussion in [35]. We leave the relation between XNLP and W[P] as an interesting open problem.

5.4. Candidate XNLP-complete problems

In this paper, we showed that the class XNLP captures the complexity of various parameterized problems. The common denominator of such problems appears to be that there is some form of "linear structure" in the problem statement: either we search for some linear structure, or the input is linearly structured. Recent work, following our investigations, revealed that there are more such linear structured problems to be complete for XNLP, e.g. [16]; we expect that there are much more examples. A number of candidates are linear graph ordering problems, like Colored Cutwidth, Feasible Register Allocation (see [13]), Shortest Common Supersequence as mentioned in [28], Restricted Completion to a Proper Interval Graph with Bounded Clique Size, (see [42]), or problems known to be in XP when parameterized by a linear width measure (like pathwidth, linear cliquewidth). Jansen et al. [40] mentioned as open problem XNLP-hardness for two planarity testing problems with treewidth as parameter. Bakkane and Jaffke [4] suggested that the generalized dominating set problems which they prove to be W[1]-hard are likely to be XNLP-hard.

CRediT authorship contribution statement

Hans L. Bodlaender: Conceptualization, Formal analysis, Investigation, Methodology, Supervision, Writing – original draft, Writing – review & editing. **Carla Groenland:** Conceptualization, Formal analysis, Investigation, Methodology, Supervision, Writing – original draft, Writing – review & editing. **Jesper Nederlof:** Conceptualization, Formal analysis, Funding acquisition, Investigation, Methodology, Writing – original draft. **Céline Swennenhuis:** Conceptualization, Formal analysis, Investigation, Methodology, Writing – original draft.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgment

The authors would like to thank the referees for useful comments on an earlier conference version of this paper.

References

[1] A. Abboud, K. Lewi, R. Williams, Losing weight by gaining edges, in: A.S. Schulz, D. Wagner (Eds.), 22th Annual European Symposium on Algorithms, ESA 2014, in: Lecture Notes in Computer Science, vol. 8737, Springer, 2014, pp. 1–12.

[2] S. Arnborg, A. Proskurowski, Linear time algorithms for NP-hard problems restricted to partial k-trees, Discrete Appl. Math. 23 (1) (1989) 11–24, https://doi.org/10.1016/0166-218X(89)90031-0.

[3] S. Arora, B. Barak, Computational Complexity: A Modern Approach, Cambridge University Press, 2007, https://theory.cs.princeton.edu/complexity/book.pdf.

[4] B.I.K. Bakkane, L. Jaffke, On the hardness of generalized domination problems parameterized by mim-width, in: H. Dell, J. Nederlof (Eds.), 17th International Symposium on Parameterized and Exact Computation, IPEC 2022, Dagstuhl, Germany, in: LIPIcs, vol. 249, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, 3.

[5] H.L. Bodlaender, The complexity of finding uniform emulations on paths and ring networks, Inf. Comput. 86 (1) (1990) 87–106, https://doi.org/10.1016/0890-5401(90)90027-F.

[6] H.L. Bodlaender, Parameterized complexity of bandwidth of caterpillars and weighted path emulation, in: L. Kowalik, M. Pilipczuk, P. Rzazewski (Eds.), 47th International Workshop on Graph-Theoretic Concepts in Computer Science, WG 2021, in: Lecture Notes in Computer Science, vol. 12911, Springer, 2021, pp. 15–27.

[7] H.L. Bodlaender, G. Cornelissen, M. van der Wegen, Problems hard for treewidth but easy for stable gonality, in: M.A. Bekos, M. Kaufmann (Eds.), 48th International Workshop on Graph-Theoretic Concepts in Computer Science, WG 2022, in: Lecture Notes in Computer Science, vol. 13453, Springer, 2022, pp. 84–97.

[8] H.L. Bodlaender, M. Cygan, S. Kratsch, J. Nederlof, Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth, Inf. Comput. 243 (2015) 86–111, https://doi.org/10.1016/j.ic.2014.12.008.

[9] H.L. Bodlaender, N. Donselaar, J. Kwisthout, Parameterized completeness results for Bayesian inference, in: A. Salmerón, R. Rumí (Eds.), International Conference on Probabilistic Graphical Models, PGM 2022, in: Proceedings of Machine Learning Research, vol. 186, PMLR, 2022, pp. 145–156, https://proceedings.mlr.press/v186/bodlaender22a.html.

[10] H.L. Bodlaender, R.G. Downey, M.R. Fellows, M.T. Hallett, H.T. Wareham, Parameterized complexity analysis in computational biology, Comput. Appl. Biosci. 11 (1) (1995) 49–57, https://doi.org/10.1093/bioinformatics/11.1.49.

[11] H.L. Bodlaender, M.R. Fellows, $W[2]$-hardness of precedence constrained K-processor scheduling, Oper. Res. Lett. 18 (2) (1995) 93–97, https://doi.org/10.1016/0167-6377(95)00031-9.

[12] H.L. Bodlaender, M.R. Fellows, M. Hallett, Beyond NP-completeness for problems of bounded width: hardness for the W hierarchy, in: 26th Annual ACM Symposium on Theory of Computing, STOC 1994, ACM Press, 1994, pp. 449–458.

[13] H.L. Bodlaender, M.R. Fellows, M.T. Hallett, T. Wareham, T.J. Warnow, The hardness of perfect phylogeny, feasible register assignment and other problems on thin colored graphs, Theor. Comput. Sci. 244 (1–2) (2000) 167–188, https://doi.org/10.1016/S0304-3975(98)00342-9.

[14] H.L. Bodlaender, C. Groenland, H. Jacob, List colouring trees in logarithmic space, in: S. Chechik, G. Navarro, E. Rotenberg, G. Herman (Eds.), 30th Annual European Symposium on Algorithms, ESA 2022, in: LIPIcs, vol. 244, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, 24.

[15] H.L. Bodlaender, C. Groenland, H. Jacob, On the parameterized complexity of computing tree-partitions, in: H. Dell, J. Nederlof (Eds.), 17th International Symposium on Parameterized and Exact Computation, IPEC 2022, in: LIPIcs, vol. 249, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, 7.

[16] H.L. Bodlaender, C. Groenland, H. Jacob, L. Jaffke, P.T. Lima, XNLP-completeness for parameterized problems on graphs with a linear structure, in: H. Dell, J. Nederlof (Eds.), 17th International Symposium on Parameterized and Exact Computation, IPEC 2022, in: LIPIcs, vol. 249, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, 8.

[17] H.L. Bodlaender, C. Groenland, H. Jacob, M. Pilipczuk, M. Pilipczuk, On the complexity of problems on tree-structured graphs, in: H. Dell, J. Nederlof (Eds.), 17th International Symposium on Parameterized and Exact Computation, IPEC 2022, in: LIPIcs, vol. 249, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, 6.

[18] H.L. Bodlaender, C. Groenland, J. Nederlof, C.M.F. Swennenhuis, Parameterized problems complete for nondeterministic FPT time and logarithmic space, in: 62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, 2021, pp. 193–204.

[19] H.L. Bodlaender, C. Groenland, M. Pilipczuk, Parameterized complexity of binary CSP: vertex cover, treedepth, and related parameters, in: K. Etessami, U. Feige, G. Puppis (Eds.), 50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, in: LIPIcs, vol. 261, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023, 27.

[20] H.L. Bodlaender, C. Groenland, C.M.F. Swennenhuis, Parameterized complexities of dominating and independent set reconfiguration, in: P.A. Golovach, M. Zehavi (Eds.), 16th International Symposium on Parameterized and Exact Computation, IPEC 2021, in: LIPIcs, vol. 214, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, 9.

[21] H.L. Bodlaender, I. Mannens, J.J. Oostveen, S. Pandey, E.J. van Leeuwen, The parameterised complexity of integer multicommodity flow, in: N. Misra, M. Wahlström (Eds.), 18th International Symposium on Parameterized and Exact Computation, IPEC 2023, in: LIPIcs, vol. 285, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023, 6.

[22] H.L. Bodlaender, R.H. Möhring, The pathwidth and treewidth of cographs, SIAM J. Discrete Math. 6 (2) (1993) 181–188, https://doi.org/10.1137/0406014.

[23] Y. Chen, J. Flum, M. Grohe, Bounded nondeterminism and alternation in parameterized complexity theory, in: 18th Annual IEEE Conference on Computational Complexity, Complexity 2003, IEEE Computer Society, 2003, pp. 13–29.

[24] M. Cygan, F.V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, S. Saurabh, Parameterized Algorithms, Springer, 2015.

[25] J.M. de Vlas, On the parameterized complexity of the perfect phylogeny problem, in: H. Fernau, S. Gaspers, R. Klasing (Eds.), 49th International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2024, in: Lecture Notes in Computer Science, vol. 14519, Springer, 2024, pp. 169–182.

[26] R.G. Downey, M.R. Fellows, Fixed-parameter tractability and completeness I: basic results, SIAM J. Comput. 24 (4) (1995) 873–921, https://doi.org/10.1137/S0097539792228228.

[27] R.G. Downey, M.R. Fellows, Fixed-parameter tractability and completeness II: on completeness for W[1], Theor. Comput. Sci. 141 (1&2) (1995) 109–131, https://doi.org/10.1016/0304-3975(94)00097-3.

[28] R.G. Downey, M.R. Fellows, Parameterized Complexity, Monographs in Computer Science, Springer, 1999.

[29] R.G. Downey, M.R. Fellows, Fundamentals of Parameterized Complexity, Texts in Computer Science, Springer, 2013.

[30] M.S. Dregi, D. Lokshtanov, Parameterized complexity of bandwidth on trees, in: J. Esparza, P. Fraigniaud, T. Husfeldt, E. Koutsoupias (Eds.), 41st International Colloquium on Automata, Languages, and Programming, ICALP 2014, in: Lecture Notes in Computer Science, vol. 8572, Springer, 2014, pp. 405–416.

[31] M. Elberfeld, A. Jakoby, T. Tantau, Logspace versions of the theorems of Bodlaender and Courcelle, in: 51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, IEEE Computer Society, 2010, pp. 143–152.

[32] M. Elberfeld, C. Stockhusen, T. Tantau, On the space and circuit complexity of parameterized problems: classes and completeness, Algorithmica 71 (3) (2015) 661–701, https://doi.org/10.1007/s00453-014-9944-y.

[33] M.R. Fellows, F.V. Fomin, D. Lokshtanov, F.A. Rosamond, S. Saurabh, S. Szeider, C. Thomassen, On the complexity of some colorful problems parameterized by treewidth, Inf. Comput. 209 (2) (2011) 143–153, https://doi.org/10.1016/j.ic.2010.11.026.

[34] M.R. Fellows, D. Hermelin, F.A. Rosamond, S. Vialette, On the parameterized complexity of multiple-interval graph problems, Theor. Comput. Sci. 410 (1) (2009) 53–61, https://doi.org/10.1016/j.tcs.2008.09.065.

[35] M.R. Fellows, F.A. Rosamond, Collaborating with Hans: some remaining wonderments, in: F.V. Fomin, S. Kratsch, E.J. van Leeuwen (Eds.), Treewidth, Kernels, and Algorithms – Essays Dedicated to Hans L. Bodlaender on the Occasion of His 60th Birthday, in: Lecture Notes in Computer Science, vol. 12160, Springer, 2020, pp. 7–17.

[36] J.P. Fishburn, R.A. Finkel, Quotient networks, IEEE Trans. Comput. C-31 (1982) 288–295, https://doi.org/10.1109/TC.1982.1675994.

[37] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman, 1979.

[38] E.M. Gurari, I.H. Sudborough, Improved dynamic programming algorithms for bandwidth minimization and the MinCut linear arrangement problem, J. Algorithms 5 (1984) 531–546, https://doi.org/10.1016/0196-6774(84)90006-3.

[39] L. Jaffke, P.T. Lima, R. Sharma, Structural parameterizations of b-coloring, in: S. Iwata, N. Kakimura (Eds.), 34th International Symposium on Algorithms and Computation, ISAAC 2023, in: LIPIcs, vol. 283, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023, 40.

[40] B.M.P. Jansen, L. Khazaliya, P. Kindermann, G. Liotta, F. Montecchiani, K. Simonov, Upward and orthogonal planarity are w[1]-hard parameterized by treewidth, in: M.A. Bekos, M. Chimani (Eds.), Graph Drawing and Network Visualization, Springer Nature Switzerland, Cham, 2023, pp. 203–217.

[41] K. Jansen, P. Scheffler, Generalized coloring for tree-like graphs, Discrete Appl. Math. 75 (2) (1997) 135–155, https://doi.org/10.1016/S0166-218X(96)00085-6.

[42] H. Kaplan, R. Shamir, Pathwidth, bandwidth, and completion problems to proper interval graphs with small cliques, SIAM J. Comput. 25 (3) (1996) 540–561, https://doi.org/10.1137/S0097539793258143.

[43] S. Kintali, S. Munteanu, Computing bounded path decompositions in logspace, Electron. Colloq. Comput. Complex. 126 (2012), https://eccc.weizmann.ac.il/report/2012/126.

[44] D. Lokshtanov, D. Marx, S. Saurabh, Known algorithms on graphs of bounded treewidth are probably optimal, ACM Trans. Algorithms 14 (2) (2018) 13, https://doi.org/10.1145/3170442.

[45] B. Monien, The bandwidth minimization problem for caterpillars with hair length 3 is NP-complete, SIAM J. Algebraic Discrete Methods 7 (1986) 505–512, https://doi.org/10.1137/0607057.
[46] C.H. Papadimitriou, The NP-completeness of the bandwidth minimization problem, Computing 16 (1976) 263–270, https://doi.org/10.1007/BF02280884.
[47] K. Pietrzak, On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems, J. Comput. Syst. Sci. 67 (4) (2003) 757–771, https://doi.org/10.1016/S0022-0000(03)00078-3.
[48] M. Pilipczuk, M. Wrochna, On space efficiency of algorithms working on structural decompositions of graphs, ACM Trans. Comput. Theory 9 (4) (2018) 18, https://doi.org/10.1145/3154856.
[49] D. Prot, O. Bellenguez-Morineau, A survey on how the structure of precedence constraints may change the complexity class of scheduling problems, J. Sched. 21 (1) (2018) 3–16, https://doi.org/10.1007/s10951-017-0519-z.
[50] J.B. Saxe, Dynamic programming algorithms for recognizing small-bandwidth graphs in polynomial time, SIAM J. Algebraic Discrete Methods 1 (1980) 363–369, https://doi.org/10.1137/0601042.
[51] L.J. Stockmeyer, The polynomial-time hierarchy, Theor. Comput. Sci. 3 (1) (1976) 1–22, https://doi.org/10.1016/0304-3975(76)90061-X.
[52] J.A. Telle, A. Proskurowski, Algorithms for vertex partitioning problems on partial k-trees, SIAM J. Discrete Math. 10 (4) (1997) 529–550, https://doi.org/10.1137/S0895480194275825.
[53] M. Wehar, On the Complexity of Intersection Non-Emptiness Problems, PhD thesis, University at Buffalo, State University of New York, 2016.