On the Complexity of Problems on Tree-structured Graphs

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

Carla Groenland
Department of Information and Computing Sciences, Utrecht University

Hugo Jacob
ENS Paris-Saclay, France

Marcin Pilipczuk
University of Warsaw

Michał Pilipczuk
University of Warsaw

Abstract

In this paper, we introduce a new class of parameterized problems, which we call XALP: the class of all parameterized problems that can be solved in $f(k)n^{O(1)}$ time and $f(k)\log n$ space on a non-deterministic Turing Machine with access to an auxiliary stack (with only top element lookup allowed). Various natural problems on 'tree-structured graphs' are complete for this class: we show that List Coloring and All-or-Nothing Flow parameterized by treewidth are XALP-complete. Moreover, Independent Set and Dominating Set parameterized by treewidth divided by $\log n$, and Max Cut parameterized by cliquewidth are also XALP-complete.

Besides finding a 'natural home' for these problems, we also pave the road for future reductions. We give a number of equivalent characterisations of the class XALP, e.g., XALP is the class of problems solvable by an Alternating Turing Machine whose runs have tree size at most $f(k)n^{O(1)}$ and use $f(k)\log n$ space. Moreover, we introduce 'tree-shaped' variants of Weighted CNF-Satisfiability and Multicolor Clique that are XALP-complete.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Graph algorithms analysis

Keywords and phrases Parameterized Complexity, Treewidth, XALP, XNLP

Funding *Carla Groenland*: Supported by the European Union's Horizon 2020 research and innovation programme under the ERC grant CRACKNP (number 853234) and the Marie Skłodowska-Curie grant GRAPHCOSY (number 101063180).
Marcin Pilipczuk: This research is a part of a project that have received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme Grant Agreement 714704.
Michał Pilipczuk: This research is a part of a project that have received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme Grant Agreement 948057.

Acknowledgements We would like to thank the organizers of the workshop on Parameterized complexity and discrete optimization, organized at HIM in Bonn, for providing a productive research environment. We would also like to thank our referees for useful suggestions.

1 Introduction

A central concept in complexity theory is *completeness* for a class of problems. Establishing completeness of a problem for a class pinpoints its difficulty, and gives implications on resources (time, memory or otherwise) to solve the problem (often, conditionally on complexity theoretic assumptions). The introduction of the W-hierarchy by Downey and Fellows in the 1990s played an essential role in the analysis of the complexity of parameterized problems [12, 13, 14]. Still, several problems are suspected not to be complete for a class in the W-hierarchy, and other classes of parameterized problems with complete problems were introduced, e.g., the A-, AW-, and M-hierarchies. (See e.g., [1, 14, 18].) In this paper, we introduce a new class of parameterized complexity, which appears to be the natural home of several 'tree structured' parameterized problems. This class, which we call XALP, can be seen as the parameterized version of a class known in classic complexity theory as NAuxPDA[poly, log] (see [3]), or ASPSZ($\log n$, $n^{O(1)}$) [23].

It can also be seen as the 'tree variant' of the class XNLP, which is the class of parameterized problems that can be solved by a non-deterministic Turing machine using $f(k)\log n$ space in $f(k)n^{O(1)}$ time for some computable function f, where k denotes the parameter and n the input size. It was introduced in 2015 by Elberfeld et al. [16]. Recently, several parameterized problems were shown to be complete for XNLP [4, 8, 7]; in this collection, we find many problems for 'path-structured graphs', including well known problems that are in XP with pathwidth or other linear width measures as parameter, and linear ordering graph problems like Bandwidth.

Thus, we can view XALP as the 'tree' variant of XNLP and as such, we expect that many problems known to be in XP (and expected not to be in FPT) when parameterized by treewidth will be complete for this class. We will prove the following problems to be XALP-complete in this paper:

- List Coloring and All-or-Nothing Flow parameterized by treewidth;
- Independent Set and Dominating Set parameterized by treewidth divided by $\log n$, where n is the number of vertices of the input graph;
- Max Cut parameterized by cliquewidth.

The problems listed in this paper should be regarded as examples of a general technique, and we expect that many other problems parameterized by treewidth, cliquewidth and similar parameters will be XALP-complete. In many cases, a simple modification of an XNLP-hardness proof with pathwidth as parameter shows XALP-hardness for the same problem with treewidth as parameter.

In addition to pinpointing the exact complexity class for these problems, such results have further consequences. First, XALP-completeness implies XNLP-hardness, and thus hardness for all classes W[t], $t \in \mathbb{N}$. Second, a conjecture by Pilipczuk and Wrochna [22], if true, implies that every algorithm for an XALP-complete problem that works in XP time (that is, $n^{f(k)}$ time) cannot simultaneously use FPT space (that is, $f(k)n^{O(1)}$ space). Indeed, typical XP algorithms for problems on graphs of bounded treewidth use dynamic programming, with tables that are of size $n^{f(k)}$.

Satisfiability on graphs of small treewidth Real-world SAT instances tend to have a special structure to them. One of the measures capturing the structure is the *treewidth* $\mathcal{TW}(\phi)$ of the given formula ϕ. This is defined by taking the treewidth of an associated graph, usually a bipartite graph on the variables on one side and the clauses on the other, where there is an edge if the variable appears in the clause. Alekhnovitch and Razborov [2] raised the question of whether satisfiability of formulas of small treewidth can be checked in polynomial space, which was positively answered by Allender et al. [3]. However, the running time of the algorithm is $3^{\mathcal{TW}(\phi)\log|\phi|}$ rather than $2^{O(\mathcal{TW}(\phi))}|\phi|^{O(1)}$, where $|\phi| = n + m$ for n the number of variables and m the number of clauses. They also conjectured that the $\log|\phi|$ factor in the exponent for the running time cannot be improved upon without using exponential space.

To support this conjecture, Allender et al. [3] show that SATISFIABILITY where the treewidth of the associated graph is $O(\log n)$ is complete for a class of problems called SAC^1: these are the problems that can be recognized by 'uniform' circuits with semi-unbounded fan-in of depth $O(\log n)$ and polynomial size. This class has also been shown to be equivalent to classes of problems that are defined using Alternating Turing Machines and non-deterministic Turing machines with access to an auxiliary stack [23, 25]. We define parameterized analogues of the classes defined using Alternating Turning Machines or non-deterministic Turing machines with access to an auxiliary stack, and show these to be equivalent. This is how we define our class XALP.

Allender et al. [3] considers SATISFIABILITY where the treewidth of the associated graph is $O(\log^k n)$ for all $k \geq 1$. We restrict ourselves to the case $k = 1$ since this is where we could find interesting complete problems, but we expect that a similar generalisation is possible in our setting.

The main contribution of our paper is to transfer definitions and results from the classical world to the parameterized setting, by which we provide a natural framework to establish the complexity of many well-known parameterized problems. We provide a number of natural XALP-complete problems, but we expect that in the future it will be shown that XALP is the 'right box' for many more problems of interest.

Paper overview In Section 2, we give a number of definitions, discuss the classical analogues of XALP, and formulate a number of key parameterized problems. Several equivalent characterizations of the class XALP are given in Section 3. In Section 4, we introduce a 'tree variant' of the wellknown MULTICOLOR CLIQUE problem. We call this problem TREE-CHAINED MULTICOLOR CLIQUE, and show it to be XALP-hard with a direct proof from an acceptance problem of a suitable type of Turing Machine, inspired by Cook's proof of the NP-completeness of SATISFIABILITY [10]. In Section 5, we build on this and give a number of other examples of XALP-complete problems, including tree variants of WEIGHTED SATISFIABILITY and several problems parameterized by treewidth or another tree-structured graph parameter.

2 Definitions

We assume that the reader is familiar with a number of well-known notions from graph theory and parameterized complexity, e.g., FPT, the W-hierarchy, clique, independent set, etc. (See e.g., [11].)

A *tree decomposition* of a graph $G = (V, E)$ is a pair $(T = (I, F), \{X_i \mid i \in T\})$ with $T = (I, F)$ a tree and $\{X_i \mid i \in I\})$ a family of (not necessarily disjoint) subsets of V (called *bags*) such that $\bigcup_{i \in I} X_i = V$, for all edges $vw \in E$, there is an i with $v, w \in X_i$, and for all v, the nodes $\{i \in I \mid v \in X_i\}$ form a connected subtree of T. The *width* of a tree decomposition $(T, \{X_i \mid i \in T\})$ is $\max_{i \in I} |X_i| - 1$, and the *treewidth* of a graph G is the maximum width over all tree decompositions of G. A *path decomposition* is a tree decomposition $(T = (I, F), \{X_i \mid i \in T\})$ with T a path, and the *pathwidth* is the minimum width over all path decompositions of G.

2.1 Turing Machines and Classes

We assume the reader to be familiar with the basic concept of a Turing Machine. Here, we consider TMs that have access to both a fixed input tape (where the machine can only read), and a work tape of specified size (where the machine can both read and write). We consider

Non-deterministic Turing Machines (NTM), where the machine can choose between different transitions, and accepts, if at least one choice of transitions leads to an accepting state, and *Alternating Turing Machines* (ATM), where the machine can both make non-deterministic steps (accepting when at least one choice leads to acceptance), and *co-non-deterministic steps* (accepting when both choices lead to acceptance). We assume a co-non-deterministic step always makes a binary choice, i.e, there are exactly two transitions that can be done.

Acceptance of an ATM A can be modelled by a rooted binary tree T, sometimes called a *run* or a *computation tree* of the machine. Each node of T is labelled with a configuration of A: the 4-tuple consisting of the machine state, work tape contents, location of work tape pointer, and location of input tape pointer. Each edge of T is labelled with a transition. The starting configuration is represented by the root of T. A node with one child makes a non-deterministic step, and the arc is labelled with a transition that leads to acceptance; a node with two children makes a co-non-deterministic step, with the children the configurations after the co-non-deterministic choice. Each leaf is a configuration with an accepting state. The *time* of the computation is the depth of the tree; the *treesize* is the total number of nodes in this computation tree. For more information, see e.g., [23, 22]. A *computation path* is a path from root to leaf in the tree.

We also consider NTMs which additionally have access to an auxiliary stack. For those, a transition can also move the top element of the stack to the current location of the work tape ('pop'), or put a symbol at the top of the stack ('push'). We stress that only the top element can be accessed or modified, the machine cannot freely read other elements on the stack.

We use the notation $\mathrm{N}[t(n,k), s(n,k)]$ to denote languages recognizable by a NTM running in time $t(n,k)$ with $s(n,k)$ working space and $\mathrm{A}[t(n,k), s(n,k)]$ to denote languages recognizable by an ATM running in treesize $t(n,k)$ with $s(n,k)$ working space. We note that we are free to put the constraint that *all* runs have treesize at most $t(n,k)$, since we can add a counter that keeps track of the number of remaining steps, and reject when this runs out (similar to what is done in the proof of Theorem 1). We write $\mathrm{NAuxPDA}[t(n,k), s(n,k)]$ to denote languages recognizable by a NTM with a stack (AUXiliary Push-Down Automaton) running in time $t(n,k)$ with $s(n,k)$ working space.

Ruzzo [23] showed that for any function $s(n)$, $\mathrm{NAuxPDA}[n^{O(1)}$ time, $s(n)$ space$] = \mathrm{A}[n^{O(1)}$ treesize, $s(n)$ space$]$. Allender et al. [3] provided natural complete problems when $s(n) = \log^k(n)$ for all $k \geq 1$ (via a circuit model called SAC, which we will not use in our paper). Our interest lies in the case $k = 1$, where it turns out the parameterized analogue is the natural home of 'tree-like' problems.

Another related work by Pilipczuk and Wrochna [22] shows that there is a tight relationship between the complexity of 3-Coloring on graphs of treedepth, pathwidth, or treewidth $s(n)$ and problems that can be solved by TMs with adequate resources depending on $s(n)$.

2.2 From classical to parameterized

In this paper we introduce the class XALP $=\mathrm{NAuxPDA}[f\text{poly}, f\log]$. Following [8], we use the name XNLP for the class $\mathrm{N}[f\text{poly}, f\log]$; $f$poly is shorthand notation for $f(k)n^{O(1)}$ for some computable function f, and $f\log$ shorthand notation for $f(k)\log n$.

The crucial difference between the existing classical results and our results is that we consider parameterized complexity classes. These classes are closed under parameterized reductions, i.e. reductions where the parameter of the reduced instance must be bounded by the parameter of the initial instance. In our context, we have an additional technicality due to the relationship between time and space constraints. While a logspace reduction is also a polynomial time reduction, a reduction using $f(k)\log n$ space (XL) could use up to

$n^{f(k)}$ time (XP). XNLP and XALP are closed under *pl-reductions* where the space bound is $f(k) + O(\log n)$ (which implies FPT time), and under *ptl-reductions* running in $f(k)n^{O(1)}$ time *and* $f(k)\log n$ space.

We now give formal definitions.

A *parameterized reduction* from a parameterized problem $Q_1 \subseteq \Sigma_1^* \times \mathbb{N}$ to a parameterized problem $Q_2 \subseteq \Sigma_2^* \times \mathbb{N}$ is a function $f \colon \Sigma_1^* \times \mathbb{N} \to \Sigma_2^* \times \mathbb{N}$ such that the following holds.

1. For all $(x, k) \in \Sigma_1^* \times \mathbb{N}$, $(x, k) \in Q_1$ if and only if $f((x, k)) \in Q_2$.
2. There is a computable function g such that for all $(x, k) \in \Sigma_1^* \times \mathbb{N}$, if $f((x, k)) = (y, k')$, then $k' \leq g(k)$.

If there is an 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, then the reduction is a *parameterized logspace reduction* or *pl-reduction*.

If there is an algorithm that computes $f((x, k))$ in time $g(k)n^{O(1)}$ and space $O(h(k)\log n)$, with g, h computable functions and $n = |x|$ the number of bits to denote x, then the reduction is a *parameterized tractable logspace reduction* or *ptl-reduction*.

3 Equivalent characterisations of XALP

In this section, we give a number of equivalent characterisations of XALP.

▶ **Theorem 1.** *The following parameterized complexity classes are all equal.*

1. *NAuxPDA[f poly, f log], the class of parameterized decision problems for which instances of size n with parameter k can be solved by a non-deterministic Turing machine with $f(k)\log n$ memory in $f(k)n^{O(1)}$ time when given a stack, for some computable function f.*
2. *The class of parameterized decision problems for which instances of size n with parameter k can be solved by an alternating Turing machine with $f(k)\log n$ memory whose computation tree is a binary tree on $f(k)n^{O(1)}$ nodes, for some computable function f.*
3. *The class of parameterized decision problems for which instances of size n with parameter k can be solved by an alternating Turing machine with $f(k)\log n$ memory whose computation tree is obtained from a binary tree of depth $O(\log n) + f(k)$ by subdividing each edge $f(k)n^{O(1)}$ times, for some computable function f.*
4. *The class of parameterized decision problems for which instances of size n with parameter k can be solved by an alternating Turing machine with $f(k)\log n$ memory, for which the computation tree has size $f(k)n^{O(1)}$ and uses $O(\log n) + f(k)$ co-non-deterministic steps per computation path, for some computable function f.*

Proof. The proof is similar to the equivalence proofs for the classical analogues, and added for convenience of the reader. We prove the theorem by proving the series of inclusions $1 \subseteq 2 \subseteq 3 \subseteq 4 \subseteq 1$.

$1 \subseteq 2$. Consider a problem that can be solved by a non-deterministic Turing Machine T with a stack and $f(k)\log n$ memory in $f(k)n^{O(1)}$ time. We will simulate T using an alternating Turing machine T'.

We place three further assumptions on T, which can be implemented by changing the function f slightly if needed.

- The Turing machine T has two counters. One keeps track of the height of the stack, and the other keeps track of the number of computation steps. A single computation step may involve several operations; we just need that the running time is polynomially bounded in the number of steps.

- We assume that T only halts with acceptance when the stack is empty. (Otherwise, do not yet accept, but pop the stack using the counter that tells the height of the stack, until the stack is empty.)
- Each pop operation performed by T is a deterministic step. This can be done by adding an extra state to T and splitting a non-deterministic step into a non-deterministic step and a deterministic step if needed.

We define a *configuration* as a tuple which includes the state of T, the value of the two pointers and the content of the memory. In particular, this does not contain the contents of the stack and so a configuration can be stored using $O(f(k)\log n)$ bits. (Note that the value of both pointers is bounded by $f(k)n^{O(1)}$.)

We will build a subroutine $A(c_1, c_2)$ which works as follows.

- The input c_1, c_2 consists of two configurations with the same stack height.
- The output is whether T has an accepting run from c_1 to c_2 without popping the top element from the stack in c_1; the run may pop elements that have yet to get pushed.

We write Apply(c, POP(s)) for the configuration that is obtained when we perform a pop operation in configuration c and obtain s from the stack. This is only defined if T can do a pop operation in configuration c (e.g. it needs to contain something on the stack). We define the configuration Apply(c, PUSH(s)) in a similar manner, where this time s gets pushed onto the stack.

We let T' simulate T starting from configuration c_s as follows. Our alternating Turing machine T' will start with the following non-deterministic step: guess the c_a configuration that accepts at the end of the run. It then performs the subroutine $A(c_s, c_a)$.

We implement $A(c_s, c_a)$ as follows. A deterministic or non-deterministic step of T is carried out as usual.

If T is in some configuration c and wants to push s to the stack, then let $c' =$ Apply(c, PUSH(s)) and let T' perform a non-deterministic step that guesses a configuration c_2' with the same stack height as c' for which the next step is to pop (and the number of remaining computation steps is plausible). Let $c_2 =$ Apply(c_2', POP(s)). We make T' do a co-non-deterministic step consisting of two branches:

- T' performs the subroutine $A(c', c_2')$.
- T' performs the subroutine $A(c_2', c_a)$.

We ensure that in configuration c_2', the number of steps taken is larger than in configuration c'. This ensures that T' will terminate.

Since a configuration can be stored using $O(f(k)\log n)$ and T' always stores at most a bounded number of configurations, T' requires only $O(f(k)\log n)$ bits of memory. The computation tree for T' is binary. The total number of nodes of the computation tree of T' is $f(k)n^{O(1)}$ since each computation step of T appears at most once in the tree (informally: our co-non-deterministic steps split up the computation path of T into two disjoint parts), and we have added at most a constant number of steps per step of T. To see this, the computation tree of T' may split a computation path $c \to_{\text{push}} c' \to \cdots \to c_2' \to_{\text{pop}} c_2 \to \cdots \to c_a$ of T into two parts: one branch will simulate $c' \to \cdots \to c_2'$ and the other branch will simulate $c_2 \to \cdots \to c_a$. At most a constant number of additional nodes (e.g. the node which takes the co-non-deterministic step) are added to facilitate this. Importantly, the configurations implicitly stored a number of remaining computation steps, and so T' can calculate from c', c_2' how many steps T is supposed to take to move between c' and c_2'.

2 $\subseteq$ 3. The intuition behind this proof is to use that any n-vertex tree has a tree decomposition of bounded treewidth of depth $O(\log n)$.

Let A be an alternating Turing machine for some parameterized problem with a computation tree of size $f(k)n^{O(1)}$ and $f(k)\log n$ bits of memory.

We build an alternating Turing machine B that simulates A for which the computation tree is a binary tree which uses $O(f(k)+\log n)$ co-non-deterministic steps per computation branch and $O(f(k)\log n)$ memory. We can after that ensure that there are $f(k)n^{O(1)}$ steps between any two co-non-deterministic steps by adding 'idle' steps if needed.

We ensure that B always has *advice* in memory: 1 configuration for which A accepts. In particular, if c' is the configuration stored as advice when A is in configuration c with a bound of n steps, then B checks if A can get from c to c' within n steps.

We also maintain a counter for the number of remaining steps: the number of nodes that are left in the computation tree of A, when rooted at the current configuration c not counting the node of c itself. In particular, the counter is 0 if c is supposed to be a leaf.

We let B simulate A as follows. Firstly, if no advice is in memory, it makes a non-deterministic step to guess a configuration as advice.

Suppose that A is in configuration c with n_0 steps left. We check the following in order. If c equals the advice, then we accept. If $n_0 \leq 0$, then we reject. If the next step of A is non-deterministic or deterministic step, then we perform the same step. The interesting things happen when A is about to perform a co-non-deterministic step starting from c with n_0 steps left. If $n_0 \leq 1$, then we reject: there is no space for such a step. Otherwise, we guess $n_1, n_2 \geq 0$ such that $n_1 + n_2 = n_0 - 2$, and children c_1, c_2 of c in the computation tree of A. Renumbering if needed, we may assume that the advice c' is supposed to appear in the subtree of c_1. We also guess an advice c_2' for c_2 We create a co-non-deterministic step with two branches, one for the computation starting from c_1 with n_1 steps and the other from c_2 with n_2. We describe how we continue the computation starting from c_1; the case in which c_2 is analogous.

Recall that some configuration c' has been stored as advice. We want to ensure that the advice is limited to one configuration. First, we non-deterministically guess a configuration c''. We non-deterministically guess whether c'' is an ancestor of c'. We perform different computation depending on the outcome.

- Suppose that we guessed that c'' is an ancestor of c'. We guess integers $\frac{1}{3}n_1 \leq a, b \leq \frac{2}{3}n_1$ with $a + b = n_1$. We do a co-non-deterministic step: one branch starts in c_1 with c' as advice and a steps, the other branch starts in c' with c'' as advice and b steps.
- Suppose that c'' is not an ancestor of c'. We guess a configuration ℓ, corresponding to the least common ancestor of c' and c'' in the computation tree. We guess integers $0 \leq a, b, a', b' \leq \frac{2}{3}n_1$ with $a + b + a' + b' = n_1$. We perform a co-non-deterministic branch to obtain four subbranches: starting in c with ℓ as advice and a steps, ℓ with c' as advice and b steps, starting in ℓ with c'' as advice and a' steps and starting in c'' with no advice and b' steps.

In order to turn our computation tree into a binary tree, we may choose to split the single co-non-deterministic step into two steps.

Since at any point, we store at most a constant number of configurations, this can be performed using $O(f(k)\log n)$ bits in memory.

It remains to show that B performs $O(\log n + f(k))$ co-non-deterministic steps per computation path. The computation of B starts with a counter for the number of steps which is at most $f(k)n^{O(1)}$; every time B performs a co-non-deterministic step, this counter is multiplied by a factor of at most $\frac{2}{3}$. The claim now follows from the fact that $\log(f(k)n^{O(1)}) = O(\log n + \log f(k))$.

3 $\subseteq$ 4. Let T be an alternating Turing machine using $f(k)\log n$ memory whose computation fits in a tree obtained from a binary tree of depth d by subdividing each edge $f(k)n^{O(1)}$ times. Then T uses $f(k)n^{O(1)}$ time (with possibly a different constant in the $O(1)$-term) and performs at most d co-non-deterministic steps per computation path. Hence this inclusion is immediate.

4 $\subseteq$ 1. We may simulate the alternating Turing machine using a non-deterministic Turing machine stack as follows. Each time we wish to do a co-non-deterministic branch, we put the current configuration c onto our stack and continue to the left-child of c. Once we have reached an accepting state, we pop an element c of the stack and next continue to the right child of c. The total computation time is bounded by the number of nodes in the computation tree and the memory requirement does not increase by more than a constant factor. (Note that in particular, our stack will never contain more than $\log n + f(k)$ elements.) ◀

Already in the classical setting, it is expected that NL $\subsetneq$ A[poly treesize, log space]. We stress the fact that this would imply XNLP $\subsetneq$ XALP, since we can always ignore the parameter. It was indeed noted in [3, Corollary 3.13] that the assumption NL $\subsetneq$ A[poly treesize, log space] separates the complexity of SAT instances of logarithmic pathwidth from SAT instances of logarithmic treewidth. Allender et al. [3] formulates this result in terms of SAC^1 instead of the equivalent A[poly treesize, log space]. We expect that a parameterized analogue of SAC can be added to the equivalent characterization above, but decided to not pursue this here. The definition of such a circuit class requires a notion of 'uniformity' that ensures that the circuits have a 'small description', which makes it more technical.

4 XALP-completeness for a tree-chained variant of Multicolor Clique

Our first XALP-complete problem is a 'tree' variant of the well-known Multicolor Clique problem.

Tree-Chained Multicolor Clique
Input: A binary tree $T = (I, F)$, an integer k, and for each $i \in I$, a collection of k pairwise disjoint sets of vertices $V_{i,1}, \ldots, V_{i,k}$, and a graph G with vertex set $V = \bigcup_{i \in I, j \in [1,k]} V_{i,j}$.
Parameter: k.
Question: Is there a set of vertices $W \subseteq V$ such that W contains exactly one vertex from each $V_{i,j}$ ($i \in I, j \in [1, k]$), and for each pair $V_{i,j}, V_{i',j'}$ with $i = i'$ or $ii' \in F$, $j, j' \in [1, k]$, $(i, j) \neq (i', j')$, the vertex in $W \cap V_{i,j}$ is adjacent to the vertex in $W \cap V_{i',j'}$?

This problem is the XALP analogue of the XNLP-complete problem Chained Multicolor Clique, in which the input tree T is a path instead. This change of 'path-like' computations to 'tree-like' computations is typical when going from XNLP to XALP.

For the Tree-Chained Multicolor Independent Set problem, we have a similar input and question except that we ask for the vertex in $W \cap V_{i,j}$ and the vertex in $W \cap V_{i',j'}$ *not* to be adjacent. In both cases, we may assume that edges of the graphs are only between vertices of $V_{i,j}$ and $V_{i',j'}$ with $i = i'$ or $ii' \in F$, $j, j' \in [1, k]$, $(i, j) \neq (i', j')$. We call *tree-chained multicolor clique* (resp. *independent set*) a set of vertices satisfying the respective previous conditions.

The problems above can be seen as binary CSPs by replacing vertex choice by assignment choice.

Membership of these problems in XNLP seems unlikely, since it is difficult to handle the 'branching' of the tree. However, in XALP this is easy to do using the co-non-deterministic steps and indeed the membership follows quickly.

▶ **Lemma 2.** Tree-chained Multicolor Clique *is in XALP.*

Proof. We simply traverse the tree T with an alternating Turing machine that uses a co-non-deterministic step when it has to check two subtrees. When at $i \in I$, the machine first guesses a vertex for each $V_{i,j}$, $j \in [k]$. It then checks that these vertices form a multicolor clique with the vertices chosen for the parent of i. The vertices chosen for the parent can now be forgotten and the machine moves to checking children of i. The machine works in polynomial treesize, and uses only $O(k \log n)$ space to keep the indices of chosen vertices for up to two nodes of T, the current position on T. ◀

We next show that Tree-Chained Multicolor Clique is XALP-hard. We will use the characterization of XALP where the computation tree of the alternating Turing machine is a specific tree (3), which allows us to control when co-non-deterministic steps can take place.

Let $\mathcal{M}$ be an alternating Turing machine with computation tree $T = (I, F)$, let x be its input of size n, and k be the parameter. The plan is to encode the configuration of $\mathcal{M}$ at the step corresponding to node $i \in V(T)$ by the choice of the vertices in $V_{i,1}, \dots, V_{i,k'}$ (for some $k' = f(k)$). The possible transitions of the Turing Machine are then encoded by edges between V_i and $V_{i'}$ for $ii' \in F$, where $V_j = \bigcup_{\ell \in [1,k']} V_{j,\ell}$.

A configuration of $\mathcal{M}$ contains the same elements as in the proof of Theorem 1:

- the current state of $\mathcal{M}$,
- the position of the head on the input tape,
- the working space which is $f(k) \log n$ bits long, and
- the position of the head on the work tape.

We partition the working space in $k' = f(k)$ pieces of $\log n$ consecutive bits, and have a set of vertices $V_{i,j}$ for each. Formally, we have a vertex $v_{q,p,b,w}$ in $V_{i,j}$ for each tuple (q, p, b, w) where q is the state of the machine, p is the position of the head on the input tape, $b \in \{\texttt{after}, \texttt{before}\} \uplus [\log n]$ indicates if the block of the work tape is before or after the head, or its position in the block, and w is the current content of the jth block of the work tape.

The edges between vertices of V_i enforce that possible choices of vertices correspond to valid configurations. There is an edge between $v \in V_{i,j}$ and $w \in V_{i,j+1}$ with corresponding tuples (q, p, b, w) and (q', p', b', w'), if and only if $q = q'$, $p = p'$, and either $b' = b \in \{\texttt{after}, \texttt{before}\}$, or $b \in [\log n]$ and $b' = \texttt{after}$, or $b = \texttt{before}$ and $b' \in [\log n]$.

▶ **Observation 3.** *If $v_1, \dots, v_{k'}$ is path with $v_j \in V_{i,j}$, then at most one of the v_j can encode a block with the work tape head, blocks before the head have $b = \texttt{before}$, blocks after the head have $b = \texttt{after}$, and all blocks encode the same state and position of the input tape head.*

The edges between vertices of V_i and $V_{i'}$ for $ii' \in F$ enforce that the configurations chosen in V_i and $V_{i'}$ encode configurations with a transition from one to the other. There is an edge between $v \in V_{i,j}$ and $w \in V_{i',j}$ with corresponding tuples (q, p, b, w) and (q', p', b', w'), such that $(b, b') \in \{(\texttt{after}, \texttt{after}), (\texttt{before}, \texttt{before}), (\texttt{after}, 1), (\texttt{before}, \log n)\}$ if and only if $w = w'$. There is an edge between $v \in V_{i,j}$ and $w \in V_{i',j}$ with corresponding tuples (q, p, b, w) and (q', p', b', w'), such that $b \in [\log n]$, if and only if, there is a transition of $\mathcal{M}$ from state q to state q' that would write $w'[b]$ when reading $x[p]$ on the input tape and $w[b]$ on the work tape, move the input tape head by $p' - p$ and the work tape by $b' - b$ (where $\texttt{after} = 0$ and $\texttt{before} = 1 + \log n$), and for $\ell \in [\log n] \setminus \{b\}$ $w[\ell] = w'[\ell]$.

▷ Claim 4. If $v_1, \dots, v_{k'}, v'_1, \dots, v'_{k'}$ induce a $2 \times k'$ 'multicolor grid' (i.e. $v_1, \dots, v_{k'}$ is a path with $v_j \in V_{i,j}$, $v'_1, \dots, v'_{k'}$ is a path with $v'_j \in V_{i',j}$, there are edges $v_j v'_j$ for $j \in [k']$, $ii' \in F$, and $v'_1, \ldots, v'_{k'}$ encodes a valid configuration), then $v_1, \ldots, v_{k'}$ encodes a valid configuration that can reach the configuration encoded by $v'_1, \ldots, v'_{k'}$ using one transition of $\mathcal{M}$.

Proof. This follows easily from the construction but we still detail why this is sufficient when the work tape head moves to a different block.

We consider the case when the head moves to the block before it. That is we consider the case where v'_j encodes $b' = \log n$ and v_j encodes $b = \texttt{before}$. First, note that there is an edge from $v_j v'_j$ allowing this. We use Observation 3 and conclude that v'_{j+1} (if it exists) must encode head position `after` for its block. The edge $v_{j+1}v'_{j+1}$ then enforces that v_{j+1} encodes head position 1 but it can also exist only if there is a transition of $\mathcal{M}$ that moves the work tape head to the previous block and the written character at the beginning of the block encoded by v'_{j+1} corresponds to such transition. Moving to the next block is a symmetric case. ◀

We have further constraints on the vertices placed in each $V_{i,j}$ based on what i is in T.

- If i is in a leaf of T, then we only have vertices with a corresponding tuple (q, p, b, w) with q an accepting state.
- If i is in a 'branching' vertex of T (i.e. i has two children), then we only have vertices with a corresponding tuple (q, p, b, w) with q a universal state.
- If i is the root, then only vertices corresponding to the initial configuration are allowed.
- Otherwise, we only have vertices with tuples encoding an existential state.

Furthermore, we have to make sure that when branching we take care of the two distinct transitions. We actually assume that T has an order on children for vertices with two children. Then for the edge of T to the first (resp. second) child, we only allow the first (resp. second) transition from the configuration of the parent (which must have a universal state).

We now complete the graph with edges that do not enforce constraints so that we may find a multicolor clique instead of only a $2 \times k'$ multicolor grid. For every $i \in I$, and $j, j' \in [k']$ such that $|j - j'| > 1$, we add all edges between $V_{i,j}$ and $V_{i,j'}$. For every $ii' \in F$, and $j, j' \in [k']$ such that $j \neq j'$, we add all edges between $V_{i,j}$ and $V_{i',j'}$. It should be clear that to find a multicolor clique for some edge ii' after adding these edges is equivalent to finding a 'multicolor grid' before they were added[1].

▷ Claim 5. The constructed graph admits a tree-chained multicolor clique, if and only if, there is an accepting run for $\mathcal{M}$ with input x and computation tree T.

Proof. The statement follows from a straight-forward induction on T showing that for each configuration C of $\mathcal{M}$ that can be encoded by the construction at $i \in I$, its encoding can be extended to a tree-chained multicolor clique of the subtree of T rooted at i, if and only if there is an accepting run of $\mathcal{M}$ from C with as computation tree the subtree of T rooted at i. ◀

Each $V_{i,j}$ has $O(|Q|n^2 \log n)$ vertices (for Q the set of states). Edges are only between $V_{i,j}$ and $V_{i',j'}$ such that $ii' \in F$ or $i = i'$. We conclude that there are $g(k)n^{O(1)}$ vertices and edges in the constructed graph per vertex of T, which is itself of size $h(k)n^{O(1)}$ so the constructed instance has size $g(k)h(k)n^{O(1)}$, for g, h computable functions. The construction can even be performed using only $g'(k) + O(\log(n))$ space for some computable function g'. Note also

[1] Asking for these multicolor grids for each edge of the tree instead of multicolor cliques also leads to an XALP-complete problem but we do not use this problem for further reductions. It could however be used as a starting point for new reductions.

that $k' = f(k)$: the new parameter is bounded by a function of the initial parameter. This shows that our reduction is a parameterized pl-reduction, and we conclude XALP-hardness. Combined with Lemma 2, we proved the following result.

▶ **Theorem 6.** TREE-CHAINED MULTICOLOR CLIQUE *is XALP-complete.*

One may easily modify this to the case where each color class has the same size, by adding isolated vertices.

By taking the complement of the graph, we directly obtain the following result.

▶ **Corollary 7.** TREE-CHAINED MULTICOLOR INDEPENDENT SET *is XALP-complete.*

5 More XALP-complete problems

In this section, we prove a collection of problems on graphs, given with a tree-structure, to be complete for the class XALP. The proofs are of different types: in some cases, the proofs are new, in some cases, reformulations of existing proofs from the literature, and in some cases, it suffices to observe that an existing transformation from the literature keeps the width-parameter at hand bounded.

5.1 List coloring

The problems LIST COLORING and PRE-COLORING EXTENSION with pathwidth as parameter are XNLP-complete [8]. A simple proof shows XALP-completeness with treewidth as parameter. Jansen and Scheffler [20] showed that these problem are in XP, and Fellows et al. [17] showed $W[1]$-hardness.

▶ **Theorem 8.** LIST COLORING *and* PRE-COLORING EXTENSION *are XALP-complete with treewidth as parameter.*

Proof. Membership follows as usual. The color of (uncolored) vertices is non-deterministically chosen when they are introduced. We maintain the color of vertices of the current bag in the working space. We use co-non-deterministic steps when the tree decomposition branches. We check that introduced edges do not contradict the coloring being proper. This uses $O(k \log n)$ space, and runs in polynomial total time.

We first show XALP-hardness of LIST COLORING. We reduce from TREE-CHAINED MULTICOLOR INDEPENDENT SET. Suppose we have an instance of this problem. The set of colors equals the set of vertices V. For each class V_{ij}, $i \in I$, $j \in [1,k]$, we take a vertex v_{ij} with set of colors V_{ij}.

For each pair of 'incident classes' V_{ij}, $V_{i'j'}$ with $i = i'$ or ii' an edge in F, $ij \neq i'j'$, and each edge $vw \in E \cap V_{ij} \times V_{i'j'}$, we add a new vertex with set of colors $\{v, w\}$, which is incident to v_{ij} and $v_{i'j'}$. Let H be the resulting graph.

Now, H has a list coloring, if and only if there is a tree-chained multicolor independent set in G. The transformation of solutions is straightforward: the chosen colors for v_{ij} are equal to the chosen vertices from V_{ij}. If we choose two adjacent vertices in incident classes, then we do not have a color available for a new vertex; if we have a tree-chained independent set, then each new vertex has at least one available color.

H has treewidth at most $2k-1$: take a root of T, for each $i \in I$, let X_i consist of all v_{ij} and $v_{i'j}$, i' the parent of i, $j \in [1,k]$. Now, for each new vertex, we add a bag containing this vertex and its two neighbors, making it incident to a bag that contains its neighbors.

The standard reduction from Pre-coloring Extension to List Coloring that adds for each forbidden color c of a vertex v a new neighbor to v precolored with c does not increase the treewidth, which shows XALP-hardness for Pre-coloring Extension with treewidth as parameter. ◀

5.2 Tree variants of Weighted Satisfiability

From Tree-Chained Multicolor Independent Set, we can show XALP-completeness of tree variants of what in [8] was called Chained Weighted CNF-Satisfiability and its variants (which in turn are analogues of Weighted CNF-Satisfiability, see e.g. [14, 15]).

Tree-Chained Weighted CNF-Satisfiability
Input: A tree $T = (I, F)$, sets of variables $(X_i)_{i \in I}$, and clauses $C_1, \dots, C_m$, each with either only variables of X_i for some $i \in I$, or only variables of X_i and X_j for some $ij \in F$.
Parameter: k.
Question: Is there an assignment of at most k variables in each X_i that satisfies all clauses?

Positive Partitioned Tree-Chained Weighted CNF-Satisfiability
Input: A tree $T = (I, F)$, sets of variables $(X_i)_{i \in I}$, and clauses of positive literals $C_1, \dots, C_m$, each with either only variables of X_i for some $i \in I$, or only variables of X_i and X_j for some $ij \in F$. Each X_i is partitioned into $X_{i,1}, \dots, X_{i,k}$.
Parameter: k.
Question: Is there an assignment of exactly one variable in each $X_{i,j}$ that satisfies all clauses?

Negative Partitioned Tree-Chained Weighted CNF-Satisfiability
Input: A tree $T = (I, F)$, sets of variables $(X_i)_{i \in I}$, and clauses of negative literals $C_1, \dots, C_m$, each with either only variables of X_i for some $i \in I$, or only variables of X_i and X_j for some $ij \in F$. Each X_i is partitioned into $X_{i,1}, \dots, X_{i,k}$.
Parameter: k.
Question: Is there an assignment of exactly one variable in each $X_{i,j}$ that satisfies all clauses?

▶ **Theorem 9.** *Positive Partitioned Tree-Chained Weighted CNF-Satisfiability, Negative Partitioned Tree-Chained Weighted CNF-Satisfiability, and Tree-Chained Weighted CNF-Satisfiability are XALP-complete.*

Proof. We first show membership for Tree-Chained Weighted CNF-Satisfiability, which implies membership for the more structured versions. We simply follow the tree shape of our instance by branching co-non-deterministically when the tree branches. We keep the indices of the $2k$ variables chosen non-deterministically for the 'local' clauses in the working space. We then check that said clauses are satisfied.

We first show hardness for Negative Partitioned Tree-Chained Weighted CNF-Satisfiability by reducing from Tree-Chained Multicolor Independent Set. For each vertex v, we have a Boolean variable x_v. We denote by $X_{i,j}$ the set of variables $\{x_v : v \in V_{i,j}\}$, and by X_i the set of variables $\{x_v : v \in V_i\}$. This preserves the partition properties. For each edge uv, we add the clause $\neg x_u \vee \neg x_v$.

▶ **Observation 10.** *U is multicolor independent set if and only if $\{x_u : u \in U\}$ is a satisfying assignment.*

To reduce to Positive Partitioned Tree-Chained Weighted CNF-Satisfiability, we simply replace negative literals $\neg x_v$ for $x_v \in X_{i,j}$ by a disjunction of positive literals $\vee_{y \in X_{i,j} \setminus \{x_v\}} y$. This works because, due to the partition constraint, a variable $x \in X_{i,j}$ is assigned $\perp$ if and only if another variable $y \in X_{i,j} \setminus \{x\}$ is assigned $\top$.

To reduce to Tree-Chained Weighted CNF-Satisfiability, we simply express the partition constraints using clauses. For each $X_{i,j}$, we add the clauses $\vee_{y \in X_{i,j}} y$, and for each pair $\{x, y\} \subseteq X_{i,j}$ the clause $\neg x \vee \neg y$. This enforces that we pick at least one variable, and at most one variable, for each $X_{i,j}$. ◀

5.3 Logarithmic Treewidth

Although XALP-complete problems are in XP and not in FPT, there is a link between XALP and single exponential FPT algorithms on tree decompositions. Indeed, by considering instances with treewidth $k \log n$, where k is the parameter, the single exponential FPT algorithm becomes an XP algorithm. We call this parameter logarithmic treewidth.

Independent Set parameterized by logarithmic treewidth
Input: A graph $G = (V, E)$, with a given tree decomposition of width at most $k \log |V|$, and an integer W.
Parameter: k.
Question: Is there an independent set of G of size at least W?

▶ **Theorem 11.** Independent Set *with logarithmic treewidth as parameter is XALP-complete.*

Proof. We start with membership which follows from the usual dynamic programming on the tree decomposition. We maintain for each vertex v in the current bag whether v is in the independent set or not. When introducing a vertex v, we non-deterministically decide if v is put in the independent set or not. We reject if an edge is introduced between two vertices of the independent set. We make a co-non-deterministic step whenever the tree decomposition is branching. Since we only need one bit of information per vertex in the bag, this requires only $O(k \log n)$ working space, as for the running time we simply do a traversal of the tree decomposition which is only polynomial treesize.

We show hardness by reducing from Positive Partitioned Tree-Chained Weighted CNF-Satisfiability. We can simply reuse the construction from [8] and note that the constructed graph has bounded logarithmic treewidth instead of logarithmic pathwidth because we reduced from the tree-chained SAT variant instead of the chained SAT variant. We describe the gadgets for completeness. First, the SAT instance is slightly adjusted for technical reasons. For each $X_{i,j}$, we add a clause containing exactly its initial variables. This makes sure that the encoding of the chosen variable is valid. We assume the variables in each $X_{i,j}$ to be indexed starting from 0.

Variable gadget. For each $X_{i,j}$, let $t_{i,j} = \lceil \log_2 |X_{i,j}| \rceil$. We add edges $\widehat{0}_\alpha \widehat{1}_\alpha$, $\alpha \in [1, t_{i,j}]$.

Clause gadget. For each clause with ℓ literals, we assume ℓ to be even by adding a dummy literal if necessary. We add paths $p_0, \ldots, p_{\ell+1}$, and $p'_1, \ldots, p'_\ell$. For $i \in [1, \ell]$, we add the edge $p_i p'_i$. We then add vertex v_i for $i \in [1, \ell]$, which represents the ith literal of the clause. Let $b_1 \ldots b_{t_{i',j'}}$ be the binary representation of the index of the corresponding variable of $X_{i',j'}$. Then v_i is adjacent to p_i, p'_i and the vertices $\widehat{1-b_\alpha}$ for $\alpha \in [1, t_{i,j}]$. For the dummy literal, there is no vertex v_i.

The clause gadget has an independent set of size $\ell + 2$ if and only if it contains a vertex v_i. When the variable gadgets have one vertex in the independent set on each edge, a vertex v_i of a clause can be added to the independent set only if the independent set contains exactly the vertices of the variable gadget that give the binary representation of the variable corresponding to v_i.

Hence, the SAT instance is satisfiable if and only if there is an independent set of size $\sum_{i,j} t_{i,j} + \sum_i 2 + \ell_i$ in our construction. ◀

▶ **Corollary 12.** *The following problems are XALP-complete with logarithmic treewidth as parameter:* VERTEX COVER*,* RED-BLUE DOMINATING SET*,* DOMINATING SET*.*

Proof. The result for VERTEX COVER follows directly from Theorem 11 and the well known fact that a graph with n vertices has a vertex cover of size at most L, iff it has an independent set of size at least $n - L$. Viewing VERTEX COVER as a special case of RED-BLUE DOMINATING SET gives the following graph: subdivide all edges of G, and ask if a set of K original (blue) vertices dominates all new (red) subdivision vertices; as the subdivision step does not increase the treewidth, XALP-hardness of RED-BLUE DOMINATING SET with treewidth as parameter follows. To obtain XALP-hardness of DOMINATING SET, add to the instance G' of RED-BLUE DOMINATING SET, two new vertices x_0 and x_1 and edges from x_1 to x_0 and all blue vertices; the treewidth increases by at most one, and the minimum size of a dominating set in the new graph is exactly one larger than the minimum size of a red-blue dominating set in G'. Membership in XALP is shown similar as in the proof of Theorem 11. ◀

5.4 Other problems

Several XALP-hardness proofs follow from known reductions. Membership is usually easy to prove, by observing that the known XP-algorithms can be turned into XALP-membership by guessing table entries, and using the stack to store the information for a left child when processing a right subtree.

▶ **Corollary 13.** *The following problems are XALP-complete:*
1. CHOSEN MAXIMUM OUTDEGREE*,* CIRCULATING ORIENTATION*,* MINIMUM MAXIMUM OUTDEGREE*,* OUTDEGREE RESTRICTED ORIENTATION*, and* UNDIRECTED FLOW WITH LOWER BOUNDS*, with the treewidth as parameter.*
2. *MAX CUT and MAXIMUM REGULAR INDUCED SUBGRAPH with cliquewidth as parameter.*

Proof. (1): The reductions given in [4] and [24] can be used; one easily observes that these reductions keep the treewidth of the constructed instance bounded by a function of the treewidth of the original instance (often, a small additive constant is added.)

(2): The reductions given in [7] can be reused with minimal changes, only the bound on linear clique-width becomes a bound on clique-width because of the 'tree-shape' of the instance to reduce. ◀

CHOSEN MAXIMUM OUTDEGREE, CIRCULATING ORIENTATION, MINIMUM MAXIMUM OUTDEGREE, OUTDEGREE RESTRICTED ORIENTATION, and UNDIRECTED FLOW WITH LOWER BOUNDS, together with ALL-OR-NOTHING FLOW were shown to be XNLP-complete with pathwidth as parameter in [4]. Gima et al. [19] showed that MINIMUM MAXIMUM OUTDEGREE with vertex cover as parameter is $W[1]$-hard. For related results, see also [24].

In [6], it is shown that Tree-Partition-Width and Domino Treewidth are XALP-complete, which can be seen as an analog to Bandwidth being XNLP-complete.

6 Conclusions

We expect many (but not all) problems that are (W[1]-)hard and in XP for treewidth as parameter to be XALP-complete; our paper gives good starting points for such proofs. Let us give an explicit example. The Pebble Game Problem [15, 21] parameterized by the number of pebbles is complete for XP, which is equal to XAL=A$[\infty, f \log]$. The problem corresponds to deciding whether there is a winning strategy in an adversarial two-player game with k pebbles on a graph where the possible moves depend on the positions of all pebbles. We can expect variants with at most $f(k) + O(\log n)$ moves to be complete for XALP.

Completeness proofs give a relatively precise complexity classification of problems. In particular, XALP-hardness proofs indicate that we do not expect a deterministic algorithm to use less than XP space if it runs in XP time. Indeed the inclusion of XNLP in XALP is believed to be strict, and already for XNLP-hard problems we have the following conjecture.

▶ **Conjecture 14** (Slice-wise Polynomial Space Conjecture [22]). *No XNLP-hard problem has an algorithm that runs in $n^{f(k)}$ time and $f(k)n^c$ space, with f a computable function, k the parameter, n the input size, and c a constant.*

While XNLP and XALP give a relatively simple framework to classify problems in terms of simultaneous bound on space and time, the parameter is allowed to blow up along the reduction chain. One may want to mimic the fine grained time complexity results based on the (Strong) Exponential Time Hypothesis. In this direction, one could assume that Savitch's theorem is optimal as was done in [9].

Since XNLP is above the W-hierarchy, it could be interesting to study the relationship of XALP with some other hierarchies like the A-hierarchy and the AW-hierarchy. It is also unclear where to place List-Coloring parameterized by tree-partition-width[2]. It was shown to be in XL and W[1]-hard [5] but neither look like good candidates for completeness.

References

1 Karl A. Abrahamson, Rodney G. Downey, and Michael R. Fellows. Fixed-parameter tractability and completeness IV: On completeness for $W[P]$ and PSPACE analogues. *Ann. Pure Appl. Log.*, 73:235–276, 1995. doi:10.1016/0168-0072(94)00034-Z.

2 Michael Alekhnovich and Alexander A. Razborov. Satisfiability, branch-width and tseitin tautologies. In *Proceedings of the 43rd Symposium on Foundations of Computer Science*, FOCS '02, pages 593–603, USA, 2002. IEEE Computer Society.

3 Eric Allender, Shiteng Chen, Tiancheng Lou, Periklis A. Papakonstantinou, and Bangsheng Tang. Width-parametrized SAT: time–space tradeoffs. *Theory of Computing*, 10:297–339, 2014. doi:10.4086/toc.2014.v010a012.

4 Hans L. Bodlaender, Gunther Cornelissen, and Marieke van der Wegen. Problems hard for treewidth but easy for stable gonality. *arXiv*, abs/2202.06838, 2022. Extended abstract

[2] A tree-partition of a graph G is a decomposition of $V(G)$ into bags $(B_i)_{i \in V(T)}$, where T is a tree, such that $uv \in V(G)$ implies that the bags of u and v are the same or adjacent in T. The width is the size of the largest bag, and the tree-partition-width of G is found by taking the minimum width over all tree-partitions of G.

to appear in Proceedings WG 2022. URL: https://arxiv.org/abs/2202.06838, arXiv:2202.06838.

5 Hans L. Bodlaender, Carla Groenland, and Hugo Jacob. List colouring trees in logarithmic space. *arXiv*, abs/2206.09750, 2022. URL: https://arxiv.org/abs/2206.09750.

6 Hans L. Bodlaender, Carla Groenland, and Hugo Jacob. On the parameterized complexity of computing tree-partitions. *arXiv*, abs/2206.11832, 2022. URL: https://arxiv.org/abs/2206.11832.

7 Hans L. Bodlaender, Carla Groenland, and Hugo Jacob. XNLP-completeness for parameterized problems on graphs with a linear structure. *arXiv*, abs/2201.13119, 2022. URL: https://arxiv.org/abs/2201.13119, arXiv:2201.13119.

8 Hans L. Bodlaender, Carla Groenland, Jesper Nederlof, and Céline M. F. Swennenhuis. Parameterized problems complete for nondeterministic FPT time and logarithmic space. In *Proceedings 62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021*, pages 193–204, 2021. doi:10.1109/FOCS52979.2021.00027.

9 Yijia Chen, Michael Elberfeld, and Moritz Müller. The parameterized space complexity of model-checking bounded variable first-order logic. *Log. Methods Comput. Sci.*, 15(3), 2019. doi:10.23638/LMCS-15(3:31)2019.

10 Stephen A. Cook. The complexity of theorem-proving procedures. In Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman, editors, *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, STOC 1971*, pages 151–158. ACM, 1971. doi:10.1145/800157.805047.

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

12 Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM J. Comput.*, 24(4):873–921, 1995. doi:10.1137/S0097539792228228.

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

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

15 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.

16 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.

17 Michael R. Fellows, Fedor V. Fomin, Daniel Lokshtanov, Frances A. Rosamond, Saket Saurabh, Stefan Szeider, and Carsten Thomassen. On the complexity of some colorful problems parameterized by treewidth. *Inf. Comput.*, 209(2):143–153, 2011. doi:10.1016/j.ic.2010.11.026.

18 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer, 2006. doi:10.1007/3-540-29953-X.

19 Tatsuya Gima, Tesshu Hanaka, Masashi Kiyomi, Yasuaki Kobayashi, and Yota Otachi. Exploring the gap between treedepth and vertex cover through vertex integrity. *Theoretical Computer Science*, 918:60–76, 2022. doi:10.1016/j.tcs.2022.03.021.

20 Klaus Jansen and Petra Scheffler. Generalized coloring for tree-like graphs. *Discrete Applied Mathematics*, 75(2):135–155, 1997. doi:10.1016/S0166-218X(96)00085-6.

21 Takumi Kasai, Akeo Adachi, and Shigeki Iwata. Classes of pebble games and complete problems. *SIAM Journal on Computing*, 8(4):574–586, 1979. doi:10.1137/0208046.

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

23 Walter L. Ruzzo. Tree-size bounded alternation. *Journal of Computer and System Sciences*, 21(2):218–235, 1980. URL: https://www.sciencedirect.com/science/article/pii/0022000080900367, doi:https://doi.org/10.1016/0022-0000(80)90036-7.

24 Stefan Szeider. Not so easy problems for tree decomposable graphs. In *Advances in Discrete Mathematics and Applications: Mysore, 2008*, volume 13 of *Ramanujan Math. Soc. Lect. Notes Ser.*, pages 179–190. Ramanujan Math. Soc., Mysore, 2010. arXiv:1107.1177.

25 H. Venkateswaran. Properties that characterize LOGCFL. *Journal of Computer and System Sciences*, 43(2):380–404, 1991. URL: https://www.sciencedirect.com/science/article/pii/0022000091900206, doi:https://doi.org/10.1016/0022-0000(91)90020-6.