Parameterized Complexity of Binary CSP: Vertex Cover, Treedepth, and Related Parameters

Hans L. Bodlaender
Department of Information and Computing Sciences, Utrecht University, The Netherlands

Carla Groenland
Mathematical Institute, Utrecht University, The Netherlands

Michał Pilipczuk
Institute of Informatics, University of Warsaw, Poland

Abstract

We investigate the parameterized complexity of Binary CSP parameterized by the vertex cover number and the treedepth of the constraint graph, as well as by a selection of related modulator-based parameters. The main findings are as follows:

- Binary CSP parameterized by the vertex cover number is W[3]-complete. More generally, for every positive integer d, Binary CSP parameterized by the size of a modulator to a treedepth-d graph is W[$2d+1$]-complete. This provides a new family of natural problems that are complete for odd levels of the W-hierarchy.
- We introduce a new complexity class XSLP, defined so that Binary CSP parameterized by treedepth is complete for this class. We provide two equivalent characterizations of XSLP: the first one relates XSLP to a model of an alternating Turing machine with certain restrictions on conondeterminism and space complexity, while the second one links XSLP to the problem of model-checking first-order logic with suitably restricted universal quantification. Interestingly, the proof of the machine characterization of XSLP uses the concept of *universal trees*, which are prominently featured in the recent work on parity games.
- We describe a new complexity hierarchy sandwiched between the W-hierarchy and the A-hierarchy: For every odd t, we introduce a parameterized complexity class S[t] with W[t] $\subseteq$ S[t] $\subseteq$ A[t], defined using a parameter that interpolates between the vertex cover number and the treedepth.

We expect that many of the studied classes will be useful in the future for pinpointing the complexity of various structural parameterizations of graph problems.

2012 ACM Subject Classification Theory of computation → W hierarchy

Keywords and phrases Parameterized Complexity, Constraint Satisfaction Problems, Binary CSP, List Coloring, Vertex Cover, Treedepth, W-hierarchy

Digital Object Identifier 10.4230/LIPIcs.ICALP.2023.27

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://doi.org/10.48550/arXiv.2208.12543

Funding *Carla Groenland*: Supported by the Marie Skłodowska-Curie grant GRAPHCOSY (number 101063180).
Michał Pilipczuk: This research is a part of the project BOBR that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement no. 948057).

1 Introduction

The Binary Constraint Satisfaction Problem (BinCSP, for short) is a fundamental problem defined as follows. We are given an undirected graph $G = (V, E)$, called the *primal* or the *Gaifman graph*, where V is a set of variables, each with a prescribed domain of possible

values. Further, each edge uv of G corresponds to a binary constraint that restricts the possible pairs of values that can be assigned to u and v. The task is to decide whether every variable can be mapped to a value from its domain so that all the constraints are satisfied.

Due to immense modeling power, constraint satisfaction problems are of great importance in multiple applications, and the theoretical study of their complexity is a field on its own. In this work we are interested in parameterized algorithms for BinCSP, with a particular focus on structural parameters of the Gaifman graph. An example of such a result is a classic observation, usually attributed to Freuder [32]: using dynamic programming, BinCSP can be solved in time $n^{k+\mathcal{O}(1)}$, where n is the maximum size of a domain and k is the treewidth of the Gaifman graph. In the language of parameterized complexity, this means that BinCSP parameterized by treewidth is *slice-wise polynomial*, or in the complexity class XP.

The class XP is very general and just placing BinCSP parameterized by treewidth within XP does not provide much insight into the actual complexity of the problem. A more detailed study of the parameterizations of BinCSP by pathwidth and by treewidth was recently performed by Bodlaender, Groenland, Nederlof, and Swennenhuis in [11], and by Bodlaender, Groenland, Jacob, Pilipczuk, and Pilipczuk in [10]. In particular, as shown in [11], BinCSP parameterized by pathwidth is complete for XNLP: the class of all parameterized problems that can be solved by a nondeterministic Turing machine using $f(k)\log n$ space and $f(k)\cdot n^{\mathcal{O}(1)}$ time, where k is the parameter and f is a computable function. A "tree variant" of XNLP, called XALP, was studied in [10]; it can be defined using the same model of a Turing machine, except that the machine additionally has access to a stack of unbounded size that can be manipulated by pushing and popping. As proved in [10], BinCSP parameterized by treewidth is complete for XALP. All in all, the recent works [7, 9, 10, 11, 26] present a variety of problems on graphs with linear or tree-like structure that are complete for XNLP and XALP, respectively. This is an evidence that XNLP and XALP capture certain fundamental varieties of computational problems: those amenable to linearly and tree-structured dynamic programming with state space of slice-wise polynomial size.

The contemporary research in parameterized algorithms features many more structural parameters of graphs, besides treewidth and pathwidth. In this work we explore the complexity of BinCSP parameterized by the following parameters of the Gaifman graph: (1) the vertex cover number, (2) the treedepth, and (3) a selection of related modulator-based parameters lying between the vertex cover number and the treedepth.

New completeness results for the W-hierarchy. The W-hierarchy was introduced around thirty years ago in the work by Downey and Fellows that founded the field of parameterized algorithms and complexity. In this hierarchy, we have a collection of classes, including $\mathrm{W}[1] \subseteq \mathrm{W}[2] \subseteq \ldots \subseteq \mathrm{W}[\mathrm{SAT}] \subseteq \mathrm{W}[\mathrm{P}]$; see [20, 21, 30] for an overview and for bibliographic references. A large variety of problems are known to be complete (under fpt reductions) for W[1] and for W[2]. However, for classes W[t] with $t \geqslant 3$, there is so far only a handful of examples of natural problems known to be complete [1, 5, 6, 14, 15, 36]. Our first contribution is to give new examples of complete problems for W[t] for all odd $t \geqslant 3$.

Our first example concerns BinCSP parameterized by the *vertex cover number*: the minimum size of a vertex cover in the Gaifman graph.

▶ **Theorem 1.** *BinCSP parameterized by the vertex cover number of the Gaifman graph is complete for the class* W[3].

It was known that BinCSP parameterized by the vertex cover number is W[1]-hard [29, 44]. The W[3]-completeness is surprising, not only due to the small number of examples of natural W[3]-complete problems, but also because many problems appear to be fixed-parameter tractable or even have a kernel of polynomial size, when the vertex cover number is used as the parameter (e.g., [28, 29, 31, 34]).

For a graph G and a graph class $\mathcal{C}$, a *modulator* to $\mathcal{C}$ in G is a set of vertices W such that $G - W \in \mathcal{C}$. For instance, vertex covers are modulators to the class of edgeless graphs. A *feedback vertex set* is another type of a modulator, now to graphs without cycles, i.e., to forests. The *feedback vertex number* of a graph G is the minimum size of a feedback vertex set in G. We prove that the parameterization by the feedback vertex number yields a much harder problem.

▶ **Theorem 2.** *BinCSP parameterized by the feedback vertex number of the Gaifman graph is* W[SAT]*-hard and in* W[P]*.*

Finally, with similar techniques, we obtain the following completeness results for W[t] for all odd $t \geqslant 3$. Here, *treedepth* is a structural parameter measuring the "depth" of a graph, we will expand on it later on.

▶ **Theorem 3.** *For each integer $d \geqslant 1$, BinCSP is complete for* W[$2d+1$] *when parameterized by the minimum size of a modulator to a graph of treedepth at most d, and when parameterized by the minimum size of a modulator to a forest of depth at most d.*

Interestingly, each increase of the depth of the trees by one corresponds to an increase in the W-hierarchy by two levels: this is because one level of depth in the tree or forest corresponds to a conjunction (looking at all children of a node) with a disjunction (the choice of a value). Theorem 3 can be seen as an interpolation between Theorems 1 and 2: by allowing the forest to have larger and larger depth, we obtain harder and harder parameterized problems. This yields a family of natural complete problems for the odd levels of the W-hierarchy.

Theorem 1 is proved in Section 3, while Theorems 2 and 3 are proved in the full version [12].

Treedepth parameterization: class XSLP. As we argued, the classes XNLP and XALP can be seen as the "natural home" for BinCSP parameterized by pathwidth and treewidth respectively, and for many other problems on "path-like" or "tree-like" graphs. We introduce a new parameterized complexity class XSLP which is the "natural home" for the parameter treedepth instead, reflecting "shallow" graphs (this is what the letter S stands for).

The *treedepth* of a graph G is the minimum depth of a rooted forest F on the same vertex set as G such that every edge of G connects a vertex with its ancestor in F; thus, it is a measure of shallowness of a graph. While treedepth is never smaller than pathwidth, it can be arbitrarily large even on graphs of bounded pathwidth: a path on n vertices has pathwidth 1 and treedepth $\lceil \log_2(n+1) \rceil$. Despite being relatively lesser known than treewidth or pathwidth, treedepth appears naturally in many seemingly disconnected areas. For instance, it features a prominent role in the theory of Sparsity (see [43, Chapters 6 and 7] for an overview), has interesting combinatorics of its own (see e.g. [16, 19, 22, 39]), corresponds to important dividing lines in finite model theory (see e.g. [25, 40]), and governs the parameterized complexity of block-structured integer programming (see [24] for an overview). More importantly for us, a line of work [33, 37, 41, 42, 45, 46] uncovered that for many classic problems, on graphs of low treedepth one can design fixed-parameter algorithms that are both time- and space-efficient, which is conjectured not to be possible for the pathwidth or treewidth parameterizations [46]. This makes treedepth a prime candidate for a parameter that can be interesting from the point of view of BinCSP.

And so, we define two complexity classes: XSLP consists of all parameterized problems that can be reduced to BinCSP parameterized by treedepth in parameterized logspace (that is, in deterministic space $f(k) + \mathcal{O}(\log n)$ for a computable f), while XSLP$^+$ has the same definition, except we consider fpt reductions. This distinction is of technical nature: on one

hand we use parameterized logspace reductions to match the definitions of XALP and XNLP and retain the inclusion XSLP $\subseteq$ XNLP $\subseteq$ XALP, and on the other hand we would like to compare XSLP with the W-hierarchy, which requires closure under fpt reductions. In fact, XSLP$^+ \supseteq$ W[t] for every integer t (this will follows from Proposition 4).

We prove two alternative characterizations of XSLP. The first one is through a machine model: we prove that XSLP can be equivalently defined as problems that can be solved by an alternating Turing machine with the following resource bounds: (1) $f(k)\log n$ bits of nondeterminism, (2) $f(k) + \mathcal{O}(\log n)$ bits of conondeterminism, (3) alternation at most $f(k)$, and (4) working space $f(k) + \mathcal{O}(\log n)$ plus a read-once stack of size $f(k)\log n$ that can be only pushed upon and read only at the end of the computation. See Theorem 10 in Section 4.1 for a formal statement. This reflects the characterization of XALP through alternating Turing machines with different bounds on conondeterminism and the size of a computation tree, see [10, Theorem 1].

The main step in the proof of our machine characterization of XSLP is a regularization lemma for the considered machine model, allowing us to assume that the computation tree has always a very concrete shape. Interestingly, this step crucially uses the existence of fpt-sized *universal trees*, a tool fundamentally underlying the recent advances in the complexity of parity games. While universal trees can be seen only implicitly in the breakthrough work of Calude et al. [13], their central role in the approach was exposed in subsequent works [18, 38].

The second characterization is through model-checking first-order logic, and is inspired by the definition of the A-hierarchy; see [30, Chapter 8]. In essence, we provide a complete problem for XSLP, which amounts to model-checking first-order sentences in which universal quantification must follow a root-to-leaf path in a rooted forest present in the structure. Details and formal statements can be found in Section 4.2.

***d*-fold vertex cover and the S-hierarchy.** We "project" the class XSLP closer to lower levels of the W-hierarchy, thus obtaining a new hierarchy of parameterized classes sandwiched between the W-hierarchy and the A-hierarchy. For this, we introduce the following parameter.

The 1-*fold vertex cover number* of a graph G is simply the number of vertices of G. Inductively, for $d \geqslant 2$, the *d-fold vertex cover number* is the smallest integer k with the following property: there is a subset of vertices $U \subseteq V(G)$ with $|U| \leqslant k$ such that every connected component of $G - U$ has $(d-1)$-fold vertex cover number at most k. Alternatively, we can also define the parameter using a "fattened" variant of elimination trees (the decomposition notion underlying treedepth). Namely, G has d-fold vertex cover number at most k if and only if there is a rooted tree T of depth at most d, and a vertex partition $\{V_t \colon t \in V(T)\}$ of $V(G)$ such that $|V_t| \leqslant k$ for all $t \in V(T)$, and edges in G between vertices of V_s and V_t are only allowed when s and t are equal or are in an ancestor-descendant relationship in T.

We now define the parameterized complexity class[1] S[$2d-1$] as the fpt-closure of BinCSP parameterized by the d-fold vertex cover number, for all integers $d \geqslant 1$. The following result relates the introduced classes to the W-hierarchy, the A-hierarchy, and the class XSLP$^+$.

▶ **Proposition 4.** *For every integer $d \geqslant 1$, we have* W[$2d-1$] $\subseteq$ S[$2d-1$] $\subseteq$ A[$2d-1$] *and* S[$2d-1$] $\subseteq$ XSLP$^+$.

The proof is straightforward and is given in the full version [12].

[1] We remark that there is an already existing concept called the S-hierarchy, related to subexponential parameterized algorithms; see [30, Definition 16.9]. Since we are not aware of any subsequent work on the structure of this hierarchy, we took the liberty of using the same naming scheme for our classes.

While the definition of d-fold vertex cover seems not to have been discussed explicitly in the literature, the idea of alternating deletions of batches of vertices and splitting into connected components is not entirely new, as similar parameters that interpolate between vertex cover and treedepth have previously been studied. For example, 2-fold vertex cover is within a multiplicative factor of two of *vertex integrity*, a parameter that was introduced by Barefoot, Entringer and Swart [4] in 1987 (see [2] for a survey). In the context of block-structure integer programs, the *fracture number* [23] can be seen as an analogue of 2-fold vertex cover, while the concept of *topological height* [24] serves a role similar to that of d in the definition of d-fold vertex cover.

Comparison to List Coloring. The classic List Coloring problem can be interpreted as the special case of BinCSP where every constraint just stipulates that the values assigned to adjacent variables are different from each other. Therefore, a hardness result for List Coloring implies one for BinCSP. Vice versa, we can attempt to turn an instance of BinCSP on graph G into an instance of List Coloring by adding, for each edge uv in G and each forbidden pair of values (a, b), a vertex to G adjacent to u and v with color list $\{a, b\}$. This transformation does not significantly affect graph parameters such as treedepth, treewidth or pathwidth, so hardness and completeness results of BinCSP may also be inherited to List Coloring. However, the transformation may make dramatic changes to other parameters such as vertex cover and vertex modulator to a graph of treedepth at most d, where we can only easily deduce W[$2d-1$]-hardness from our W[$2d+1$]-hardness results. In fact, we separate the two problems with the following result, proved in the full version [12].

▶ **Theorem 5.** *List Coloring is in* W[2] *when parameterized by the vertex cover number and in* W[$2d$] *when parameterized by the size of a modulator to a treedepth-d graph.*

We believe that due to its robustness, BinCSP better suited to measure the complexity of parameters than List Coloring is. This is also witnessed by the (nearly) tight completeness results presented in Theorems 1, 2, and 3. Table 1 below presents a comparison of the parameterized complexity landscapes of BinCSP and of List Coloring under various structural parameterizations. We discuss this table in the full version [12].

2 Preliminaries

For integers $a \leqslant b$, we write $[a, b]$ for $\{a, a+1, \ldots, b\}$.

Graphs and their parameters. In this paper, we denote the *depth* of a rooted tree as the maximum number of vertices on a path from root to leaf. A *rooted forest* is a collection of rooted trees. The *depth* of a rooted forest is the maximum depth of the trees in the forest.[2]

We use standard graph notation. An *elimination forest* of a graph G, is a rooted forest F with the same vertex set as G, such that for each edge uv of G, u is an ancestor of v or v is an ancestor of u in F. (Note that the forest can contain edges that are not in G.) The *treedepth* of a graph G is the minimum depth of of rooted forest embedding of G.

Let $\mathcal{C}$ be a class of graphs. A *modulator* to $\mathcal{C}$ in a graph G is a set of vertices $W \subseteq V(G)$, such that the graph $G - W$ belongs to $\mathcal{C}$. A *vertex cover* of a graph G is a set of vertices $W \subseteq V(G)$, such that every edge of G has at least one endpoint in W. Note that a set of

[2] The definitions of depth of a tree used in the literature can differ by one. Here we count the number of vertices, e.g., a tree consisting of a single vertex has depth 1.

Table 1 Complexity of BinCSP and List Coloring. Results marked with $*$ are shown in this paper. Some results without a reference are easy to obtain.

Parameter	Binary CSP	List Coloring
number of vertices	W[1]-complete [27, 44]	poly-kernel
vertex cover	W[3]-complete $*$	W[1]-hard [29], in W[2] $*$
feedback vertex set	W[SAT]-hard, in W[P] $*$	W[3]-hard, in W[P] $*$
modulator to treedepth-d	W[$2d+1$]-complete $*$	W[$2d-1$]-hard, in W[$2d$] $*$
modulator to depth d-forest	W[$2d+1$]-complete $*$	W[$2d-1$]-hard, in W[$2d$] $*$
modulator to clique	para-NP-complete	FPT, poly-kernel [3, 35]
treedepth	XSLP-complete $*$	XSLP-complete $*$
tree partition width	XALP-complete [10]	W[1]-hard, in XL [8]
tree partition width + degree	XALP-complete [10]	FPT
pathwidth	XNLP-complete [11]	XNLP-complete [11]
bandwidth	XNLP-complete [11]	FPT
treewidth	XALP-complete [10]	XALP-complete [10]
treewidth + degree	XALP-complete [10]	FPT

vertices is a vertex cover if and only if it is a modulator to the class of edgeless graphs, or, equivalently, to the class of graphs with treedepth at most 1. A *feedback vertex set* in a graph G is a modulator to a forest, or, equivalently, a set of vertices that intersects each cycle in G.

Constraint satisfaction problems. We consider the BinCSP problem defined as follows. An instance of BinCSP is a triple

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

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 $uv \in E(G)$, $C(u,v) \subseteq D(u) \times D(v)$ is a binary relation called the *constraint* at uv. Note that $C(u,v)$ is not necessarily symmetric; throughout this paper, we apply the convention that $C(v,u) = \{(b,a) \mid (a,b) \in C(u,v)\}$.

In the context of a BinCSP instance, we may sometimes call vertices *variables*. 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 uv of G, we have $(\eta(u), \eta(v)) \in C(u,v)$. The BinCSP problem asks, for a given instance I, whether I is *satisfiable*, that is, there is a satisfying assignment for I.

The List Coloring problem is a special case of BinCSP defined as follows. An instance consists of a graph G and, for every vertex u of G, a set (*list*) of colors $L(u)$. The question is whether there is a mapping f of vertices to colors such that for every vertex u we have $f(u) \in L(u)$, and for each edge uv of G, we have $f(u) \neq f(v)$. Note that this is equivalent to a BinCSP instance where lists $L(u)$ are the domains, and all constraints are non-equalities: $C(u,v) = \{(a,b) \in L(u) \times L(v) \mid a \neq b\}$ for every edge uv.

Complexity theory. We assume the reader to be familiar with standard notions of the parameterized complexity theory, such as the W-hierarchy or parameterized reductions. For more background, see [17, 20, 21, 30]. Let us recall concepts directly used in this paper.

We say that a parameterized problem Q is in *parameterized logspace* if Q can be decided in (deterministic) space $f(k) + \mathcal{O}(\log n)$, for some computable function f. Note that every problem in parameterized logspace is fixed-parameter tractable, because a Turing machine working in space $f(k) + \mathcal{O}(\log n)$ has $2^{\mathcal{O}(f(k))} \cdot n^{\mathcal{O}(1)}$ configurations, and hence its acceptance can be decided in fixed-parameter time.

An *fpt-reduction* is a parameterized reduction that works in fixed-parameter time. A *pl-reduction* is a parameterized reduction that works in parameterized logspace, that is, can be computed in (deterministic) space $f(k) + \mathcal{O}(\log n)$, for some computable function f.

A Boolean formula is said to be *t-normalized* when it is the conjunction of disjunctions of conjunctions of ... of literals, with t levels of conjunctions or disjunctions. We only consider the case where $t \geqslant 2$, and assume that we start by conjunctions. Note that 2-normalized Boolean formulas are in Conjunctive Normal Form.

In the Weighted t-Normalized Satisfiability problem, we are given a t-normalized Boolean formula F on n variables, and an integer k, and ask if we can satisfy F by setting exactly k of the variables to true, and all other variables to false. This problem is complete for W[t], see e.g. [20, 21]. A t-normalized expression is said to be *anti-monotone* if each literal is the negation of a variable. We use the following result to simplify our proofs.

▶ **Theorem 6** (Downey and Fellows, see [20, 21]). *For every odd $t \geqslant 3$, Weighted Anti-Monotone t-Normalized Satisfiability is complete for W[t].*

We use the following result as starting point for membership proofs.

▶ **Theorem 7** (Downey and Fellows, see [20, 21]). *For every $t \geqslant 2$, Weighted t-Normalized Satisfiability is complete for* W[t].

3 W[3]-completeness for BinCSP parameterized by vertex cover

In this section, we prove Theorem 1. We prove the hardness below and refer to the full version [12] for the proof of membership.

▶ **Lemma 8.** *BinCSP with vertex cover as parameter is* W[3]*-hard.*

Proof. Take an instance of Weighted 3-Normalized Anti-Monotone Satisfiability, i.e., we have a Boolean formula F that is a conjunction of disjunctions of conjunctions of negative literals, and ask if we can satisfy it by setting exactly k variables to true. Suppose $x_1, \ldots, x_n$ are the variables used by F. Suppose F is the conjunction of r disjunctions of conjunctions of negative literals.

We build a graph G as follows. The vertex set $V(G)$ consists of a set $W = \{w_1, \ldots, w_k\}$ of size k, and a set $S = \{v_1, v_2, \ldots, v_r\}$ of size r. The set W will be the vertex cover of G, and S will form an independent set. We add edges from each vertex in W to each other vertex in the graph.

The domain of a vertex $w \in W$ is $D(w) = \{x_1, \ldots, x_n\}$. For distinct $w, w' \in W$, $w' \neq w'$, we set $C(w, w') = \{(x_i, x_j) \mid i \neq j\}$. This enforces that all vertices in W are assigned a different value – this corresponds to setting exactly k variables to true.

Now consider a vertex $v_i \in S$ for $i \in [1, r]$. We say that v_i represents the ith disjunction of conjunctions of literals in F, i.e., each of the disjunctions in the formula is represented by one vertex in the independent set. Suppose that this disjunction has t_i terms (each term is a conjunction of negative literals). We set $D(v_i) = [1, t_i]$, that is, each value for v_i is an integer in $[1, t_i]$.

The intuition is as follows. We set a variable x_i to true, if and only if exactly one vertex in W is assigned x_i. As all vertices in W will get a different value, we set in this way exactly k variables to true. The formula F is the conjunction of r disjunctions; each of these disjunctions is represented by one of the vertices $v_i \in S$. For each v_i, the disjunction represented by v_i must be satisfied, so one of its terms must be satisfied. The value of v_i tells a satisfied term, i.e., if the value of v_i is $j \in [1, t_i]$, then the jth term is satisfied. This is checked by looking at the edges from v_i to the vertices in W.

We now give the constraints that ensure the term is satisfied. Consider a vertex $v_i \in S$ and $w \in W$. Recall that the value of v_i is an integer in $[1, t_i]$ which represents one term in the ith disjunction of F, and that term is a conjunction of a number of negative literals. For $j \in [1, t_i]$ and $j' \in [1, n]$, we have $(j, x_{j'}) \in C(v_i, w)$ if and only if for each literal $\neg x_{j''}$ that appears in the jth term of the ith disjunction of F, $j'' \neq j'$.

We call the constructed graph G and write I for the corresponding instance of BinCSP.

▷ Claim 9. F can be satisfied by setting exactly k variables to true, if and only if I has a satisfying assignment.

Proof of Claim 9. Suppose F can be satisfied by making $x_{i_1}, \ldots, x_{i_k}$ true, and all other literals false. Then assign the vertices in W the values $x_{i_1}, \ldots, x_{i_k}$ successively. The constraints between vertices in W are thus satisfied.

Now consider a vertex $v_i \in S$. Consider the ith term F_i of the (upper level) conjunction of F. This term must be satisfied by the truth assignment. Suppose the term is $F_i = F_{i,1} \vee \cdots \vee F_{i,t_i}$. At least one of the F_{i_j}'s must be satisfied by the truth assignment, say $F_{i,j'}$. Then assign v_i the value j'.

We can verify that the constraints for edges between v_i and each w_j are fulfilled. By assumption, $F_{i,j'}$ holds. It thus cannot contain a negative literal $\neg x_\alpha$, where x_α is set to true. So w_j cannot be assigned x_α when $\neg x_\alpha$ is a literal in $F_{i,j'}$. Thus we found a satisfying assignment for I.

Now, suppose that I has a satisfying assignment. From the constraints between vertices in W, we see that all vertices in W have a different value. Set a variable x_i to true, if and only if a vertex in W has value x_i, and otherwise, set it to false. We have thus set exactly k variables to true.

Consider the ith term of the upper level conjunction of F. Suppose this term is $F_{i,1} \vee \ldots \vee F_{i,t_i}$. Suppose v_i is assigned value j. For each negative literal $\neg x_\alpha$ in the conjunction $F_{i,j}$, by the constraints, we cannot have a vertex in W that is assigned x_α, and thus x_α is set to false. Thus, the term $F_{i,j}$ is satisfied by the truth assignment, and thus F_i is satisfied. As this holds for all conjuncts of F, F is satisfied by the specified assignment. ◁

From Claim 9, we see that we have a parameterized reduction from Weighted Anti-Monotone 3-Normalized Satisfiability to BinCSP with vertex cover as parameter. The result now follows from the W[3]-hardness of Weighted Anti-Monotone 3-Normalized Satisfiability (Theorem 6). ◀

4 XSLP and treedepth

In this section we discuss the class XSLP and its various characterizations. As discussed in Section 1, we actually define two variants of this class, depending on the kind of reductions that we would like to speak about. Let BinCSP/td denote the following parameterized problem. We are given a BinCSP instance I and an elimination forest of the Gaifman graph of I

of depth at most k, which is the parameter. The task is to decide whether I is satisfiable. Then the two variants of XSLP are defined as the closures of this problem under pl- and fpt-reductions, respectively:

$$\text{XSLP} = [\text{BinCSP}/\text{td}]^{\text{pl}} \qquad \text{and} \qquad \text{XSLP}^+ = [\text{BinCSP}/\text{td}]^{\text{fpt}}.$$

That is, XSLP consists of all parameterized problems that are pl-reducible to BinCSP/td, and XSLP$^+$ is defined similarly, but with fpt-reductions in mind.

Note that in the BinCSP/td problem we assume that a suitable elimination forest is provided on input. This is to abstract away the need of computing such an elimination forest; the complexity of this task is also an interesting question, but lies beyond the scope of this work.

4.1 A machine characterization

We first give a machine characterization of XSLP. We will use a model of an *alternating read-once stack machine*, or *AROSM* for brevity, which we now define. We assume familiarity with standard Turing machines, on which we build our model.

An alternating read-once stack machine M is a Turing machine that has access to three types of memory, each using $\{0, 1\}$ as the alphabet:

- a read-only input tape;
- a working tape; and
- a read-once stack.

The input tape and the working tape are accessed and manipulated as usual, by a head that may move, read, and (in the case of the working tape) write on the tape. The input to the machine is provided on the input tape. On the other hand, the stack is initially empty and the machine may, upon any transition, push a single symbol onto the stack. It cannot, however, read the stack until the final step of the computation. More precisely, the acceptance condition is as follows: The machine has a specified *final state*. Once it is reached, the computation finishes and the machine reads the ith bit of the stack, where i is the number whose binary encoding is the current content of the working tape. If this bit is 1, then M accepts, and otherwise it rejects.

A *configuration* of M is a 5-tuple consisting of the state, the content of the working tape, the content of the stack, and the positions of the heads on the input and the working tape.

Further, M is an *alternating machine*, which means that its states are partitioned into three types: *existential states*, *universal states*, and *deterministic states*. A configuration of a machine is existential/universal/deterministic if its state is so. When the state of the machine is deterministic, there is exactly one transition allowed. At existential and universal states, there are always two transitions allowed; these will be named the 0-*transition* and the 1-*transition*. The acceptance is defined as usual in alternating machines: when in an existential state, M may accept if at least one allowed transition leads to a configuration from which it may accept, and in a universal state we require that both transitions lead to configurations from which M may accept. The notion of a machine deciding a (parameterized) problem is as usual.

The $\forall$ *computation tree* of M for input x is defined as a tree of configurations with the following properties:

- the root is the initial configuration with input x;
- the leaves are configurations with the final state;
- every deterministic and every existential configuration has exactly one child, which is the unique, respectively any of the two configurations to which the machine may transit;

- every universal configuration has exactly two children, corresponding to the two configurations to which the machine may transit.

It follows that M accepts input x if there is a $\forall$ computation tree for input x where every leaf is a configuration in which M accepts. We call such $\forall$ computation trees *accepting*.

A *branch* of a (rooted) tree is a root-to-leaf path. For a $\forall$ computation tree T of machine M, we define the following quantities:

- The *working space* of T is the minimum number i such among configurations present in T, the head on the working tape is never beyond the ith cell.
- The *stack size* of T is the maximum size of the stack among all configurations in T.
- The *nondeterminism* of T is the maximum number of existential configurations on any branch of T.
- The *conondeterminism* of T is the maximum number of universal configurations on any branch of T.
- The *alternation* of a branch of T is the minimum number of blocks into which the branch can be partitioned so that each of the blocks does not simultaneously contain an existential and a universal configuration. The *alternation* of T is the maximum alternation on any branch of T.

We say that a machine M *decides* a parameterized problem Q using certain resources among those described above, if for any input (x, k), we have $(x, k) \in Q$ if and only if there is an accepting $\forall$ computation tree for (x, k) that has the resources bounded as prescribed.

Having all the necessary definitions in place, we can state the main result of this section.

▶ **Theorem 10.** *The following conditions are equivalent for a parameterized problem Q.*

(1) $Q \in \mathrm{XSLP}$*;*

(2) *Q can be decided by an alternating read-once stack machine that for input (x, k) with $|x| = n$, uses working space at most $f(k)+\mathcal{O}(\log n)$, stack size $f(k)\log n$, nondeterminism $f(k)\log n$, co-nondeterminism $f(k)+\mathcal{O}(\log n)$, and alternation $f(k)$, for some computable function f.*

Before we proceed to the proof of Theorem 10, let us discuss the necessity of different resource restrictions described in (2):

- Increasing the working space to $f(k)\log n$ (and thus rendering the stack, the nondeterminism and the co-nondeterminism unnecessary) would make the machine model at least as powerful (and in fact, equivalently powerful) as deterministic Turing machines with $f(k)\log n$ space; this corresponds to a class called XL. As XL^+ (the closure of XL under fpt reductions) contains AW[SAT] [30, Exercise 8.39], the supposition that the amended model is still equivalent to XSLP would imply the inclusion $\mathrm{AW}[\star] \subseteq \mathrm{AW}[\mathrm{SAT}] \subseteq \mathrm{XSLP}^+$. From the logic characterization that will be provided in Section 4.2 it follows that $\mathrm{AW}[\star] \supseteq \mathrm{XSLP}^+$, so in fact we would obtain a collapse $\mathrm{AW}[\star] = \mathrm{AW}[\mathrm{SAT}] = \mathrm{XSLP}^+$.
- If we increase the bound on allowed co-nondeterminism to $f(k)\log n$, thus matching the bound on the allowed nondeterminism, then it is not hard to see that the obtained machine model would be able to solve the model-checking problem for first-order logic on general relational structures, which is AW[$\star$]-complete. Consequently, if the amended machine model was still equivalent to XSLP, we would again obtain equality $\mathrm{AW}[\star] = \mathrm{XSLP}^+$, which we consider unlikely.
- If we let the machine use unbounded nondeterminism, then already for k constant and assuming no use of co-nondeterminism, our machines would be able to solve every problem in NL, including Directed Reachability. If the obtained machine model was still equivalent to XSLP, then Directed Reachability would be reducible (in L) to BinCSP on graphs of constant treedepth. But the latter problem is actually in L, so we would obtain L = NL.

- We believe that increasing the alternation from $f(k)$ to $f(k) + \mathcal{O}(\log n)$ yields a strictly more powerful machine model, though at this point we cannot pinpoint any concrete collapse that would be implied by the converse. However, it is not hard to check that an AROSM with resource bounds as in Theorem 10, but alternation $f(k) + \mathcal{O}(\log n)$, is able to solve BinCSP instances with Gaifman graphs of treedepth as large as $\log n$, but with all domains of size at most k. We do not see how to reduce this problem to BinCSP with domains of unbounded size, but treedepth bounded by $f(k)$.
- It is an interesting question whether the $f(k) \log n$ bound on the stack size can be lifted; that is, whether allowing unbounded stack size strictly increases the power of the considered machine model. On one hand, in all our proofs, the stack is essentially only used to store nondeterministic bits, and in any run there are at most $f(k) \log n$ of them anyway. So if the stack is used only for this purpose, then it is immaterial whether its size is bounded by $f(k) \log n$ or unbounded. On the other hand, the restriction on the stack size plays an important role in the proof of the implication (2) $\Rightarrow$ (1) of Theorem 10. We leave resolving this question open.

The remainder of this section is devoted to the proof of Theorem 10. Naturally, the argument is split into the forward and the backward implication.

We refer to the full version [12] for the proof of the simpler implication (1) $\Rightarrow$ (2), but briefly sketch it here. We use an AROSM to guess a satisfying assignment to the given ${}^{\text{BinCSP}}/_{\text{td}}$ instance, by going top-down through the associated forest. We use nondeterminism to guess the assignment for the next vertex u, and conondeterminism to verify whether the currently guessed partial assignment can be extended to all the subtrees rooted at the children of u.

We now proceed to the more difficult implication (2) $\Rightarrow$ (1) of Theorem 10. The main idea is that we introduce a restricted variant of a *regular* AROSM, which is an AROSM whose $\forall$ computation tree has a very specific shape, computable from k and the length of the input. We will then show two lemmas: (i) for every AROSM there is an equivalent regular one, and (ii) acceptance of a regular AROSM can be reduced to ${}^{\text{BinCSP}}/_{\text{td}}$. The main point in this strategy is that the assumption that the computation tree is fixed allows us to fix it as the elimination tree of the Gaifman graph of the constructed BinCSP instance.

More precisely, we will be working with the *contracted* $\forall$ *computation trees* defined as follows. Let T be a $\forall$ computation tree of an AROSM M, where without loss of generality we assume that the starting state of M is universal. A *universal block* of T is an inclusion-wise maximal subtree A of T such that the root of A is a universal configuration and A does not contain existential configurations. Note that removing all universal blocks from T breaks T into a collection of disjoint paths consisting only of deterministic and existential configurations; these will be called *existential blocks*. The *contraction* of T is the tree T' whose nodes are universal blocks of T, where the ancestor order is naturally inherited from T: one block is an ancestor of the other in T' if this holds for their roots in T. Note that a universal block B is a child of a universal block A in T' if and only if there is an existential block C that connects the root of B with a leaf of A. Thus, the edges of T' are in one-to-one correspondence with the existential blocks of T.

▶ **Definition 11.** *An AROSM M is* regular *if given $(1^n, k)$ one can in parameterized logspace compute a rooted tree $T_{n,k}$ with the following properties:*

- *$T_{n,k}$ has depth at most $f(k)$, for some computable function f; and*
- *for any input (x, k) with $|x| = n$, if M accepts (x, k), then M has a $\forall$ computation tree accepting (x, k) whose contraction is $T_{n,k}$.*

With this definition in place, we can state the two lemmas described before.

▶ **Lemma 12.** *If a parameterized problem Q can be decided by an AROSM M using the resource bounds stated in Theorem 10, then it can also be decided by a regular AROSM M' using such resource bounds.*

▶ **Lemma 13.** *If Q can be decided by a regular AROSM M using the resource bounds stated in Theorem 10, then $Q \in$ XSLP.*

The (2) ⇒ (1) implication of Theorem 10 follows directly by combining the two lemmas above. The proof of Lemma 13 is a conceptually straightforward, though technically a bit involved encoding of a ∀ computation tree of the machine through an instance of BinCSP whose elimination tree is (roughly) $T_{n,k}$. We give this proof in the full version [12]. The proof of Lemma 12 is the interesting part of the argument, as it involves the notion of *universal trees*.

Before we proceed, let us state a simple lemma that is used in our proofs several times. We included a proof in the full version [12] for completeness.

▶ **Lemma 14.** *Suppose T is a rooted tree with N leaves. Then there exists a labelling λ that maps every edge e of T to a binary string $\lambda(e) \in \{0,1\}^\star$ with the following properties:*
- *For every node u, the labels of edges connecting u with its children are pairwise different.*
- *For every leaf ℓ, the total length of labels on the root-to-ℓ path in T is at most $\lceil \log N \rceil$.*

Moreover, given T the labelling λ can be computed in deterministic logarithmic space.

4.1.1 Regularization

We now prove Lemma 12. We need the following definitions. An *ordered tree* is a rooted tree where for every vertex u there is a linear order $\preceq$ on the children of u. An *embedding* of an ordered tree S into an ordered tree T is an injective mapping $\phi\colon V(S) \to V(T)$ such that
- the root of S is mapped to the root of T, and
- for every node u of S, the children of u in S are mapped to distinct children of $\phi(u)$ in T in an order-preserving way: if $v \prec v'$ are distinct children of u in S, then $\phi(v) \prec \phi(v')$.

We will use the following result about the existence of *universal trees*.

▶ **Lemma 15** (follows from Jurdziński and Lazić [38], see also Theorem 2.2 of [18])**.** *For every pair of integers $n, k \in \mathbb{N}$ there exists an ordered tree $U_{n,k}$ such that*
- *$U_{n,k}$ has depth k;*
- *$U_{n,k}$ has at most $2n \cdot \binom{\lceil \log n \rceil + k + 1}{k}$ leaves; and*
- *for every ordered tree T of depth at most k and with at most n leaves, there is an embedding of T into $U_{n,k}$.*

Moreover, given $(1^n, k)$, the tree $U_{n,k}$ can be computed parameterized logspace.

We remark that the claim about the computability of $U_{n,k}$ in parameterized logspace is not present in [18, 38], but follows directly from the recursive construction presented there. In fact, we will also need the following property, which again follows directly from the construction, and which strengthens the embedding property stated in Lemma 15.

▶ **Lemma 16.** *For every node u of $U_{n,k}$, the subtree of $U_{n,k}$ rooted at u is isomorphic to $U_{n',k'}$ for some $n' \leqslant n$ and $k' \leqslant k$; the labeling of nodes of $U_{n,k}$ with suitable numbers n', k' can be computed along with $U_{n,k}$ within the algorithm of Lemma 15. Moreover, if $n_1, \ldots, n_p$ are nonnegative integers such that $n_1 + \ldots + n_p \leqslant n$, then there are distinct children $v_1 \prec v_2 \prec \ldots \prec v_p$ of the root of $U_{n,k}$ such that for every $i \in \{1, \ldots, p\}$, the subtree of $U_{n,k}$ rooted at v_i is isomorphic to $U_{n'_i,k-1}$ for some $n'_i \geqslant n_i$.*

Finally, observe that

$$2n \cdot \binom{\lceil \log n \rceil + k + 1}{k} \leqslant 2n \cdot 2^{\lceil \log n \rceil + k + 1} \leqslant \mathcal{O}(2^k \cdot n^2),$$

hence $U_{n,k}$ has $\mathcal{O}(2^k \cdot n^2)$ leaves.

We proceed to the proof of Lemma 12. Let us fix an AROSM M that on any input (x, k) with $|x| \leqslant n$, uses $f(k) \log n$ nondeterminism, $f(k) + d \log n$ conondeterminism, $f(k)$ alternation, $f(k) + d \log n$ working space, and $f(k) \log n$ stack size, where f is a computable function and $d \in \mathbb{N}$ is a constant. We may assume w.l.o.g. that the starting state of M is universal. Denote $K = f(k)$ and $N = 2^{f(k) + \lceil d \log n \rceil} \leqslant 2^{f(k)+1} \cdot n^d$; then K is an upper bound on the depth and N is an upper bound on the total number of leaves of any $\forall$ computation tree accepting (x, k) within the stipulated resources. By Lemma 15, we may compute the universal tree $U_{N,K}$ in deterministic space $h(k) + \mathcal{O}(\log n)$ for a computable function h. Note that $U_{N,K}$ has $N' = \mathcal{O}(2^K \cdot N^2) \leqslant \mathcal{O}(2^{3f(k)} \cdot n^{2d})$ leaves. The tree $U_{N,K}$ will serve the role of $T_{n,k}$ in the proof. Also, we use Lemma 14 to compute a suitable labeling λ of the edges of $U_{N,K}$ in which the total length of labels on every branch of $U_{N,K}$ is at most $\lceil \log N' \rceil \leqslant 3f(k) + 2d \log n + \mathcal{O}(1)$.

We are left with designing an AROSM M' that is equivalent to M, in the sense that M' accepts input (x, k) if and only if M does, and in such case the contracted $\forall$ computation tree of M' on (x, k) may be $U_{N,K}$. The idea is that machine M' will simulate M while inserting some dummy computation to "fill" the contracted $\forall$ computation tree of M to $U_{N,K}$. However, we will need to be very careful about how the conondeterminism of M is simulated.

A *stackless configuration* is a configuration of M, but without specifying the content of the stack; that is, it consists of the state of M, the content of the working tape, and the positions of the heads on the input and the working tapes. For a universal stackless configuration c of M, we define the *universal block* rooted at c, denoted $U(c)$, as a rooted tree of stackless configurations that is obtained just as the $\forall$ computation tree, except that M starts at c and we do not continue the simulation once the final state or any existential configuration is reached. Note here since M cannot read the stack except for the end of the computation, $U(c)$ is uniquely defined for every stackless configuration c. Thus, the leaves of $U(c)$ are existential or final (stackless) configurations, and whenever c is present in a $\forall$ computation tree T of M, T contains a copy of $U(c)$ rooted at c as a subtree.

The next claim shows that given a stackless configuration c, the universal block $U(c)$ can be computed within the allowed resources.

▷ Claim 17. Given a stackless configuration c of M, the universal block $U(c)$, together with a labelling of its edges with transitions taken, can be computed in deterministic space $h(k) + \mathcal{O}(\log n)$, for some computable h.

Proof. Let $Z = f(k) + \lfloor d \log n \rfloor$. Observe that for every binary string $r \in \{0, 1\}^Z$, we can compute the branch of $U(c)$ that takes the consecutive conondeterministic choices as prescribed by the consecutive bits of r. To do this, just simulate M starting from c and, whenever a conondeterministic choice needs to be made, use the next bit of r to determine how it is resolved. (This simulation stops when an existential or a final configuration is encountered.) Having this subprocedure, the whole $U(c)$ can be easily computed by iterating through consecutive strings $r \in \{0, 1\}^Z$ and outputting the branches of $U(c)$ one after the other. (Strictly speaking, from every next branch we output only the part after diverging from the previous branch.) Finally, note that r can be stored within the allowed space. ◁

With Claim 17 established, we proceed to the construction of M'. For the sake of the proof, suppose M has a $\forall$ computation tree T that is accepting and uses the allowed resources. Machine M' tries to verifies the existence of such T by traversing the universal tree $U_{N,K}$ and guessing, along the way, how the contraction T' of T embeds into $U_{N,K}$. By Lemma 15 we know that such an embedding always exists. The traversal of $U_{N,K}$ will be done in such a way that the contracted $\forall$ computation tree of M' will be always $U_{N,K}$.

At every point of computation, M' stores on its working tape a node u of $U_{N,K}$ and its contracted $\forall$ computation tree from this point on should be the subtree of $U_{N,K}$ rooted at u. Machine M' is always either in the *real mode* or in the *dummy mode*. In the real mode, M' is in the process of guessing a subtree of T. Therefore, then M' holds the following data:

- On the working tape, M' stores a stackless configuration c of M. The reader should think of c as of the configuration of M at the root of a universal block of T.
- On its own stack, M' holds the content of the stack of M.
- Additionally on the working tape, M' stores two integers a and b, denoting the total number of nondeterministic and conondeterministic bits used by M so far, respectively. (In other words, a and b are the total number of existential and universal configurations visited so far on a branch of T.) We maintain the following invariant: the subtree of $U_{N,K}$ rooted at u is $U_{N',K'}$ for some $K' \leqslant K$ and N' such that $N' \geqslant N/2^b$.

Then the task of M' is to verify the existence of a subtree S of a $\forall$ computation tree of M such that

- S has c supplied with the current content of the stack at its root;
- S embeds into the subtree of $U_{N,K}$ rooted at u;
- the nondeterminism and the conondeterminism of S together with a and b add to at most $f(k)\log n$ and $f(k) + d\log n$, respectively; and
- S is accepting, that is, every leaf of S is an accepting configuration.

In the dummy mode, M' is not guessing any part of T, so its task is to perform some meaningless computation in order to make its contracted $\forall$ computation tree equal to the subtree of $U_{N,K}$ rooted at u. So in this mode, M' holds on its working tape only the node u.

We now explain steps taken by M' in the real mode. Given c, M' applies the algorithm of Claim 17 to compute the universal block $U(c)$. (Formally speaking, $U(c)$ is not computed explicitly, as it would not fit within the working space, but at any point a bit from the description of $U(c)$ is needed, we run the algorithm of Claim 17 to compute this bit.) Let $\ell_1, \ldots, \ell_p$ be the leaves of $U(c)$, in the order as they appear in the description of $U(c)$. Informally, we wish to fit in $U(c)$ into the computation tree of M' while keeping enough "space" for the remaining computations M may wish to perform, without knowing how the computation will continue at the leaves. For every $i \in \{1, \ldots, p\}$, let b_i be total number of universal configurations on the branch of $U(c)$ finishing at ℓ_i. By assumption, the subtree of $U_{N,K}$ rooted at u is isomorphic to $U_{N',K'}$ for some $N' \geqslant N/2^b$ and $K' \leqslant K$. Similarly, we would like to find children $v_1 \prec v_2 \prec \ldots \prec v_p$ of u in $U_{N,K}$ such that the subtree rooted at each v_i is isomorphic to $U_{N_i,K'-1}$ where $N_i \geqslant N/2^{b+b_i}$. This follows from Lemma 16: we check that

$$\sum_{i=1}^{p} N/2^{b+b_i} \leqslant N' \sum_{i=1}^{p} 2^{-b_i} = N',$$

where the last equality follows since $U(c)$ is a binary tree. Note that given $b_1, \ldots, b_p$, we may compute the corresponding children $v_1, \ldots, v_p$ with sufficiently large subtrees in logarithmic space greedily: having found v_i, we can set v_{i+1} to be the $\prec$-smallest child v of u such that $v_i \prec v$ and the subtree rooted at v is isomorphic to $U_{N'',K'-1}$ for some $N'' \geqslant m_i$. Hence,

from now on we assume that the children $v_1, \ldots, v_p$ are given to us. (Again, formally, when we need any v_i, we run the logarithmic space algorithm computing $v_1, \ldots, v_p$ to retrieve the sought v_i.)

Machine M' conondeterministically guesses the label $\lambda(uv)$ of an edge uv connecting u with a child v; this can be done using conondeterministic $2|\lambda(uv)| + 1$ bits[3]. Noting that the pair $(u, \lambda(uv))$ uniquely determines v, we can now compute v. We have two cases:

- Suppose $v = v_i$ for some $i \in \{1, \ldots, p\}$. Then M' simulates all transitions of M on the path from c to the leaf ℓ_i in $U(c)$ (this may include some push operations). If ℓ_i is a final configuration, M' finishes the computation and verifies acceptance in the same way M would do. Otherwise, if ℓ_i is an existential configuration, M' further nondeterministically simulates M using its own nondeterminism, until a final or a universal configuration is encountered, or the bound of $f(k) \log n$ on the total number of nondeterministic steps is exceeded (together with a). In case of a final configuration, we do the same as before: machine M' concludes the computation and verifies whether M accepts. In case of a universal configuration, say c', M' moves the currently considered node of $U_{N,K}$ from u to v, and proceeds with working with c' at v. The counters a and b are updated by the total number of nondeterministic and conondeterminisitc bits used between ℓ and c' and between c and ℓ, respectively. Note here that the content of the stack has been appropriately updated while simulating the transitions of M from c to c'.
- Suppose $v \notin \{v_1, \ldots, v_p\}$. Then M' moves the currently considered node of $U_{N,K}$ from u to v, but enters v in the dummy mode.

This concludes the description of the behavior of M' in the real mode.

Finally, when in the dummy mode, machine M' does as follows:

- If u is a leaf, M' just accepts.
- If u is not a leaf, M' conondeterministically chooses a label $\lambda(uv)$ of an edge uv connecting u with a child v, using $2|\lambda(u, v)|+1$ conondeterminstic bits. Then M' computes v, performs a trivial nondeterministic transition, and enters v, again in the dummy mode.

This completes the construction of M'.

From the construction it follows that the contracted $\forall$ computation tree of M' on (x, k) is always $U_{N,K}$, hence M' is regular. Moreover, on every branch M' uses as much nondeterminism as M, that is, at most $f(k) \log n$, while the conondeterminism of M' is bounded by $2\lceil \log N' \rceil + k \leqslant 6f(k) + 4d \log n + k + \mathcal{O}(1)$, by the assumed properties of the labeling λ. The maximum stack length of M' is the same as that of M, while on its working tape, M' holds always at most one configuration of M plus $h(k) + \mathcal{O}(\log n)$ additional bits, for some computable function h. Finally, since every contracted $\forall$ computation tree of M accepting (x, k) within prescribed resources embeds into $U_{N,K}$, it is straightforward to see from the construction that M' accepts (x, k) within the prescribed resources if and only if M does. This concludes the proof of Lemma 12, so the proof of Theorem 10 is also complete.

4.2 A logic characterization

We now provide another characterization of XSLP, by providing a complete problem related to model-checking first-order logic. This reflects the definitions of classes AW[$\star$] and of the A-hierarchy, see [30, Chapter 8].

[3] For instance, the machine can guess consecutive bits of $\lambda(uv)$ interleaved with symbols 0 and 1, where 0 denotes "continue guessing" and 1 denotes "end of $\lambda(uv)$".

We use the standard terminology for relational structures. A (relational) signature is a set Σ consisting of *relation symbols*, where each relation symbol R has a prescribed arity $\mathrm{ar}(R) \in \mathbb{N}$. A Σ-*structure* $\mathbb{A}$ consists of a universe U and, for every relation symbol $R \in \Sigma$, its *interpretation* $R^{\mathbb{A}} \subseteq U^{\mathrm{ar}(R)}$ in $\mathbb{A}$. In this paper we only consider *binary signatures*, that is, signatures where every relation has arity at most 2.

For a signature Σ, we may consider standard first-order logic over Σ-structures. In this logic there are variables for the elements of the universe. Atomic formulas are of the form $x = y$ and $R(x_1, \ldots, x_k)$ for some $R \in \Sigma$ with $k = \mathrm{ar}(R)$, with the obvious semantics. These can be used to form larger formulas by using Boolean connectives, negation, and quantification (both existential and universal).

A Σ-structure $\mathbb{A}$ is called *forest-shaped* if Σ contains a binary relation parent such that $\mathsf{parent}^{\mathbb{A}}$ is the parent relation on a rooted forest with the vertex set being the universe of $\mathbb{A}$, and a unary relation root such that $\mathsf{root}^{\mathbb{A}}$ is the set of roots of this forest. We say that a first-order sentence φ over Σ is $\forall$-*guided* if it is of the form:

$$\varphi \quad = \quad \forall x_1 \exists y_1 \ \ldots \ \forall x_k \exists y_k \ (\mathsf{root}(x_1) \wedge \mathsf{parent}(x_1, x_2) \wedge \ldots \wedge \mathsf{parent}(x_{k-1}, x_k)) \Rightarrow \psi(x_1, y_1, \ldots, x_k, y_k)$$

where ψ is quantifier-free. In other words, φ is in a prenex form starting with a universal quantifier, and moreover we require that the first universally quantified variable is a root and every next universally quantified variable is a child of the previous one. Note that there are no restrictions on existential quantification.

For a binary signature Σ, we consider the problem of model-checking $\forall$-guided formulas on forest-shaped Σ-structures. In this problem we are given a forest-shaped Σ-structure $\mathbb{A}$ and a $\forall$-guided sentence φ, and the question is whether φ holds in $\mathbb{A}$. We consider this as a parameterized problem where $\|\varphi\|$ – the total length of an encoding of the sentence φ – is the parameter.

The following statement provides a characterization of XSLP in terms of the model-checking problem described above.

▶ **Theorem 18.** *There exists a binary signature Σ such that the following conditions are equivalent for a parameterized problem Q.*

(1) $Q \in$ XSLP*;*

(2) Q *can be* pl*-reduced to the problem of model-checking $\forall$-guided sentences on forest-shaped Σ-structures.*

The proof of Theorem 18 can be found in the full version [12], but we sketch it here.

For the (1) $\Rightarrow$ (2) implication, it suffices to pl-reduce $\textsc{BinCSP}/_{\mathrm{td}}$ to the model-checking problem for $\forall$-guided sentences on forest-shaped structures. This is a fairly straightforward construction. Given an instance I of $\textsc{BinCSP}/_{\mathrm{td}}$, we build a relational structure $\mathbb{A}$ consisting of the (given) elimination forest F of the Gaifman graph of I and the disjoint union of domains $D(u)$ of variables u of I. These domains are bound to respective variables using a binary predicate, and there is another binary predicate encoding the constraints. Then it is straightforward to write a $\forall$-guided sentence φ that checks the satisfiability of I in a top-down manner on F: existential variables are used to guess the evaluation of variables of I, while universal variables are used to verify the possibility of extending the current partial evaluation further down.

For the (2) $\Rightarrow$ (1) implication, by Theorem 10 it suffices to design an AROSM M that solves the model-checking problem for $\forall$-guided sentences on forest-shaped Σ-structures within the bounds stipulated in Theorem 10. Machine M uses its conondeterminism and nondeterminism to universally and existentially guess the evaluations of consecutive variables

$x_1, y_1, \ldots, x_k, y_k$, within $2k$ rounds of alternation. Here, the assumption that the input sentence is $\forall$-guided and the input structure is forest-shaped can be used in conjunction with Lemma 14 to bound the total conondeterminism used by $\mathcal{O}(k + \log n)$. Once all variables are evaluated, satisfaction of ψ can be checked within 4 additional rounds of alternation by assuming without loss of generality that ψ is in DNF.

5 Conclusion

In this paper we explored the parameterized complexity of BinCSP for a variety of relatively strong structural parameters, including the vertex cover number, treedepth, and several modulator-based parameters. We believe that together with the previous works on XALP and XNLP [7, 9, 10, 11, 26], our work uncovers a rich complexity structure within the class XP, which is worth further exploration. We selected concrete open questions below.

- In [10, 11], several problems such as Independent Set or Dominating Set, which are fixed-parameter tractable when parameterized by treewidth, were shown to be XALP- and XNLP-complete when parameterized by the *logarithmic* treewidth and pathwidth, which is at most k when the corresponding width measure is at most $k \log n$. Can one prove similar results for the class XSLP and parameterization by logarithmic treedepth?
- Theorem 3 provides natural complete problems only for the odd levels of the W-hierarchy. Similarly, we defined the S-hierarchy only for odd levels. It would be interesting to have a natural description of the situation also for the even levels.
- The characterizations of XSLP given by Theorems 10 and 18 can be "projected" to a rough characterizations of classes S[d] for odd d by stipulating that the alternation is at most d. Unfortunately, this projection turns out not to be completely faithful: the obtained problems do not precisely characterize the class S[d], but lie somewhere between S[$d - \mathcal{O}(1)$] and S[$d + \mathcal{O}(1)$]. Can we provide a compelling description of the levels of the S-hierarchy in terms of machine problems or in terms of model-checking first-order logic?
- What is the complexity of List Coloring parameterized by the vertex cover number? Currently, we know it is W[1]-hard and in W[2]. Similarly, what is the complexity of List Coloring and Precoloring Extension with the minimum size of a modulator to a treedepth-d graph as the parameter?
- Can one obtain a better understanding of the complexity of BinCSP and List Coloring parameterized by the feedback vertex number?

References

1 Faisal N. Abu-Khzam, Henning Fernau, Benjamin Gras, Mathieu Liedloff, and Kevin Mann. Enumerating minimal connected dominating sets. In *30th Annual European Symposium on Algorithms, ESA 2022*, volume 244 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 1:1–1:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ESA.2022.1.

2 Kunwarjit S. Bagga, Lowell W. Beineke, Wayne D. Goddard, Marc J. Lipman, and Raymond E. Pippert. A survey of integrity. *Discrete Applied Mathematics*, 37:13–28, 1992. doi:10.1016/0166-218X(92)90122-Q.

3 Aritra Banik, Ashwin Jacob, Vijay Kumar Paliwal, and Venkatesh Raman. Fixed-parameter tractability of $(n - k)$ list coloring. *Theory of Computing Systems*, 64(7):1307–1316, 2020. doi:10.1007/s00224-020-10014-9.

4 Curtis A. Barefoot, Roger Entringer, and Henda Swart. Vulnerability in graphs—a comparative survey. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 1(38):13–22, 1987.

5 Thomas Bläsius, Tobias Friedrich, Julius Lischeid, Kitty Meeks, and Martin Schirneck. Efficiently enumerating hitting sets of hypergraphs arising in data profiling. *Journal of Computer and System Sciences*, 124:192–213, 2022. doi:10.1016/j.jcss.2021.10.002.

6 Thomas Bläsius, Tobias Friedrich, and Martin Schirneck. The complexity of dependency detection and discovery in relational databases. *Theoretical Computer Science*, 900:79–96, 2022. doi:10.1016/j.tcs.2021.11.020.

7 Hans L. Bodlaender, Gunther Cornelissen, and Marieke van der Wegen. Problems hard for treewidth but easy for stable gonality. In *48th International Workshop on Graph-Theoretic Concepts in Computer Science, WG 2022*, volume 13453 of *Lecture Notes in Computer Science*, pages 84–97. Springer, 2022. doi:10.1007/978-3-031-15914-5_7.

8 Hans L. Bodlaender, Carla Groenland, and Hugo Jacob. List Colouring Trees in Logarithmic Space. In *Proceedings 30th Annual European Symposium on Algorithms, ESA 2022*, volume 244 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 24:1–24:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ESA.2022.24.

9 Hans L. Bodlaender, Carla Groenland, Hugo Jacob, Lars Jaffke, and Paloma T. Lima. XNLP-completeness for parameterized problems on graphs with a linear structure. In *Proceedings 17th International Symposium on Parameterized and Exact Computation, IPEC 2022*, volume 249 of *LIPIcs*, pages 8:1–8:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.IPEC.2022.8.

10 Hans L. Bodlaender, Carla Groenland, Hugo Jacob, Marcin Pilipczuk, and Michał Pilipczuk. On the complexity of problems on tree-structured graphs. In *17th International Symposium on Parameterized and Exact Computation, IPEC 2022*, volume 249 of *LIPIcs*, pages 6:1–6:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.IPEC.2022.6.

11 Hans L. Bodlaender, Carla Groenland, Jesper Nederlof, and Céline 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*, pages 193–204. IEEE, 2021. doi:10.1109/FOCS52979.2021.00027.

12 Hans L. Bodlaender, Carla Groenland, and Michał Pilipczuk. Parameterized complexity of binary csp: Vertex cover, treedepth, and related parameters. *CoRR*, abs/2208.12543, 2022. arXiv:2208.12543.

13 Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasi-polynomial time. *SIAM J. Comput.*, 51(2):17–152, 2022. doi:10.1137/17m1145288.

14 Katrin Casel, Henning Fernau, Mehdi Khosravian Ghadikolaei, Jérôme Monnot, and Florian Sikora. On the complexity of solution extension of optimization problems. *Theoretical Computer Science*, 904:48–65, 2022. doi:10.1016/j.tcs.2021.10.017.

15 Jianer Chen and Fenghui Zhang. On product covering in 3-tier supply chain models: Natural complete problems for $W[3]$ and $W[4]$. *Theoretical Computer Science*, 363(3):278–288, 2006. doi:10.1016/j.tcs.2006.07.016.

16 Jiehua Chen, Wojciech Czerwiński, Yann Disser, Andreas Emil Feldmann, Danny Hermelin, Wojciech Nadara, Marcin Pilipczuk, Michał Pilipczuk, Manuel Sorge, Bartlomiej Wróblewski, and Anna Zych-Pawlewicz. Efficient fully dynamic elimination forests with applications to detecting long paths and cycles. In *2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 796–809. SIAM, 2021. doi:10.1137/1.9781611976465.50.

17 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

18 Wojciech Czerwiński, Laure Daviaud, Nathanaël Fijalkow, Marcin Jurdziński, Ranko Lazić, and Pawel Parys. Universal trees grow inside separating automata: Quasi-polynomial lower bounds for parity games. In *Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 2333–2349. SIAM, 2019. doi:10.1137/1.9781611975482.142.

19 Wojciech Czerwiński, Wojciech Nadara, and Marcin Pilipczuk. Improved bounds for the excluded-minor approximation of treedepth. *SIAM J. Discret. Math.*, 35(2):934–947, 2021. doi:10.1137/19M128819X.

20 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer, 1999. doi:10.1007/978-1-4612-0515-9.

21 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

22 Zdenek Dvořák, Archontia C. Giannopoulou, and Dimitrios M. Thilikos. Forbidden graphs for tree-depth. *Eur. J. Comb.*, 33(5):969–979, 2012. doi:10.1016/j.ejc.2011.09.014.

23 Pavel Dvořák, Eduard Eiben, Robert Ganian, Dušan Knop, and Sebastian Ordyniak. The complexity landscape of decompositional parameters for ILP: Programs with few global variables and constraints. *Artificial Intelligence*, 300:103561, 2021. doi:10.1016/j.artint.2021.103561.

24 Friedrich Eisenbrand, Christoph Hunkenschröder, Kim-Manuel Klein, Martin Koutecký, Asaf Levin, and Shmuel Onn. An algorithmic theory of integer programming. *CoRR*, abs/1904.01361, 2019. arXiv:1904.01361.

25 Michael Elberfeld, Martin Grohe, and Till Tantau. Where first-order and monadic second-order logic coincide. *ACM Trans. Comput. Log.*, 17(4):25, 2016. doi:10.1145/2946799.

26 Michael Elberfeld, Christoph Stockhusen, and Till Tantau. On the space and circuit complexity of parameterized problems: Classes and completeness. *Algorithmica*, 71(3):661–701, 2015. doi:10.1007/s00453-014-9944-y.

27 Michael R. Fellows, Danny Hermelin, Frances A. Rosamond, and Stéphane Vialette. On the parameterized complexity of multiple-interval graph problems. *Theoretical Computer Science*, 410(1):53–61, 2009. doi:10.1016/j.tcs.2008.09.065.

28 Michael R. Fellows, Daniel Lokshtanov, Neeldhara Misra, Frances A. Rosamond, and Saket Saurabh. Graph layout problems parameterized by vertex cover. In *19th International Symposium on Algorithms and Computation, ISAAC 2008*, volume 5369 of *Lecture Notes in Computer Science*, pages 294–305. Springer, 2008. doi:10.1007/978-3-540-92182-0_28.

29 Jirí Fiala, Petr A. Golovach, and Jan Kratochvíl. Parameterized complexity of coloring problems: Treewidth versus vertex cover. *Theoretical Computer Science*, 412(23):2513–2523, 2011. doi:10.1016/j.tcs.2010.10.043.

30 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. doi:10.1007/3-540-29953-X.

31 Fedor V. Fomin, Bart M. P. Jansen, and Michał Pilipczuk. Preprocessing subgraph and minor problems: When does a small vertex cover help? *Journal of Computer and System Sciences*, 80(2):468–495, 2014. doi:10.1016/j.jcss.2013.09.004.

32 Eugene C. Freuder. Complexity of K-tree structured Constraint Satisfaction Problems. In *8th National Conference on Artificial Intelligence, AAAI-90*, pages 4–9. AAAI Press / The MIT Press, 1990. URL: http://www.aaai.org/Library/AAAI/1990/aaai90-001.php.

33 Martin Fürer and Huiwen Yu. Space saving by dynamic algebraization. In *9th International Computer Science Symposium in Russia, CSR 2014*, volume 8476 of *Lecture Notes in Computer Science*, pages 375–388. Springer, 2014. doi:10.1007/978-3-319-06686-8_29.

34 Robert Ganian. Improving vertex cover as a graph parameter. *Discrete Mathematics and Theoretical Computer Science*, 17(2):77–100, 2015. doi:10.46298/dmtcs.2136.

35 Gregory Z. Gutin, Diptapriyo Majumdar, Sebastian Ordyniak, and Magnus Wahlström. Parameterized pre-coloring extension and list coloring problems. *SIAM Journal on Discrete Mathematics*, 35(1):575–596, 2021. doi:10.1137/20M1323369.

36 Miika Hannula, Bor-Kuan Song, and Sebastian Link. An algorithm for the discovery of independence from data. *arXiv*, abs/2101.02502, 2021. arXiv:2101.02502.

37 Falko Hegerfeld and Stefan Kratsch. Solving connectivity problems parameterized by treedepth in single-exponential time and polynomial space. In *37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020*, volume 154 of *LIPIcs*, pages 29:1–29:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.STACS.2020.29.

38 Marcin Jurdziński and Ranko Lazić. Succinct progress measures for solving parity games. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017*, pages 1–9. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005092.

39 Ken-ichi Kawarabayashi and Benjamin Rossman. A polynomial excluded-minor approximation of treedepth. In *2018 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, pages 234–246. SIAM, 2018. doi:10.1137/1.9781611975031.17.

40 Deepanshu Kush and Benjamin Rossman. Tree-depth and the formula complexity of subgraph isomorphism. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*, pages 31–42. IEEE, 2020. doi:10.1109/FOCS46700.2020.00012.

41 Wojciech Nadara, Michał Pilipczuk, and Marcin Smulewicz. Computing treedepth in polynomial space and linear FPT time. In *30th Annual European Symposium on Algorithms, ESA 2022*, volume 244 of *LIPIcs*, pages 79:1–79:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ESA.2022.79.

42 Jesper Nederlof, Michał Pilipczuk, Céline M. F. Swennenhuis, and Karol Węgrzycki. Hamiltonian Cycle parameterized by treedepth in single exponential time and polynomial space. In *46th International Workshop on Graph-Theoretic Concepts in Computer Science*, volume 12301 of *Lecture Notes in Computer Science*, pages 27–39. Springer, 2020. doi:10.1007/978-3-030-60440-0_3.

43 Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity – Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012. doi:10.1007/978-3-642-27875-4.

44 Christos H. Papadimitriou and Mihalis Yannakakis. On the complexity of database queries. *Journal of Computer and System Sciences*, 58(3):407–427, 1999. doi:10.1006/jcss.1999.1626.

45 Michał Pilipczuk and Sebastian Siebertz. Polynomial bounds for centered colorings on proper minor-closed graph classes. *J. Comb. Theory, Ser. B*, 151:111–147, 2021. doi:10.1016/j.jctb.2021.06.002.

46 Michał Pilipczuk and Marcin Wrochna. On space efficiency of algorithms working on structural decompositions of graphs. *ACM Trans. Comput. Theory*, 9(4):18:1–18:36, 2018. doi:10.1145/3154856.