Exploring the class XNLP
ARPE final report

Hugo Jacob

école normale supérieure paris-saclay

Supervised by Carla Groenland and Hans L. Bodlaender

Utrecht University

General context Parameterized complexity provides a refined analysis of the inherent difficulty of computational problems. This is done by introducing parameters of the input in the measure of the algorithms' complexity.

Studied problem We consider the class XNLP of parameterized problems that can be solved by a nondeterministic algorithm with both time bounded by $f(k)n^c$ and space bounded by $f(k)\log(n)$, for f a computable function, k the parameter, n the input size, and c a constant. XNLP-hardness is conjectured to imply that there is no algorithm running in time $n^{f(k)}$ and space $f(k)n^c$. We seek to find more XNLP-complete problems and better understand the boundaries of XNLP.

Contribution We provide XNLP-complete problems parameterized by several 'linear width measures' of the input graph. This allows for dense graph instances, unlike the previously established XNLP-complete problems.

The computational model of XNLP seems to enforce a linear structure, hence we extend the results of XNLP to treelike structures. This is done by defining the class XALP via Alternating Turing Machines (or equivalently Nondeterministic Turing Machines equipped with a stack).

Perspectives The relationship of XALP to existing parameterized hierarchies, and the completeness of some tiling problems remain to be explored.

Hardness results of problems parameterized by treedepth would be interesting although completeness seems unlikely.

1. Introduction

The principal interest of parameterized complexity is to provide an understanding of what makes a problem instance tractable or intractable. The point of parameterized complexity is to add a number (or several) to problem instances, called the parameter, which proves to be relevant to the tractability of the instance. The parameter is used in the analysis of the running time and the space requirements. The parameter could be the size of the sought solution, a hint to the structure of the instance, the number of a particular set of elements that constitute the instance, etc. This is a very natural thing to do when designing and analysing algorithms, but it also turned out to be very fruitful from a complexity point of view. The theoretical aspects of parameterized complexity were introduced by Downey and Fellows in the early 90s.

The concept of NP-hardness undoubtedly provides some insight on whether a problem is hard or not. However, an NP-hard problem may contain relatively few hard instances, with most instances being very easy to solve. As an example of this, you can consider Hamiltonian Cycle, the problem of deciding if a graph has a cycle covering each vertex exactly once. It is not difficult to see that if the instance is not a connected graph (and even if it is not a 2-connected graph), a property that can be checked in linear time, we know that it is a NO-instance. Intuitively, most dense graphs should also be YES-instances (several criteria have been found for the identification of simple instances). This already motivates finding distinctions between instances of NP-hard problems.

While NP-complete problems are reducible to each other in polynomial time, some NP-complete problems are definitely easier than others. Outside of the parameterized complexity world, this is visible with moderately exponential algorithms. Consider the following two NP-complete problems: Hamiltonian Cycle and Vertex Cover, the problem of finding in a given graph a set of vertices that hits all edges. There is no known algorithm running faster than the classical $O^*(2^n)$[1] [HK62] for the former, while the latter admits algorithms running in $O^*(1.221^n)$ [FGK09]. Note that the technique Measure & Conquer that lead to this result, defines some notion of measure of the hardness of a subproblem in order to obtain this improved bound on the running time.

We conclude this list of motivations for parameterized complexity by introducing the notion of kernel. Although the usual way of quantifying hardness of an instance in classical complexity is to consider its size, it is easy to see that you can often make instances that are arbitrarily larger without being harder. A kernel is a "reduced" instance of size bounded by the initial parameter, obtained from an initial instance by applying reduction rules to reduce the instance size. We say that a problem admits a kernel if there is a polynomial time algorithm producing a kernel for any instance of the problem. While admitting a kernel is an interesting notion and turns out to be equivalent to admitting an FPT algorithm (the notion will be introduced later), the question of admitting a polynomial kernel (i.e. the size of the kernel is bounded by a polynomial of the parameter) raises more interest. It is especially interesting to note that there are ways to prove that a problem does not admit a polynomial kernel, unless a common assumption of classical complexity is disproved.

The most important classes of parameterized complexity are FPT and XP. FPT stands for *Fixed Parameter tractable*. FPT is the class of problems that can be solved in time $f(k)\cdot n^{O(1)}$,

[1] The notation O^* hides the polynomial factors.

where n is the instance size, k the parameter, and f a (computable) function. Ideally, we want f to grow as slowly as possible, and having a small polynomial term is also desirable. Note however that, if the problem is not in P, f cannot be polynomial for usual parameters. Furthermore, if the problem is NP-hard, f is expected to be at least exponential. In a general setting, there is no reason to be able to optimise both f and the polynomial term. Most works focus on optimising only one of them. XP is the class of problems that can be solved in time $f(k) \cdot n^{g(k)}$, where n is the instance size, k the parameter, and f, g computable functions. XP is the class of problems that are in P for each fixed value of the parameter, and can be called *slice-wise polynomial*. It should be clear that FPT is contained in XP. In that sense, finding an FPT algorithm instead of an XP algorithm may be considered as an equivalent to finding a polynomial time algorithm in classical complexity. Once again, there are some techniques to show negative results: W[1]-hard problems are believed not to be in FPT (the class W[1] will be introduced in the following section), and problems that are NP-hard for a constant value of the parameter cannot be in XP unless P=NP (the class of such problems is called para-NP).

By making assumptions on the optimal running time of algorithms for SAT, it is possible to prove tight lower bounds on the parameterized complexity of some problems. This line of work is called *fine-grained* parameterized complexity.

For a more detailed introduction to parameterized complexity and parameterized algorithms, consider the textbooks [CFK+15, DF13, FG06].

While complexity classes with simultaneous bounds on the time and space requirements have been studied in the classical setting, this is relatively new to the landscape of parameterized complexity. The goal of the project is to make progress in this direction with the study of the class XNLP, which corresponds to problems that can be solved nondeterministically in FPT time, i.e. $f(k) \cdot n^{O(1)}$, and $g(k) \log n$ space, where n is the instance size, k the parameter, and f, g are computable functions. This class has natural ties to dynamic programming on linear decompositions. XNLP-hard problems are not expected to be solvable faster than in XP time while using XP space.

Since many combinatorial problems can be expressed in terms of graphs (or generalizations), it is quite natural to ask for structures which make problems tractable on graphs. Finding a decomposition of a graph is a way of making the structure explicit. Designing algorithms on graph decompositions shows how to make use of the given structure to solve problems. One of the most successful notions of graph decomposition to date is the tree decomposition. It has applications in the study of graph minors, and has been widely studied regarding algorithmic applications. In particular, a theorem of Bruno Courcelle shows that any problem expressible in Monadic Second Order logic is solvable in FPT time when parameterized by the width of a given a tree decomposition. It turns out that computing such a decomposition can also be done in FPT time, so the assumption that the decomposition is given is not needed.

The design of efficient parameterized algorithms on graph decompositions is an active area of research and XNLP appears to be a relevant tool in this area. It can often be used to show stronger hardness results than what is currently known without too much effort. Current hardness results are usually centered on running time (e.g. W[1]-hardness, NP-hardness), for which XNLP-hardness does not give more information. However, it gives information on the space required for efficient algorithms. While this might seem less important, this is very important when implementing algorithms. You may consider having your algorithm run for 1

hour instead of a 1 second when solving large instances, but if it now also requires thousands of gigabytes of RAM instead of 1 gigabyte, it will be hard to find a computer with such specifications.

2. Preliminaries

2.1. Graph theory terminology

We use standard notations but introduce them for convenience. We mostly consider simple graphs, i.e. undirected and loopless graphs. Formally, a graph is an ordered pair (V, E) where V is the set of *vertices* and E is the set of *edges* which are unordered pairs of vertices called endpoints of the edge. We use notation $V(G)$ and $E(G)$ for a graph G to denote its set of vertices or edges respectively. In the case of multigraphs, E becomes a multiset.

The *neighborhood* of a vertex v in graph $G = (V, E)$ is denoted by $N_G(v)$ and corresponds to the set $\{w \in V \setminus \{v\} \mid vw \in E\}$. This is extended to subsets of vertices as follows. For $U \subseteq V$, $N_G(U) = \left(\bigcup_{v \in U} N_G(v)\right) \setminus U$. The *closed* neighborhoods are denoted by $N_G[v]$ and $N_G[U]$, and defined by $N_G[v] = \{v\} \cup N_G(v)$ and $N_G[U] = U \cup N_G(U)$. The degree of v is the cardinality of its neighborhood. When the considered graph is clear from context, the subscript is dropped.

A graph $H = (V', E')$ is a *subgraph* of $G = (V, E)$ if $V' \subseteq V$, $E' \subseteq E$, and edges of E' contain only vertices of V'. The subgraph of $G = (V, E)$ *induced* by the subset of vertices $U \subseteq V$ is the subgraph (U, E') where E' contains all edges of E with both endpoints in U, it is denoted $G[U]$. The subgraph $G - A$ denotes $(V, E \setminus A)$. We let $G - v$ and $G - U$ denote $G[V \setminus \{v\}]$ and $G[V \setminus U]$ respectively. Given X, Y disjoint subsets of V, we denote by $G[X, Y]$, the bipartite subgraph induced by (X, Y), i.e. the subgraph on vertex set $X \cup Y$, with only edges of G incident to X and Y remaining. We use $E(X, Y) = E(G[X, Y])$ as a shorthand when G is clear from context.

A *minor* H of G is a graph obtained from a subgraph of G by contracting edges (i.e. merging two vertices that were incident to the same edge and removing this edge). A *topological minor* H of G is a graph obtained from a subgraph of G by smoothing vertices of degree 2 (i.e. removing such vertices while adding an edge between their neighbors). An *immersion* H of G is a graph such that there is an injective mapping of vertices of H to vertices of G where the images of vertices adjacent in H are connected by edge-disjoint paths.

A cutvertex is a vertex whose removal increases the number of connected components. A 2-connected component of G is a maximal subset C of vertices such that $G[C] - v$ is connected for all $v \in C$. The maximum degree of a graph G is often denoted $\Delta(G)$. The complete graph on n vertices is K_n, and the complete bipartite graph with n and m vertices on each side is $K_{n,m}$.

2.2. Parameterized complexity

Definition 1. A *parameterized problem* is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a fixed finite alphabet. For an instance $(x, k) \in \Sigma^* \times \mathbb{N}$, k is called the *parameter*. The *size* of the instance is $|x| + k$, as if the parameter was written in unary.

Definition 2. A parameterized problem L is called *fixed-parameter tractable* (FPT) if there exists an algorithm $\mathcal{A}$ (called fixed-parameter algorithm), a computable function $f : \mathbb{N} \to \mathbb{N}$, and a constant c such that, "$(x, k) \in L$" can be decided by $\mathcal{A}$ in time bounded by $f(k) \cdot |(x, k)|^c$. The class of FPT problems is called FPT.

It should be clear that P $\subseteq$ FPT as you can let f be a constant function.

A classical example of parameterized problem would be Vertex Cover parameterized by the solution size. We quickly point out that when the parameter is not explicitly given, it usually means that the parameter is the solution size. Hence, it would be standard to refer to this parameterized problem simply as Vertex Cover. It is well known that Vertex Cover, Independent Set, and Dominating Set are NP-complete problems, and reducing one to other is not difficult in the classical setting.

A straightforward FPT algorithm for Vertex Cover is the following: Consider an edge that is not already covered and try including one endpoint or the other (at least one of them has to be since the edge must be covered). Because we look for a solution of size k, the recursive choices made with this strategy correspond to a binary tree of depth k. Therefore, a naive analysis already gives time bound $O(2^k \cdot |E|)$, where $|E|$ is the number of edges of the input graph. Vertex Cover even has a polynomial kernel.

However, Independent Set and Dominating Set are thought not to be in FPT (this will be discussed in more details).

The following result was mentioned in the introduction:

Theorem 1. A parameterized problem is in FPT if and only if it admits a kernel and it is decidable.

Proof. If the problem is in FPT, then it is decidable. Let $f(k) \cdot |(x, k)|^c$ be the bound on an FPT algorithm for this problem. We can run the first $|(x, k)|^{c+1}$ steps of this algorithm. If it terminates, we can return a trivial YES-instance or NO-instance as a kernel. If it does not terminate then it must be that $f(k) > |(x, k)|$, meaning that we can return the original instance as a kernel.

Conversely, if the problem admits a kernel and is decidable, we can compute a kernel in polynomial time, and then, since the problem is decidable, we can compute the answer with a running time bound depending only on the parameter. The total running time suffices for membership in FPT. □

The equivalence between the two notions showcases that finding a parameter for which a problem becomes FPT means that this parameter is a "good" estimate of the hardness of an instance. Finding good parameters for a problem is a complicated task.

The very successful notion of reduction on which classical complexity is based, is also used to classify parameterized problems. However, the crucial difference is that parameter growth must be controlled.

Definition 3. Given two parameterized problems A, B over the same alphabet, a *parameterized reduction* from A to B is an algorithm $\mathcal{A}$ that, given an instance (x, k) of A, outputs an instance (x', k') of B such that:

- (x, k) is a yes-instance of A if and only if (x', k') is a yes-instance of B
- $k' \leq g(k)$ for some computable function g

FPT is closed under *fpt-reductions*, the parameterized reductions such that $\mathcal{A}$ runs in FPT time. Let us point out that while this notion of reduction with an FPT algorithm is standard, parameterized classes involving space restrictions such as XNLP must use a different restriction on the reduction algorithm. We use *pl-reductions* (where $\mathcal{A}$ should run using $f(k) + O(\log n)$ space), and *ptl-reductions* (where $\mathcal{A}$ should run in FPT time using $f(k) \log n$ space).

Note that the simple reduction from Independent Set to Vertex Cover by taking the complement of the solution is not a parameterized reduction because a solution of size k is mapped to a solution of size $n - k$. On the contrary, Independent Set and Clique can be reduced to one another because the simple reduction consists of considering the complemented graph and preserves the solution size.

We now introduce the W hierarchy which can be thought of as a parameterized equivalent of the polynomial hierarchy.

Definition 4. A *boolean circuit*, is a circuit consisting of boolean input gates, AND, OR, and NOT gates, and a single output gate.

The problem Weighted Circuit Satisfiability asks whether we can satisfy (i.e. the output gate should be 1) a given boolean circuit with only k input gates with value 1. The parameter is k, the bound on the Hamming weight of a satisfying assignment.

If we restrict the problem to circuits having constant depth d and weft t, we obtain the problem Weighted Weft-t-Depth-d Circuit Satisfiability. The weft of a circuit is the maximum number of gates with unbounded inputs on a path from an input gate of the circuit to its output gate. The class of parameterized problems that can be reduced to this problem for some value of d, is denoted by W[t].

W[0] is FPT, and W[i] $\subseteq$ W[j] for all $i \leq j$. The hierarchy is believed to be strict. Independent Set is W[1]-complete, and Dominating Set is W[2]-complete. For $t \geq 2$, Weighted Depth-t SAT, the problem of finding an assignment with at most k true variables for a boolean formula with parenthesis depth at most t, is W[t]-complete.

It is commonly assumed that FPT $\neq$ W[1], hence to prove that a problem is believed not to be in FPT, W[1]-hardness results are given. Note that this assumption is stronger than P $\neq$ NP because W[1] $\subseteq$ NP.

Fine-grained parameterized complexity uses stronger assumptions on the optimal complexity of SAT.

k-SAT **Input:** A CNF formula φ with at most k variables per clause **Parameter:** k. **Question:** Is φ satisfiable ?

Let s_k be the infimum over positive reals ε such that there exists an algorithm running in time $O(2^{\varepsilon n})$ to solve k-SAT, where n is the number of variables.

The **Exponential Time Hypothesis** (ETH) states that $s_3 > 0$. This implies that there is no $2^{o(n)}$ time algorithm for 3-SAT. By reducing k-SAT to some problem, we can conclude that there is no $2^{o(f(n))}$ time algorithm for it under the ETH. The ETH implies FPT $\neq$ W[1], hence P $\neq$ NP.

The **Strong Exponential Time Hypothesis** (SETH) states that $\lim_{k\to+\infty} s_k = 1$. This means that for some arbitrary formula, there is no algorithm significantly better than testing every possible assignment. By reducing k-SAT to some problem with the new parameter bounded independently of the number of clauses, we can prove that a running time of the form $O\left(c^k|x|^{O(1)}\right)$ has optimal c under the SETH. While this assumption is considered less likely by part of the computational complexity community, note that this mainly allows to express neatly the hardness result obtained from a reduction but is not used in the reduction. Furthermore, it indicates that trying to beat the lower bound is at least as hard as disproving the SETH.

2.3. Graph decompositions

One approach to solving hard problems on graphs that have a nice structure, is to first compute a decomposition of the graph and then apply an efficient algorithm on the decomposition. The decompositions that we will consider are well suited to the design of dynamic programming.

Definition 5. A *tree decomposition* of a graph $G = (V, E)$ is a pair $(T, \mathcal{X})$ where T is a tree and $\mathcal{X} = \{X_t\}_{t\in T}$ is a set of *bags* such that :

- bags contain vertices (elements of V)
- for $v \in V$, the bags that contain v form an induced subtree of T (they are connected)
- every vertex $v \in V$ is contained in at least one bag
- for every edge $uv \in E$, there is a bag X_t that contains u and v.

The width of a tree decomposition is defined as $\max_{t\in T}\{|X_t| - 1\}$.

The treewidth of G is the minimal width over all possible tree decompositions of G, we will denote it by $\mathbf{tw}(G)$.

The pathwidth of G is the minimal width over all possible tree decompositions of G where the tree is a path, we will denote it by $\mathbf{pw}(G)$.

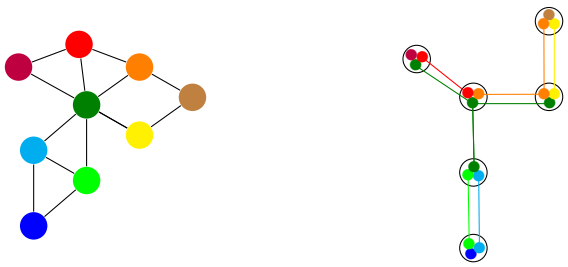

Figure 1: A graph and one of its tree decompositions

Definition 6 (Nice tree decomposition)**.** A nice tree decomposition of a graph G is a rooted tree decomposition $(T, \{X_t\}_{t \in V(T)})$ such that:

- the root and leaves of T have empty bags; and
- other nodes are of one of the following types:
  - **Introduce vertex node**: a node t with exactly one child t' such that $X_t = X_{t'} \cup \{v\}$ with $v \notin X_{t'}$. We say that v is introduced at t.
  - **Forget vertex node**: a node t with only one child t' such that $X_t = X_{t'} \setminus \{v\}$ with $v \in X_{t'}$. We say that v is forgotten at t.
  - **Join node**: a node t with two children t_1, t_2 such that $X_t = X_{t_1} = X_{t_2}$.

For each node t of the decomposition, we define a *partial* graph $G_t = G\left[\bigcup_{s \leq t} X_s\right] - E(G[X_t])$.

Note that edges of partial graphs appear at forget vertex nodes and that they correspond to adding edges between the forgotten vertex and its neighbours.

From a tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ of G of width k, a nice tree decomposition of width k with $O(k|V(G)|)$ nodes can be computed in time $O(k^2 \cdot \max(|V(T)|, |V(G)|))$ [Klo94].

The state of the art for FPT computation of a tree decomposition of size k (or a certificate that the input has greater treewidth than k) is an algorithm running in time $k^{O(k^3)} \cdot n$ [Bod96]. There is also a very recent 2-approximation algorithm running in time $2^{O(k)} \cdot n$ [Kor21].

Since the number of edges of a graph is at most $\mathbf{tw} \cdot n$, this decomposition is only interesting for sparse graphs. This contrasts with clique-width which captures simple dense graphs.

To define clique-width and k-expressions, we first define labeled graphs and operations on them.

Definition 7. A *k-labeled graph* is a triple $G = (V, E, \gamma)$ where (V, E) is a graph and $\gamma : V \to \{1, \ldots, k\}$ is a function called *labeling function*. We let $V(G), E(G)$, and $\gamma(G)$ denote the set of vertices, set of edges, and labeling function of labeled graph G.

Given a k-labeled graph $G = (V, E, \gamma)$, we refer to $\gamma^{-1}(\{i\})$ by vertices of label i or even simply label i.

We will only consider undirected graphs in the following but this notion can be extended to directed graphs.

Definition 8. For two k-labeled graphs G_1, G_2, $G_1 \oplus G_2$ is their disjoint union.

For a given k-labeled graph G, $\rho_{i \to j}(G)$ is the labeled graph obtained from G when changing the label of vertices labeled i to j.

For a given k-labeled graph G, $\eta_{i \times j}(G)$ is the labeled graph obtained from G by adding edges between vertices labeled i and vertices labeled j.

The graph consisting of a single vertex v labeled i is simply denoted $i(v)$.

A *k-expression* is a term that defines a labeled graph using these operations.

Definition 9. The *clique-width* of a graph is the minimal k such that it can be represented by a k-expression.

Definition 10. A linearized version of clique-width called *linear clique-width* corresponds to the minimum width of k-expressions such that disjoint union operations must contain a single vertex on one side.

Note that it may happen that an edge having its endpoints in two different labels i and j is already present in the graph G before performing the join $\eta_{i\times j}(G)$. Despite that the edge is not produced twice, the existence of such a situation may be problematic in our algorithms when we only consider a compact representation of G. To circumvent this problem, we can assume that every edge of a graph appears at most once in the join of our given k-expressions. More precisely, when performing a join operation $\eta_{i\times j}(G)$, we can assume that none of the edges in G has its endpoints in i and j, respectively. An expression with such property can be computed in time $O(k^2 \cdot n)$ from a given arbitrary expression of size n [CO00].

The following inequality stands between treewidth and clique-width : $\mathbf{cw}(G) \leq 3 \cdot 2^{\mathbf{tw}(G)-1}$ Hence, the class of graphs of bounded treewidth is included in the class of graphs of bounded clique-width. Graphs of bounded clique-width can be dense unlike graphs of bounded treewidth. A simple example is the complete graph on n vertices: it has treewidth $n-1$ (a single bag containing all vertices) but linear clique-width 2 (add vertices one by one by joining a label containing the new vertex to a label containing other vertices and then relabeling the new vertex).

Pathwidth can be compared to the other considered parameters with $\mathbf{tw}(G) \leq \mathbf{pw}(G)$ (obvious from definition) and $\mathbf{cw}(G) \leq \mathbf{lcw}(G) \leq \mathbf{pw}(G) + 2$ (use one label per vertex in the bag and an additional label for forgotten vertices). This means that a complexity lower bound involving pathwidth also stands for treewidth and clique-width.

There is no known FPT algorithm to obtain an optimal k-expression. The best approximation algorithms running in FPT time provide only expressions with exponentially too many labels [Oum05]. Graphs of clique-width at most 3 (called distance hereditary graphs) can be recognized in polynomial time [CHL+12]. Recognition of graphs of clique-width at most k for $k > 3$ is an open problem, hence for algorithms parameterized by clique-width, we assume that the graph is given with a k-expression describing it.

The monadic second order logic of graphs consists of formulae on vertices, edges, sets of edges and sets of vertices with predicates for equality, membership, adjacency and incidence testing. MSO_2 stands for the Monadic Second Order logic of graphs in which quantification over sets of vertices and sets of edges is possible, in MSO_1 only quantification over sets of vertices is allowed. A common extension called counting monadic second order logic consists of adding modular counting predicates.

A theorem by Bruno Courcelle shows that a graph property defined by a formula φ in MSO_2 can be checked in time $f(|\varphi|, \mathbf{tw}) \cdot n$ on graphs of treewidth $\mathbf{tw}$ [Cou90]. Similarly, a graph property defined by a formula φ in MSO_1 can be checked in time $g(|\varphi|, \mathbf{cw}) \cdot n$ on graphs of clique-width $\mathbf{cw}$ [CMR00]. The original hypothesis that a decomposition is given as an input is not necessary due to approximation algorithms for both decompositions. Courcelle's theorems give a very general result, however the running time of algorithms constructed by the proof is often irrelevant. These results are mainly interesting from a theoretical point of view.

Definition 11. A *tree-partition* of a graph $G = (V, E)$ is a partition $\mathcal{U} = \{V_1, \ldots, V_\ell\}$ of V

such that the quotient[2] of G by $\mathcal{U}$ is a forest.

The *tree-partition-width* of a graph is the minimum over its tree-partitions of the maximum size of a bag. We denote it by $\mathbf{tpw}(G)$.

Tree-partition-width has also been called *strong treewidth*, and can equivalently be defined by fixing a tree embedding and then mapping the vertices of the graph to this tree while preserving adjacency (edges with both endpoints mapped to the same vertex of the tree are considered to be preserved). Similarly to treewidth, we call *bags* the set of vertices of the graph that are mapped to a given vertex of the tree. Keep in mind that the bags of a tree-partition are *disjoint*. However, by adding an intermediate bag on each edge of the tree, which contains the union of the bags on the edge's endpoints, we obtain a tree decomposition. This shows $\mathbf{tw}(G) \leq 2\,\mathbf{tpw}(G) - 1$.

Given a graph width measure μ, we call *logarithmic* μ the parameter $\mu / \log n$ where n is the number of vertices. This means the width μ of an instance is bounded by $k \log n$ when k is this parameter.

2.4. Related work

The name XNLP was introduced recently by Bodlaender et al. [BGNS21], but the class was first studied by Elberfeld et al. [EST15], who called it $N[f\,\mathrm{poly}, f\,\log]$. It is the class of problems that can be solved nondeterministically in time $f(k)n^{O(1)}$ and space $f(k)\log(n)$. This class is not closed by fpt-reductions, but is closed under parameterized logspace reductions (pl-reductions), which require the reduction to use $f(k) + O(\log n)$ space. A less restrictive reduction type, for which XNLP would still be closed, is a reduction that uses $O(f(k)\log n)$ space and $O(g(k)n^c)$ time. Guillemot introduced the class WNL which is the variant of XNLP that is closed by fpt-reductions [Gui11].

The first established complete problems for XNLP are Timed Nondeterministic Cellular Automaton and Longest Common Subsequence, due to Elberfeld et al. [EST15]. Bodlaender et al. followed up with a large collection of complete problem including a "chained" variant of SAT, Bandwidth, List Coloring parameterized by pathwidth, and some problems related to scheduling and reconfiguration.

Longest Common Subsequence and Bandwidth were already known to be hard for W[t] for any constant t. This can be generalised by the following statement proved in [BGNS21].

Lemma 1. If a parameterized problem Q is XNLP-hard, then it is hard for class W[t] for each $t \in \mathbf{Z}^+$.

XNLP is contained in XP because the reachable configurations of the nondeterministic Turing machine can be tabulated using $n^{f(k)}$ space and then the computation can be done via dynamic programming on these configurations in time polynomial in the number of configurations.

The following conjecture on XNLP-hard problems is an equivalent generalisation of a conjecture on Longest Common Subsequence by Pilipczuk and Wrochna [PW18].

[2] In the quotient graph, the vertices are the parts of the partition, and there is an edge between two parts if there is a vertex in each part so that the vertices are adjacent.

Conjecture 1 (Slice-wise Polynomial Space Conjecture)**.** XNLP-hard problems do not have an algorithm that runs in time $n^{f(k)}$ and space $f(k)n^c$, with f a computable function, k the parameter, n the input size, and c a constant.

XNLP membership is closely related to the existence of a dynamic programming algorithm, while XNLP-hardness can be thought of as showing that we can't hope for better algorithms if the conjecture holds.

The following problems shown to be XNLP-complete in [BGNS21], are often used for reductions.

Chained Positive CNF-SAT
Input: r sets of Boolean variables $X_1, X_2, \dots X_r$, each of size q; an integer $k \in \mathbf{N}$; Boolean formula ϕ, which is in conjunctive normal form and an expression on $2q$ variables, using only positive literals; for each i, a partition of X_i into $X_{i,1}, \dots, X_{i,k}$ that are all of the same size.
Parameter: k.
Question: Is it possible to satisfy the formula $\bigwedge_{1 \leq i \leq r-1} \phi(X_i, X_{i+1})$ by setting from each set $X_{i,j}$ exactly 1 variable to true and all others to false?

Chained Multicolored Clique
Input: A graph $G = (V, E)$, a partition of V into sets $V_1, \dots, V_r$, such that each edge $uv \in E$ with $u \in V_i$ and $v \in V_j$ satisfies $|j - i| \leq 1$, and partitions $V_{i,1}, \dots, V_{i,k}$ of each V_i.
Parameter: k.
Question: Is there a subset $W \subseteq V$ such that for all i, j $|W \cap V_{i,j}| = 1$, and, for each $i \in [r-1]$, $G[W \cap (V_i \cup V_{i+1})]$ is a clique ?

2.5. Overview of the contribution

In Section 3, we find new complete problems for XNLP. In particular, we find complete problems for parameters corresponding to denser settings. This is done by using the parameters linear clique-width and linear mim-width. We also obtain completeness for a variant of Bandwidth.

Since XNLP seems naturally tied to linear structures and dynamic programming, we naturally turn our attention to dynamic programming on 'treelike decompositions'. Indeed, most dynamic programming schemes generalize to this setting. This is the reason why many standard graph decompositions are based on a tree structure, while we had to restrict ourselves to linear variants for XNLP.

In the following sections, we look at possible generalizations of XNLP to treelike structures. In Section 4, we first consider List Coloring on trees and obtain membership in L, which is generalized to XL membership for graphs of tree-partition-width k. In Section 5, we define the class XALP, a natural superset of XNLP that extends it to treelike structures, and show completeness of Tree-Chained Multicolored Clique, which allows us to translate hardness for linear width measures to treelike width measures. In Section 6, we show that computing the tree-partition-width is XALP-complete, but approximating it can be done in

polynomial time.

The content of this report corresponds mostly to [BGJ22a, BGJ22b, BGJ+22c, BGJ+22d], see Appendix B for more details on personal contribution.

3. New completeness results for XNLP

The main intuition about XNLP-complete problems is that they are hard parameterized problems on linear structures. Another way of expressing it is that to certify a solution we require something of polynomial size but locally we only need a certificate of size $f(k)\log(n)$. All known XNLP-complete problems are solvable by dynamic programming in XP time and XP space and have some form of linear structure. This is conjectured to be optimal for this running time, see Conjecture 1.

Some XNLP-complete problems parameterized by (logarithmic) pathwidth are given in [BGNS21]:

- List Coloring parameterized by pathwidth.
- Independent Set and Dominating Set parameterized by logarithmic pathwidth.

A collection of flow problems parameterized by pathwidth were also shown to be XNLP-complete in [BCvdW22].

Since graphs of bounded pathwidth are sparse, a natural question to ask is whether XNLP requires this sparsity. Graphs of bounded linear clique-width can be dense (e.g. a clique has linear clique-width 2), but have a linear structure similarly to graphs of bounded pathwidth.

We give several examples of relatively natural problems parameterized by linear clique-width that are XNLP-complete in [BGJ+22c].

As already mentioned in Subsection 2.3, the linear clique-width is bounded by the pathwidth. This means that XNLP-hardness for (logarithmic) pathwidth implies XNLP-hardness for (logarithmic) linear clique-width. Similarly, XNLP membership for (logarithmic) linear clique-width implies XNLP membership for (logarithmic) pathwidth.

3.1. Complete problems for linear clique-width

While List Coloring parameterized by pathwidth is XNLP-complete, it seems to be too hard parameterized by linear clique-width. Indeed, it is NP-complete on cographs (graphs of clique-width at most 2). The easier Coloring parameterized by linear clique-width has already been studied in fine-grained parameterized complexity and has an optimal running time of $n^{O(2^k)}$ [FGL+19] using a dynamic programming algorithm. This makes it a plausible candidate for XNLP-completeness. We show XNLP-completeness of the following three problems:

Coloring
Input: A graph G described by a given linear k-expression, and an integer C.
Parameter: k.
Question: Is there a proper coloring of G using at most C colors ?

Maximum Regular Induced Subgraph
Input: A graph G described by a given linear k-expression, and two integers W and D. **Parameter:** k. **Question:** Is there a D-regular induced subgraph of G on at least W vertices?

Maximum Cut
Input: A graph $G = (V, E)$ described by a given linear k-expression, and an integer W. **Parameter:** k. **Question:** Is there a bipartition of V into (V_1, V_2) such that $\lvert E(V_1, V_2)\rvert \geq W$?

All three problems have already been studied and are known to admit XP algorithms and be W[1]-hard when parameterized by clique-width [AEIM14, MT09, MS08, BGP13, Wan94, FGLS14]. Our hardness is stronger and does not require much additional work, which in our opinion showcases XNLP as a meaningful tool for the classification of parameterized problems.

Theorem 2 ([BGJ+22c])**.** Coloring, Maximum Regular Induced Subgraph, and Max Cut parameterized by linear clique-width are XNLP-complete.

3.2. Complete problems for logarithmic pathwidth and logarithmic linear clique-width

We provide new XNLP-complete problems for the parameter logarithmic pathwidth, and show that these problems and the previously known XNLP-complete problems for this parameter [BGNS21] are also complete for the parameter logarithmic linear clique-width. Our results are summarised below.

The motivation to study the logarithmic linear clique-width or logarithmic pathwidth comes from the observation that many FPT algorithms with linear clique-width or pathwidth as parameter have a single exponential time dependency on the parameter. Thus, if linear cliquewidth or pathwidth is logarithmic in the size of the graph, these algorithms turn into XP algorithms. This allows us to relate such algorithms to Conjecture 1, and argue about space requirements of single exponential FPT algorithms.

Theorem 3 ([BGJ+22c])**.** When parameterized by logarithmic pathwidth or logarithmic linear clique-width, Independent Set, Dominating Set, q-List-Coloring and q-Coloring for $q \geq 3$, and Odd Cycle Transversal are XNLP-complete, and Feedback Vertex Set is XNLP-hard.

The problems q-List-Coloring and q-Coloring correspond to variants of List-Coloring and Coloring where the sought coloring assigns only colors from $[q]$. Odd Cycle Transversal is the problem of finding in a graph a subset S of vertices such that every odd cycle contains a vertex of S. This is equivalent to asking that the deletion of S leaves the graph bipartite. Feedback Vertex Set is the problem of finding in a graph a subset of vertices whose deletion leaves the graph acyclic.

Lokshtanov et al. [LMS18] established tight lower bounds for these problems for the parameter pathwidth under the Strong Exponential Time Hypothesis. Several of our gadgets are based on those used for these lower bounds by [LMS18].

3.3. Complete problems for other width measures

During a visit of Lars Jaffke and Paloma T. Lima, some problems parameterized by linear mim-width were shown to be XNLP-complete.

Theorem 4 ([BGJ+22c]). Independent Set, Dominating Set, q-Coloring for $q \geq 5$, and Feedback Vertex Set are XNLP-complete when parameterized by the linear mim-width of the input graph.

The linear mim-width of a vertex ordering σ corresponds to the maximum size of an induced matching across the cut $(\sigma^{-1}([i]), \sigma^{-1}([i+1, n]))$. The linear mim-width of a graph is the minimum linear mim-width over all vertex orderings.

3.4. Bandwidth variants

The Bandwidth problem was shown to be XNLP-complete even restricted to caterpillars in [BGNS21, Bod20]. It can be expressed as the problem of finding a permutation of the vertices of a graph such that the resulting adjacency matrix as nonzero entries only along its diagonal.

Bandwidth
Input: A graph $G = (V, E)$ and an integer k.
Parameter: k.
Question: Is there an ordering $\alpha : V \to [n]$ such that for each $uv \in E$, $|\alpha(u) - \alpha(v)| \leq k$?

We consider the following variants.

Bipartite Bandwidth
Input: A bipartite graph $G = (X, Y, E)$ and an integer k.
Parameter: k.
Question: Are there orderings $\alpha : X \to [n]$ and $\beta : Y \to [m]$ such that for each $uv \in E$, $|\alpha(u) - \beta(v)| \leq k$?

A possible application of this problem is as follows. Let a matrix M be given. Create a vertex $x_i \in X$ for each row i and a vertex $y_j \in Y$ for each column j, and let x_i be adjacent to y_j if and only if $M_{i,j} \neq 0$. This graph has bipartite bandwidth at most k if and only if the rows and columns of M can be permuted (individually) in such a way that all non-zero entries are within k distance from the main diagonal.

Topological Bandwidth
Input: A graph G, and an integer k.
Parameter: k.
Question: Is there a subdivision G' of G of bandwidth at most k?

This problem has applications for solving systems of equations where the edge subdivision corresponds to creating the copy of a variable.

We obtain the following result.

Theorem 5 ([BGJ+22c]). Bipartite Bandwidth is XNLP-complete for trees.

Open Question 1. Is Topological Bandwidth XNLP-complete?

It seems that reducing from Bandwidth by replacing each vertex by a big enough clique might work. Membership is not much harder to obtain than for Bandwidth.

4. List coloring on trees

List Coloring parameterized by pathwidth is XNLP-complete, but can we extend this result to the parameter treewidth? As a first step towards answering this question, we consider the problem on trees, which have unbounded pathwidth and treewidth 1.

Note that trees have pathwidth $O(\log n)$. This implies that the problem can be solved nondeterministically with $O(\log^2 n)$ space and polynomial time. We first show that we can actually solve the problem deterministically with $O(\log^2 n)$ space and polynomial time. We then improve the space complexity to $O(\log n)$. Proofs of these results are given in [BGJ22a].

Trees are 2-list-colorable meaning that if every vertex has at least 2 available colors, then we are sure to find a proper coloring satisfying the constraint. This implies that the hard part of the problem is to propagate the information from vertices that have a single available color. Using this it is relatively simple to design a linear time algorithm [JS97], but this algorithm uses linear space. We say that a vertex is *critical* if we manage to compute from part of the constraints on other vertices that at most one color is available for it.

4.1. A first algorithm

We define a procedure `solve`(v, p) which given a vertex v and number $p \in \{0, 1, \ldots, n\}$, determines whether the subtree T_v rooted on v can be list coloured, where for $p \geq 1$ there is an additional constraint that v cannot receive the pth colour in $L(v)$. If `solve`(v, p) fails for two different values of p (or for $p = 0$), then we can reject.

Let T be a tree with root r, non-heavy children $v_1, \ldots, v_k$ and heavy child u. The algorithm works as follows.

1. For $i = 1, \ldots, k$, we recursively verify that T_{v_i} can be list coloured (`solve`$(v_i, 0)$); we reject if any of these rejects.
2. If $|L(r)| \geq d(r) + 1 = k + 2$, then we are 'non-critical': we free up our memory and recursively verify whether T_u can be list coloured.
3. Otherwise, we try to find the smallest $p_1 \in \{1, \ldots, |L(r)|\}$ for which we can assign v the p_1th colour in its list and extend to a list colouring of $T \setminus T_u$.
   - If no such p_1 exists, we reject.

   Next, we try to find the smallest $p_2 > p_1$ in $\{1, \ldots, |L(r)|\}$ for which we can assign v the p_2th colour in its list and extend to a list colouring of $T \setminus T_u$.
   - If such p_2 exists, we are again 'non-critical', free up our memory and recursively verify whether T_u can be list coloured.
   - If p_2 does not exist, then r must get colour p_1 and we run `solve`(u, p_1).

Theorem 6. The algorithm solves List Colouring on n-vertex trees in polynomial time and $O(\log^2 n)$ space.

We simply give the proof of the space bound, as the running time analysis is more technical. We stress the fact that traversing a subtree can be done in logspace, hence we can also count the number of vertices in a subtree, find the subtree with the most or the least vertices, and other similar procedures.

Claim 1. The algorithm uses $O(\log^2 n)$ bits of space.

Proof. Denote by $S(n)$ the maximum amount of memory used for a tree of order n. The largest subtree, and the next subtree in the size ordering can be computed in $O(\log n)$ space and linear time. The local variables require $O(\log n)$ memory, and we free their space before recursing to the largest subtree. We conclude that we have $S(n) \leq \max(O(\log n) + S(n/2), S(n-1))$. The master theorem allows us to conclude that $S(n) = O(\log^2 n)$. □

4.2. Improving space complexity

It turns out that with a few additional ideas, we can improve our algorithm to show membership in L. The algorithm becomes significantly more technical and will not described here, but a description is given in [BGJ22a].

The first idea is that we allow ourselves to store more information about the current choice when processing smaller subtrees. In particular, we partition subtrees by size in ranges of the form $[s(v)/2^{2^j}, s(v)/2^{2^{j-1}}[$, where $s(v)$ is the number of vertices in the subtree rooted in v, and store about 2^j bits when processing subtrees in this range. Since $\sum_{i=0}^{k} 2^j = 2^{k+1}$, having this amount of bits stored only costs $O(\log n)$ space.

The reason why we can afford to store this amount of information is that the more we get in the ranges of large subtrees, the less subtrees to check there are. Since we can get in a 'non-critical' state, whenever there are more colors remaining than the number of subtrees to check. More precisely, the subtrees in the ranges $[s(v)/2^{2^{j+1}}, s(v)/2^{2^j}[$ for j from k to 1, have size at least $s(v)/2^{2^k}$ but their total size is at most $s(v)$ so there are at most 2^{2^k} of them. By checking the subtrees range by range by increasing range, we can use only about 2^k bits to encode the index of a color in the set of colors that are not forbidden by smaller subtrees. Encoding colors like this requires to recursively recompute a color whenever necessary.

4.3. Generalisation to bounded tree-partition-width

We can extend the previous result to the parameterized setting by showing that List Coloring is in XL parameterized by the width of a given tree-partition. This gives an XP algorithm for the corresponding parameterized problem. We should not expect an FPT algorithm as List Coloring is already W[1]-hard parameterized by the vertex cover number which is an upper bound to tree-partition-width.

First, note that a graph of tree-partition-width k is $2k$-list-colorable. Then, the previous idea of criticality still works because vertices contained in some bag of a tree-partition are only adjacent to vertices of neighbouring bags. A subtree can now forbid at most k colors instead of at most one, but this only shifts our guarantees by a function of k.

Using the above and the logspace algorithm for trees, we obtain an algorithm using $O(k \log k \log n)$ space.

The parameter treewidth does not seem to allow such techniques that are based on local interactions, because some vertices could be in many bags. Note that generalizing the previous result to treewidth would contradict Conjecture 1. It turns out that List-Coloring parameterized by treewidth is not only XNLP-hard but also complete for the generalization of XNLP introduced in the next section.

5. XALP: a generalization of XNLP to treelike structures

XNLP is defined via Nondeterministic Turing Machines, and appears to be tied to linear structures. We can compare that to runs of NTMs being paths in the graph of configurations. This turns to a tree if we replace NTMs by Alternating Turing Machines, and with some tricks we can ask for the trees to have a specific structure.

5.1. Definition of XALP

Definition 12. Informally, an *Alternating Turing Machine* (ATM) is simply a Nondeterministic Turing Machine with some additional states that require that all transitions lead to acceptance. We call co-nondeterministic step this action of going to a constant amount of independant configurations and checking if each leads to acceptance. A run is then a tree and the running time of the ATM refers to its depth while the treesize refers to its total size.

We can always ask for co-nondeterministic steps to lead to exactly two states. Indeed, they can only lead to a finite amount of states, and they can be replaced by a finite amount of co-nondeterministic steps that lead to two states.

XALP is the class of problems that can be solved by an Alternating Turing Machine working in $f(k)n^{O(1)}$ treesize and $f(k)\log n$ space.

Alternating Turing Machines have been extensively studied in the classical setting. In particular, the class of languages recognizable by ATMs with polynomial tree size and logarithmic space is known to be equal to the class of languages recognizable by NTMs equipped with a stack running in polynomial time and logarithmic space (not counting the stack), and to SAC^1, a class defined by circuits.

Already in the classical setting, it is believed that $\mathrm{NL} \subsetneq \mathrm{SAC}^1$. This already justifies the assumption XNLP $\subsetneq$ XALP, as the classical classes are the 'projections' to constant parameter of the parameterized classes.

The classical equivalences can be translated in the parameterized setting:

Theorem 7 ([BGJ+22d])**.** 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 that can be solved by an alternating Turing machine with $f(k)\log n$ memory whose computation fits in 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 that can be solved by an alternating Turing machine with $f(k)\log n$ memory whose computation fits in a tree which 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. $A[f\,\mathrm{poly}, f\log, f+\log]$, 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 in $f(k)n^{O(1)}$ treesize and using $O(\log n)+f(k)$ co-nondeterministic steps per computation path, for some computable function f.

The proof is not given here, but is in [BGJ+22d].

5.2. Complete problems

The characterisation 3 is used to obtain XALP-completeness of the following problem.

Tree-Chained Multicolored Clique
Input: A graph $G=(V,E)$, a binary tree $T=(I,F)$, and a partition of V into sets $(V_i)_{i\in I}$, such that each edge $uv\in E$ with $u\in V_i$ and $v\in V_j$ satisfies $i=j$ or $ij\in F$, and partitions of each V_i, $V_{i,1},\dots,V_{i,k}$.
Parameter: k.
Question: Is there a subset $W\subseteq V$ such that for all i,j $|W\cap V_{i,j}|=1$, and, for each $ij\in F$, $G[W\cap(V_i\cup V_j)]$ is a clique ?

We encode the simulation of an Alternating Turing Machine with adequate resources within this problem and also solve it with an Alternating Turing Machine with adequate resources (proof in [BGJ+22d]). This shows XALP-completeness of the problem. We do not use all of the clique edges to encode the simulation so it would be possible to obtain XALP-completeness of a variant asking for grids instead of cliques. This could be used to obtain XALP-complete problems for planar problems, tiling problems, or width parameters that are sparser than those previously mentioned. In particular, this should easily give that Binary CSP when parameterized by cutwidth is XNLP-complete, and when parameterized by treecut-width is XALP-complete. Binary CSP is basically another formalization of Multicolored Clique, where the graph is formed by identifying all vertices in a same part of the partition, and then having a list of available colors to assign the resulting vertices that behaves like choosing the vertex of the corresponding part.

Using this problem and the previous XNLP-hardness proofs, a lot of XALP-completeness results are almost immediate. Indeed, the gadgets from the XNLP-hardness can often be reused without changing anything. In [BGJ+22d], we give the following XALP-complete problems:

- List Coloring and Pre-Coloring Extension parameterized by treewidth.
- Tree-Chained CNF-SAT and more structured variants.
- The flow problems from [BCvdW22] parameterized by treewidth.

- Max Cut, Maximum Regular Induced Subgraph parameterized by clique-width.
- Independent Set, Dominating Set and variants parameterized by logarithmic treewidth.

5.3. Comparison to known classes

XNLP is a subset of XALP by definition because a NTM is a special case of ATM. For this reason, XALP-hard problems are also XNLP-hard and as such Conjecture 1 also applies to them. This also means that they are W[t]-hard for all positive integers t.

In simpler terms, a problem that is XALP-complete, can be solved in $n^{f(k)}$ time and $n^{f(k)}$ space, but we conjecture that they can't be solved in $n^{f(k)}$ time and $f(k)n^{O(1)}$ space. If we could solve it in $f(k)n^{O(1)}$ time, it would imply P = NP.

There are other hierarchies in parameterized complexity than the W hierarchy. It might be possible to relate them to XALP, in a similar way to the way XNLP relates to the W hierarchy.

Open Question 2. Is there an inclusion of the A hierarchy or the AW hierarchy in XALP ?

6. Computing tree-partitions

As already mentioned in the preliminaries, treewidth is bounded by tree-partition-width $\mathbf{tw} \leq 2\,\mathbf{tpw} - 1$. The converse is not true – consider for example a fan, i.e. a path with an additional vertex adjacent to all vertices of the path. Nonetheless, it is possible to bound tree-partition-width by the treewidth and the maximum degree: $\mathbf{tpw} = O(\Delta\,\mathbf{tw})$. The first proof is due to an anonymous reviewer of Ding and Oporowski, the constant is slightly improved by Wood in [Woo09][3].

In [BCvdW22], the XNLP-hardness results are constrasted with FPT algorithms parameterized by tree-partition-width. Although these results were theoretically interesting on their own, they required a tree-partition to be given. This motivated the study of the parameterized complexity of computing tree-partitions.

Computing an optimal tree-partition is XALP-complete, hence intractable. This can be seen as a form of generalization of the XNLP-completeness of Bandwidth to the treelike setting.

Tree-Partition-Width
Input: A graph G and an integer k.
Parameter: k.
Question: Is the tree-partition-width of G more than k?

Theorem 8 ([BGJ22b])**.** Tree-Partition-Width is XALP-complete.

However, computing a tree-partition of approximate width is FPT. This result can be obtained as a consequence of the result of Ding and Oporowski [DO96]. They only give a

[3]Note that the claimed constant in the abstract is a mistake.

characterisation of tree-partition-width by forbidden topological minors. However, finding a fixed topological minor can be done in FPT time, and their proof explicitly constructs a tree-partition. This is sufficient to obtain a tree-partition of width $f(k)$ in FPT time or certify that the tree-partition-width of the graph is greater than k, for some function f that is a tower of exponentials. The structure of their argument can be mostly reused to obtain a simpler approximation algorithm. We briefly explain the scheme of the approximation.

Let G denote the given graph and k be the target value for the tree-partition-width.

1. We compute a (possibly approximate) tree decomposition with target value $2k-1$, and obtain a decomposition of width w. If the treewidth is more than this value, the tree-partition-width must be more than k.
2. We compute the pairs of vertices that are connected by at least $b \geq 2k-1$ vertex disjoint paths. These pairs are the edges of the auxiliary graph G^b on $V(G)$.
3. We contract the connected components of G^b in G to obtain the auxiliary graph H.
4. We compute the 2-connected components of H. The degree within such a component is bounded by a function of k (combinatorial argument). If we observe a degree above this bound we report the tree-partition-width to be more than k.
5. We use the construction that gives $\mathbf{tpw} = O(\Delta\,\mathbf{tw})$ on each 2-connected component of H. We can have one vertex to be the single vertex of its bag for free, we do so for cutvertices as they are in several components (could be unbounded). We then combine the tree-partition of the 2-connected components into a tree-partition of H.
6. We conclude by replacing vertices of H by the vertices of G of the corresponding component of G^b. This gives a tree-partition of G.

Using this procedure, we can obtain the following result.

Theorem 9 ([BGJ22b])**.** There is an algorithm that computes a tree-partition of width $O(k^7)$ or reports that the tree-partition-width of the input graph is greater than k in time $k^{O(1)}n^2$.

The procedure can be adapted by using different algorithms to obtain running times that depend less on n and more on k, or conversely; with possibly better bounds on the width.

6.1. Other structural results

Wollan introduced recently the graph parameter treecut-width [Wol15]. Much like tree-partition-width, this parameter is a strengthening of treewidth. To get an idea of what this parameter means without getting into the technical details, we simply mention the following result of Wollan [Wol15]: treecut-width is bounded if and only if there is no large grid immersion.

It came to our attention that the algorithmic results in [BCvdW22] correspond to results obtained for treecut-width in [GKS15]. This raised the question of how similar the parameters are, which we answer in [BGJ22b] using insights from the result of Ding and Oporowski [DO96].

Theorem 10 ([BGJ22b])**.** Tree-partition-width is polynomially bounded by treecut-width.

However, the parameters are not bounded by functions of each other. Indeed, the graphs $K_{3,n}$ have unbounded treecut-width but tree-partition-width 3.

It seems that the algorithmic results obtained in [GKS15] for treecut-width extend to tree-partition-width, extending the results of [BCvdW22].

7. Conclusion and perspectives

We give new XNLP-complete problems for several structural parameters, but there are still many other possible parameters to consider. However, this still demonstrates XNLP as a simple and adequate tool to prove hardness of problems on linear structures.

The extension of XNLP to XALP is nice on two aspects: the definition is quite simple and makes the link to XNLP clear, and many problems that were complete for XNLP in their linear form are complete for XALP in their treelike form.

The parameter tree-partition-width is a natural and fruitful parameter which raised some questions. It would be nice to have an algorithmic meta-theorem similar to Courcelle's theorem to distinguish problems that can be solved in FPT time parameterized by tree-partition-width. It would also be nice to find the parameterized class for which List Coloring parameterized by tree-partition-width is complete.

The parameter treedepth was not studied in our work but the results of Pilipczuk and Wrochna [PW18] give evidence that natural problems for this parameter are unlikely to be complete for XALP. Usually this parameter allows for algorithms that use polynomial space.

References

[AEIM14] Yuichi Asahiro, Hiroshi Eto, Takehiro Ito, and Eiji Miyano. Complexity of finding maximum regular induced subgraphs with prescribed degree. *Theor. Comput. Sci.*, 550:21–35, 2014.

[BCvdW22] Hans L. Bodlaender, Gunther Cornelissen, and Marieke van der Wegen. Problems hard for treewidth but easy for stable gonality. *CoRR*, abs/2202.06838, 2022.

[BGJ22a] Hans L. Bodlaender, Carla Groenland, and Hugo Jacob. List colouring trees in logarithmic space. *CoRR*, abs/2206.09750, 2022.

[BGJ22b] Hans L. Bodlaender, Carla Groenland, and Hugo Jacob. On the parameterized complexity of computing tree-partitions. *CoRR*, abs/2206.11832, 2022.

[BGJ+22c] Hans L. Bodlaender, Carla Groenland, Hugo Jacob, Lars Jaffke, and Paloma T. Lima. XNLP-completeness for parameterized problems on graphs with a linear structure. *CoRR*, abs/2201.13119, 2022.

[BGJ+22d] Hans L. Bodlaender, Carla Groenland, Hugo Jacob, Marcin Pilipczuk, and Michał Pilipczuk. On the complexity of problems on tree-structured graphs. *CoRR*, abs/2206.11828, 2022.

[BGJJ22] Paul Bastide, Carla Groenland, Hugo Jacob, and Tom Johnston. Exact antichain saturation numbers via a generalisation of a result of lehman-ron, 2022.

[BGNS21] Hans L. Bodlaender, Carla Groenland, Jesper Nederlof, and Céline M. F. Swennenhuis. Parameterized problems complete for nondeterministic FPT time and logarithmic space, 2021. Extended abstract to appear in Proceedings FOCS 2021.

[BGP13] Hajo Broersma, Petr A. Golovach, and Viresh Patel. Tight complexity bounds for FPT subgraph problems parameterized by the clique-width. *Theor. Comput. Sci.*, 485:69–84, 2013.

[Bod96] Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

[Bod20] Hans L. Bodlaender. Parameterized complexity of bandwidth of caterpillars and weighted path emulation. *CoRR*, abs/2012.01226, 2020.

[CFK+15] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

[CHL+12] Derek G. Corneil, Michel Habib, Jean-Marc Lanlignel, Bruce Reed, and Udi Rotics. Polynomial-time recognition of clique-width at most 3 graphs. *Discrete Applied Mathematics*, 160(6):834–865, 2012. Fourth Workshop on Graph Classes, Optimization, and Width Parameters Bergen, Norway, October 2009.

[CMR00] Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000.

[CO00] Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77–114, 2000.

[Cou90] Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990.

[DF13] Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

[DO96] Guoli Ding and Bogdan Oporowski. On tree-partitions of graphs. *Discret. Math.*, 149(1-3):45–58, 1996.

[EST15] 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.

[FG06] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.

[FGK09] Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. A measure & conquer approach for the analysis of exact algorithms. *J. ACM*, 56(5):25:1–25:32, 2009.

[FGL+19] Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. Clique-width III: hamiltonian cycle and the odd case of graph coloring. *ACM Trans. Algorithms*, 15(1):9:1–9:27, 2019.

[FGLS14] Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Almost optimal lower bounds for problems parameterized by clique-width. *SIAM J. Comput.*, 43(5):1541–1563, 2014.

[GKS15] Robert Ganian, Eun Jung Kim, and Stefan Szeider. Algorithmic applications of tree-cut width. In Giuseppe F. Italiano, Giovanni Pighizzini, and Donald Sannella, editors, *Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part II*, volume 9235 of *Lecture Notes in Computer Science*, pages 348–360. Springer, 2015.

[Gui11] Sylvain Guillemot. Parameterized complexity and approximability of the Longest Compatible Sequence problem. *Discret. Optim.*, 8(1):50–60, 2011.

[HK62] Michael Held and Richard M. Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210, 1962.

[JP22] Hugo Jacob and Marcin Pilipczuk. Bounding twin-width for bounded-treewidth graphs, planar graphs, and bipartite graphs. *CoRR*, abs/2201.09749, 2022.

[JS97] Klaus Jansen and Petra Scheffler. Generalized coloring for tree-like graphs. *Discrete Applied Mathematics*, 75(2):135–155, 1997.

[Klo94] Ton Kloks. *Treewidth: computations and approximations*, volume 842. Springer Science & Business Media, 1994.

[Kor21] Tuukka Korhonen. Single-exponential time 2-approximation algorithm for treewidth. *CoRR*, abs/2104.07463, 2021.

[LMS18] Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. *ACM Trans. Algorithms*, 14(2):13:1–13:30, 2018.

[MS08] Luke Mathieson and Stefan Szeider. The parameterized complexity of regular subgraph problems and generalizations. In James Harland and Prabhu Manyem, editors, *Proceedings 14th Computing: The Australasian Theory Symposium (CATS 2008)*, volume 77 of *CRPIT*, pages 79–86. Australian Computer Society, 2008.

[MT09] Hannes Moser and Dimitrios M. Thilikos. Parameterized complexity of finding regular induced subgraphs. *J. Discrete Algorithms*, 7(2):181–190, 2009.

[Oum05] Sang-il Oum. Approximating rank-width and clique-width quickly. In Dieter Kratsch, editor, *Graph-Theoretic Concepts in Computer Science*, pages 49–58, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[PW18] 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.

[Wan94] Egon Wanke. k-NLC graphs and polynomial algorithms. *Discret. Appl. Math.*, 54(2-3):251–266, 1994.

[Wol15] Paul Wollan. The structure of graphs not admitting a fixed immersion. *J. Comb. Theory, Ser. B*, 110:47–66, 2015.

[Woo09] David R. Wood. On tree-partition-width. *Eur. J. Comb.*, 30(5):1245–1253, 2009.

A. Course of the internship

October until December Visit of Nicolas Bousquet and Théo Pierron at LIRIS (Lyon) with Carla. New XNLP-hardness proofs for logarithmic pathwidth and realizing that they also hold for linear clique-width. XNLP-hardness of Maximum Regular Induced Subgraph parameterized by linear clique-width. Looking at Bipartite Bandwidth. Looking at List Coloring on trees, and finding the $O(\log^2 n)$ algorithm. Continuing a project on twin-width started with Marcin Pilipczuk at the end of my M1 internship.

January and February Polishing write up of twin-width paper ([JP22]) and submission to WG22 (accepted). Looking at Max Cut/**lcw** and Capacitated Red-Blue Dominating Set/**pw**. Write up of XNLP results for submission to SWAT (rejected). Briefing on discussions of Carla and Hans with Marcin and Michał Pilipczuk at a workshop in Bonn at the end of December. Looking at Carla's idea of 'brackets' to improve the List Coloring algorithm on trees, realizing that it gives logspace membership. Writing up mid-term report. First (unsuccessful) look at the problem of computing tree-partitions.

March and April Looking seriously into the paper of Ding and Oporowski, concluding on the approximation scheme. Visit of MC2 team at LIP (Lyon) and presenting the twin-width result in the local seminar. Write up of the List Coloring paper and submission to ESA (accepted). Working on strategies for a combinatorial game during a visit of Miloš Stojaković. Participating to an online workshop on structural directed graph theory. Visit of AlGCo team at LIRMM (Montpellier).

May and June New XNLP results during visit of Lars Jaffke and Paloma Lima. Write up of new XNLP results into previous paper, XALP paper, and tree-partition-width paper. Visit of CombAlgo team at LaBRI (Bordeaux). Working on extremal combinatorics during and after a visit of Tom Johnston ([BGJJ22]). Comparing treecut-width and tree-partition-width. Attending WG22. Submission of XNLP paper, XALP paper and tree-partition-width paper to IPEC22 (accepted and XNLP paper won best paper award).

B. Rough description of personal contribution

I am the main contributor of XNLP-completeness results for logarithmic linear clique-width and for Maximum Regular Induced Subgraph, of the simple algorithm using $O(\log^2(n))$ space for List Coloring on trees, of the approximation algorithms for tree-partition-width, and of the bound of tree-partition-width by treecut-width.

Although I contributed to all other results, the logspace algorithm for List Coloring on trees is mostly due to Carla, the construction showing XALP-hardness of Tree-Partition-Width is mostly due to Hans, and the results on mim-width are mostly due to Lars and Paloma.