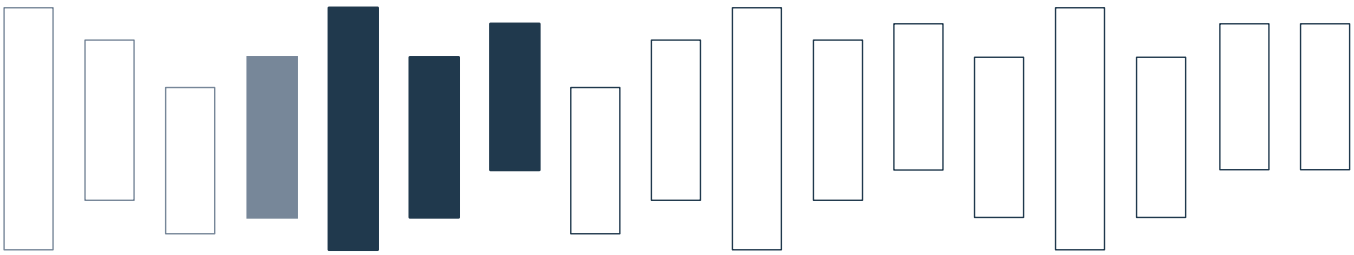
Technical Report AC-TR-15-005

December 2015

Algorithmic Applications of Tree-Cut Width

Robert Ganian, Eun Jung Kim and Stefan Szeider



This is the authors' copy of a paper that appeared at the 40th International Symposium on Mathematical Foundations of Computer Science, August 24-28, 2015 Milano (Italy).

www.ac.tuwien.ac.at/tr

Algorithmic Applications of Tree-Cut Width

Robert Ganian[1⋆], Eun Jung Kim[2], and Stefan Szeider[1⋆]

[1] Algorithms and Complexity Group, TU Wien, Vienna, Austria
[2] CNRS, Université Paris-Dauphine, Paris, France

Abstract. The recently introduced graph parameter *tree-cut width* plays a similar role with respect to immersions as the graph parameter *treewidth* plays with respect to minors. In this paper we provide the first algorithmic applications of tree-cut width to hard combinatorial problems. Tree-cut width is known to be lower-bounded by a function of treewidth, but it can be much larger and hence has the potential to facilitate the efficient solution of problems which are not known to be fixed-parameter tractable (FPT) when parameterized by treewidth. We introduce the notion of nice tree-cut decompositions and provide FPT algorithms for the showcase problems Capacitated Vertex Cover, Capacitated Dominating Set and Imbalance parameterized by the tree-cut width of an input graph G. On the other hand, we show that List Coloring, Precoloring Extension and Boolean CSP (the latter parameterized by the tree-cut width of the incidence graph) are W[1]-hard and hence unlikely to be fixed-parameter tractable when parameterized by tree-cut width.

1 Introduction

In their seminal work on graph minors, Robertson and Seymour have shown that all finite graphs are not only well-quasi ordered by the *minor* relation, but also by the *immersion* relation[3], the Graph Immersion Theorem [19]. This verified another conjecture by Nash-Williams [17]. As a consequence of this theorem, each graph class that is closed under taking immersions can be characterized by a finite set of forbidden immersions, in analogy to a graph class closed under taking minors being characterized by a finite set of forbidden minors.

In a recent paper [21], Wollan introduced the graph parameter *tree-cut width*, which plays a similar role with respect to immersions as the graph parameter *treewidth* plays with respect to minors. Wollan obtained an analogue to the Excluded Grid Theorem for these notions: if a graph has bounded tree-cut width, then it does not admit an immersion of the r-wall for arbitrarily large r [21, Theorem 15]. Marx and Wollan [16] proved that for all n-vertex graphs H with maximum degree k and all k-edge-connected graphs G, either H is an immersion of G, or G has tree-cut width bounded by a function of k and n.

⋆ Research supported by the FWF Austrian Science Fund (X-TRACT, P26696).

[3] A graph H is an immersion of a graph G if H can be obtained from G by applications of vertex deletion, edge deletion, and edge lifting, i.e., replacing two incident edges by a single edge which joins the two vertices not shared by the two edges.

In this paper we provide the first algorithmic applications of tree-cut width to hard combinatorial problems. Tree-cut width is known to be lower-bounded by a function of treewidth, but it can be much larger than treewidth if the maximum degree is unbounded (see Subsection 2.5 for an comparison of tree-cut width to other parameters). Hence tree-cut width has the potential to facilitate the efficient solution of problems which are not known or not believed to be fixed-parameter tractable (FPT) when parameterized by treewidth. For other problems it might allow the strengthening of parameterized hardness results.

We provide results for both possible outcomes: in Section 4 we provide FPT algorithms for the showcase problems Capacitated Vertex Cover, Capacitated Dominating Set and Imbalance parameterized by the tree-cut width of an input graph G, while in Section 5 we show that List Coloring, Precoloring Extension and Boolean CSP parameterized by tree-cut width (or, for the third problem, by the tree-cut width of the incidence graph) are not likely to be FPT. Table 1 provides an overview of our results. The table shows how tree-cut width provides an intermediate measurement that allows us to push the frontier for fixed-parameter tractability in some cases, and to strengthen W[1]-hardness results in some other cases.

	Parameter		
Problem	*tw*	*tree-cut width*	*max-degree and tw*
Capacitated Vertex Cover	W[1]-hard[4]	FPT(Thm 3)	FPT
Capacitated Dominating Set	W[1]-hard[4]	FPT(Thm 5)	FPT
Imbalance	Open[15]	FPT(Thm 4)	FPT[15]
List Coloring	W[1]-hard[7]	W[1]-hard(Thm 6)	FPT(Obs 4)
Precoloring Extension	W[1]-hard[7]	W[1]-hard(Thm 6)	FPT(Obs 4)
Boolean CSP	W[1]-hard[20]	W[1]-hard(Thm 7)	FPT[20]

Table 1: Overview of results (*tw* stands for treewidth).

Our FPT algorithms assume that a suitable decomposition, specifically a so-called *tree-cut decomposition*, is given as part of the input. Since the class of graphs of tree-cut width at most k is closed under taking immersions [21, Lemma 10], the Graph Immersion Theorem together with the fact that immersions testing is fixed-parameter tractable [10] gives rise to a non-uniform FPT algorithm for testing whether a graph has tree-cut width at most k. In a recent unpublished manuscript, Kim et al. [12] provide a uniform FPT algorithm which constructs a tree-cut decomposition whose width is at most twice the optimal one. Effectively, this result allows us to remove the condition that a tree-cut decomposition is supplied as part of the input from our uniform FPT algorithms.

We briefly outline the methodology used to obtain our algorithmic results. As a first step, in Section 3 we develop the notion of *nice tree-cut decompositions*[4] and show that every tree-cut decomposition can be transformed into a nice one in polynomial time. These nice tree-cut decompositions are of independent interest, since they provide a means of simplifying the complex structure of tree-cut

[4] We call them "nice" as they serve a similar purpose as the nice tree decompositions [13], although the definitions are completely unrelated.

decompositions. In Section 4 we introduce a general three-stage framework for the design of FPT algorithms on nice tree-cut decompositions and apply it to our problems. The crucial part of this framework is the computation of the "joins." We show that the children of any node in a nice tree-cut decomposition can be partitioned into (i) a bounded number of children with complex connections to the remainder of the graph, and (ii) a potentially large set of children with only simple connections to the remainder of the graph. We then process these by a combination of branching techniques applied to (i) and integer linear programming applied to (ii). The specifics of these procedures differ from problem to problem.

2 Preliminaries

2.1 Basic Notation

We use standard terminology for graph theory, see for instance [3]. All graphs except for those used to compute the torso-size in Subsection 2.4 are simple; the multigraphs used in Subsection 2.4 have loops, and each loop increases the degree of the vertex by 2.

Given a graph G, we let $V(G)$ denote its vertex set and $E(G)$ its edge set. The (open) neighborhood of a vertex $x \in V(G)$ is the set $\{y \in V(G) : xy \in E(G)\}$ and is denoted by $N_G(x)$. The closed neighborhood $N_G[v]$ of x is defined as $N_G(v) \cup \{v\}$. For a vertex subset X, the (open) neighborhood of X is defined as $\bigcup_{x \in X} N(x) \setminus X$ and denoted by $N_G(X)$. The set $N_G[X]$ refers to the closed neighborhood of X defined as $N_G(X) \cup X$. We refer to the set $N_G(V(G) \setminus X)$ as $\partial_G(X)$; this is the set of vertices in X which have a neighbor in $V(G) \setminus X$. The degree of a vertex v in G is denoted $deg_G(v)$. When the graph we refer to is clear, we drop the lower index G from the notation. We use $[i]$ to denote the set $\{0, 1, \ldots, i\}$.

2.2 Parameterized Complexity

We refer the reader to [5] for standard notions in parameterized complexity, such as the complexity classes FPT and W[1], FPT Turing reductions and the Multi-Colored Clique (MCC) problem.

2.3 Integer Linear Programming

Our algorithms use an Integer Linear Programming (ILP) subroutine. ILP is a well-known framework for formulating problems and a powerful tool for the development of fpt-algorithms for optimization problems.

Definition 1 (p-Variable Integer Linear Programming Optimization). *Let $A \in \mathbb{Z}^{q \times p}, b \in \mathbb{Z}^{q \times 1}$ and $c \in \mathbb{Z}^{1 \times p}$. The task is to find a vector $x \in \mathbb{Z}^{p \times 1}$ which minimizes the objective function $c \times \bar{x}$ and satisfies all q inequalities given by A and b, specifically satisfies $A \cdot \bar{x} \geq b$. The number of variables p is the parameter.*

Theorem 1 ([14,11,9,8]). *p-OPT-ILP can be solved using $O(p^{2.5p+o(p)} \cdot L)$ arithmetic operations in space polynomial in L, L being the number of bits in the input.*

2.4 Tree-Cut Width

The notion of tree-cut decompositions was first proposed by Wollan [21], see also [16]. A family of subsets $X_1, \ldots, X_k$ of X is a *near-partition* of X if they are pairwise disjoint and $\bigcup_{i=1}^{k} X_i = X$, allowing the possibility of $X_i = \emptyset$.

Definition 2. *A* tree-cut decomposition *of G is a pair $(T, \mathcal{X})$ which consists of a tree T and a near-partition $\mathcal{X} = \{X_t \subseteq V(G) : t \in V(T)\}$ of $V(G)$. A set in the family $\mathcal{X}$ is called a* bag *of the tree-cut decomposition.*

For any edge $e = (u, v)$ of T, let T_u and T_v be the two connected components in $T - e$ which contain u and v respectively. Note that $(\bigcup_{t \in T_u} X_t, \bigcup_{t \in T_v} X_t)$ is a partition of $V(G)$, and we use $cut(e)$ to denote the set of edges with one endpoint in each partition. A tree-cut decomposition is *rooted* if one of its nodes is called the root, denoted by r. For any node $t \in V(T) \setminus \{r\}$, let $e(t)$ be the unique edge incident to t on the path between r and t. We define the *adhesion* of t ($adh_T(t)$ or $adh(t)$ in brief) as $|cut(e(t))|$; if t is the root, we set $adh_T(t) = 0$.

The *torso* of a tree-cut decomposition $(T, \mathcal{X})$ at a node t, written as H_t, is the graph obtained from G as follows. If T consists of a single node t, then the torso of $(T, \mathcal{X})$ at t is G. Otherwise let $T_1, \ldots, T_\ell$ be the connected components of $T - t$. For each $i = 1, \ldots, \ell$, the vertex set Z_i of $V(G)$ is defined as the set $\bigcup_{b \in V(T_i)} X_b$. The torso H_t at t is obtained from G by *consolidating* each vertex set Z_i into a single vertex z_i. Here, the operation of consolidating a vertex set Z into z is to substitute Z by z in G, and for each edge e between Z and $v \in V(G) \setminus Z$, adding an edge zv in the new graph. We note that this may create parallel edges.

The operation of *suppressing* a vertex v of degree at most 2 consists of deleting v, and when the degree is two, adding an edge between the neighbors of v. Given a graph G and $X \subseteq V(G)$, let the *3-center* of (G, X) be the unique graph obtained from G by exhaustively suppressing vertices in $V(G) \setminus X$ of degree at most two. Finally, for a node t of T, we denote by $\tilde{H}_t$ the 3-center of (H_t, X_t), where H_t is the torso of $(T, \mathcal{X})$ at t. Let the *torso-size* $tor(t)$ denote $|\tilde{H}_t|$.

Definition 3. *The width of a tree-cut decomposition $(T, \mathcal{X})$ of G is $\max_{t \in V(T)}\{adh(t), tor(t)\}$. The tree-cut width of G, or* $\mathbf{tcw}(G)$ *in short, is the minimum width of $(T, \mathcal{X})$ over all tree-cut decompositions $(T, \mathcal{X})$ of G.*

We conclude this subsection with some notation related to tree-cut decompositions. For $t \in V(T) \setminus \{r\}$, we let $p_T(t)$ (or $p(t)$ in brief) denote the parent of t in T. For two distinct nodes $t, t' \in V(T)$, we say that t and t' are *siblings* if $p(t) = p(t')$. Given a tree node t, let T_t be the subtree of T rooted at t. Let $Y_t = \bigcup_{b \in V(T_t)} X_b$, and let G_t denote the induced subgraph $G[Y_t]$. The *depth* of a node t in T is the distance of t from the root r. The vertices of $\partial(Y_t)$ are called the *borders* at node t. A node $t \neq r$ in a rooted tree-cut decomposition is *thin* if $adh(t) \leq 2$ and *bold* otherwise.

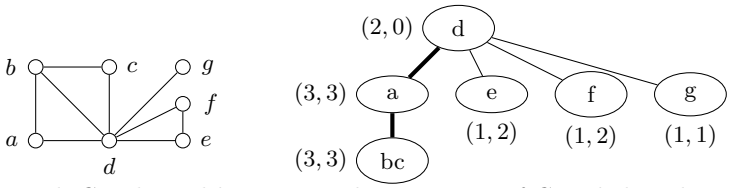


Fig. 1: A graph G and a width-3 tree-cut decomposition of G, including the torso-size (left value) and adhesion (right value) of each node.

2.5 Relations to Other Width Parameters

Here we review the relations between the tree-cut width and other width parameters, specifically *treewidth* (**tw**), *pathwidth* (**pw**), and *treedepth* (**td**) [18]. We also compare to the maximum over treewidth and maximum degree, which we refer to as *degree treewidth* (**degtw**).

Proposition 1 ([21,16]). *There exists a function h such that* $\mathbf{tw}(G) \leq h(\mathbf{tcw}(G))$ *and* $\mathbf{tcw}(G) \leq 4\mathbf{degtw}(G)^2$ *for any graph G.*

Below, we provide an explicit bound on the relationship between treewidth and tree-cut width, and show that it is incomparable with pathwidth and treedepth.

Proposition 2. *For any graph G it holds that* $\mathbf{tw}(G) \leq 2\mathbf{tcw}(G)^2 + 3\mathbf{tcw}(G)$.

Proposition 3. *There exists a graph class $\mathcal{H}_1$ of bounded pathwidth and treedepth but unbounded tree-cut width, and there exists a graph class $\mathcal{H}_2$ of bounded tree-cut width but unbounded pathwidth and treedepth.*

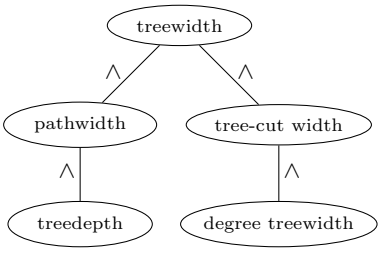


Fig. 2: Relationships between selected graph parameters (A>B means that every graph class of bounded A is also of bounded B, but there are graph classes of bounded B which are not of bounded A).

3 Nice Tree-Cut Decompositions

In this section we introduce the notion of a *nice tree-cut decomposition* and present an algorithm to transform any tree-cut decomposition into a nice one without increasing the width. A tree-cut decomposition $(T, \mathcal{X})$ rooted at r is *nice* if it satisfies the following condition for every thin node $t \in V(T)$: $N(Y_t) \cap \bigcup_{b \text{ is a sibling of } t} Y_b = \emptyset$.

The intuition behind nice tree-cut decompositions is that we restrict the neighborhood of thin nodes in a way which facilitates dynamic programming.

Lemma 1. *There exists a cubic-time algorithm which transforms any rooted tree-cut decomposition $(T, \mathcal{X})$ of G into a nice tree-cut decomposition of the same graph, without increasing its width or number of nodes.*

The proof of Lemma 1 is based on the following considerations. Let $(T, \mathcal{X})$ be a rooted tree-cut decomposition of G whose width is at most w. We say that a node t, $t \neq r$, is *bad* if it violates the above condition, i.e., $adh(t) \leq 2$ and there is a sibling b of t such that $N(Y_t) \cap Y_b \neq \emptyset$. For a bad node t, we say that b is a *bad neighbor* of t if $N(Y_t) \cap X_b \neq \emptyset$ and b is either a sibling of t or a descendant of a sibling of t.

> REROUTING(t): let t be a bad node and let b be a bad neighbor of t of maximum depth (resolve ties arbitrarily). Then remove the tree edge $e(t)$ from T and add a new tree edge $\{b, t\}$.
>
> TOP-DOWN REROUTING: as long as $(T, \mathcal{X})$ is not a nice tree-cut decomposition, pick any bad node t of minimum depth. Perform REROUTING(t).

The proof of Lemma 1 then follows by showing that REROUTING does not increase the width of the decomposition and that TOP-DOWN REROUTING terminates after performing at most a quadratic number of REROUTING calls.

The following Theorem 2 builds upon Lemma 1 by additionally giving a bound on the size of the decomposition.

Theorem 2. *If* $\mathbf{tcw}(G) = k$*, then there exists a nice tree-cut decomposition $(T, \mathcal{X})$ of G of width k with at most $2|V(G)|$ nodes. Furthermore, $(T, \mathcal{X})$ can be computed from any width-k tree-cut decomposition of G in cubic time.*

4 FPT Algorithms

In this section we will introduce a general dynamic programming framework for the design of FPT algorithms on nice tree-cut decompositions. The framework is based on leaf-to-root processing of the decompositions and can be divided into three basic steps:

- **Data Table**: definition of a data table $\mathcal{D}_T(t)$ ($\mathcal{D}(t)$ in brief) for a problem $\mathcal{P}$ associated with each node t of a nice tree-cut decomposition $(T, \mathcal{X})$.
- **Initialization and Termination**: computation of $\mathcal{D}(t)$ in FPT time for any leaf t, and solution of $\mathcal{P}$ in FPT time if $\mathcal{D}(r)$ is known for the root r.
- **Inductive Step**: an FPT algorithm for computing $\mathcal{D}(t)$ for any node t when $\mathcal{D}(t')$ is known for every child t' of t.

Let t be a node in a nice tree-cut decomposition. We use B_t to denote the set of thin children t' of t such that $N(Y_{t'}) \subseteq X_t$, and we let A_t contain every child of t not in B_t. The following lemma is a crucial component of our algorithms, since it bounds the number of children with nontrivial edge connections to other parts of the decomposition.

Lemma 2. *Let t be a node in a nice tree-cut decomposition of width k. Then $|A_t| \leq 2k + 1$.*

In the remainder of this section we employ this high-level framework on the design of FPT algorithms for the following problems: Capacitated Vertex Cover, Imbalance, and Capacitated Dominating Set.

4.1 Capacitated Vertex Cover

The Capacitated Vertex Cover is a generalization of the classical Vertex Cover problem. Unlike its uncapacitated variant, Capacitated Vertex Cover is known to be W[1]-hard when parameterized by treewidth [4].

A capacitated graph is a graph $G = (V, E)$ together with a capacity function $c : V \to \mathbb{N}_0$. Then we call $C \subseteq V$ a *capacitated vertex cover* of G if there exists a mapping $f : E \to C$ which maps every edge to one of its endpoints so that the total number of edges mapped by f to any $v \in C$ does not exceed $c(v)$. We say that f *witnesses* C.

tcw-Capacitated Vertex Cover (tcw-CVC) *Instance*: A capacitated graph G on n vertices together with a width-k tree-cut decomposition $(T, \mathcal{X})$ of G, and an integer d. *Parameter*: k. *Task*: Decide whether there exists a capacitated vertex cover C of G containing at most d vertices.

Data Table, Initialization and Termination. Informally, we store for each node t two pieces of information: the "cost" of covering all edges inside $G[Y_t]$, and how much more it would cost to additionally cover edges incident to Y_t. We formalize below.

For any graph $G = (V, E)$ and $U \subseteq V$, we let $cvc(G, U)$ denote the minimum cardinality of a capacitated vertex cover $C \subseteq U$ of G; if no such capacitated vertex cover exists, we instead let $cvc(G, U) = \infty$. For any node t in a nice tree-cut decomposition of a capacitated graph $G = (V, E)$, we then use a_t to denote $cvc(G[Y_t], Y_t)$.

Let E_t denote the set of all edges with both endpoints in Y_t and let K_t denote the set of edges with exactly one endpoint in Y_t. Then $\mathcal{Q}_t = \{ H = (Y_t \cup N(Y_t), E_t \cup E') \mid E' \subseteq K_t \}$. Finally, we define $\beta_t : \mathcal{Q}_t \to \mathbb{N}_0$ such that $\beta_t(H) = cvc(H, Y_t) - a_t$ (whereas ∞ acts as an absorbing element).

Definition 4. $\mathcal{D}(t) = (a_t, \beta_t)$.

Next, we show that the number of possible functions β_t is bounded.

Lemma 3. *Let k be the width of a nice tree-cut decomposition $(T, \mathcal{X})$ of G and let t be any node of T. Then $\beta_t(H) \in [k] \cup \{\infty\}$ for every $H \in \mathcal{Q}_t$.*

Lemma 4. *Let t be a leaf in a nice tree-cut decomposition $(T, \mathcal{X})$ of a capacitated graph G, and let k be the width of $(T, \mathcal{X})$. Then $\mathcal{D}(t)$ can be computed in time $2^{O(k \cdot \log k)}$.*

Observation 1 *Let (G, d) be an instance of tcw-*CVC *and let r be the root of a nice tree-cut decomposition of G. Then (G, d) is a yes-instance if and only if $a_r \leq d$.*

Inductive Step. Our next and final goal is to show how to compute $\mathcal{D}(t)$ of a node t once we have $\mathcal{D}(t')$ for each child t' of t. We call this problem CVC Join, and we use a two-step approach to solve it. First, we reduce the problem to a simplified version, which we call Reduced CVC Join and which has the following properties: A_t is empty, $adh(t) = 0$, and $G[X_t]$ is edgeless.

Lemma 5. *There is an FPT Turing reduction from* CVC Join *to $2^{O(k^2)}$ instances of* Reduced CVC Join *which runs in time $2^{O(k^2)} \cdot (|B_t| + 1)$.*

Lemma 6. *There exists an algorithm which solves* Reduced CVC Join *in time $k^{O(k^2)} \cdot (|B_t| + 1)$.*

Proof (Sketch). We develop an ILP formulation by partitioning B_t into types, which contain nodes with the same β_t and the same neighborhood in X_t. We use variables to express how edges are assigned between types and X_t. □

Corollary 1. *There exists an algorithm which solves* CVC Join *in time $k^{O(k^2)} \cdot (|B_t| + 1)$.*

Theorem 3. *tcw-*CVC *can be solved in time $k^{O(k^2)} \cdot n + |T|^3$.*

Proof. We use Theorem 2 to transform $(T, \mathcal{X})$ into a nice tree-cut decomposition with at most $2n$ nodes. We then use Lemma 4 to compute $\mathcal{D}(t)$ for each leaf t of T, and proceed by computing $\mathcal{D}(t)$ for nodes in a bottom-to-top fashion by Corollary 1. The total running time is dominated by $\sum_{t \in T}(k^{O(k^2)} \cdot (|B_t| + 1))$, which is bounded by $n \cdot k^{O(k^2)}$ since each node t' appears in at most a single set B_t. Once we obtain $\mathcal{D}(r)$, we can correctly output by Observation 1. □

4.2 Imbalance

The Imbalance problem was introduced by Biedl et al. [1]. The problem is FPT when parameterized by **degtw** [15]. In this subsection we prove that Imbalance remains FPT even when parameterized by the more general tree-cut width.

Given a linear order R of vertices in a graph G, let $\lhd_R(v)$ and $\rhd_R(v)$ denote the number of neighbors of v which occur, respectively, before ("to the left of") v in R and after ("to the right of") v in R. The *imbalance* of a vertex v, denoted $imb_R(v)$, is then defined as the absolute value of $\rhd_R(v) - \lhd_R(v)$, and the imbalance of R, denoted imb_R, is equal to $\sum_{v \in V(G)} imb_R(v)$.

tcw-Imbalance (tcw-IMB)
Instance: A graph $G = (V, E)$ with $\|V\| = n$, and a width-k tree-cut decomposition $(T, \mathcal{X})$ of G, and an integer d.
Parameter: k.
Task: Decide whether there exists a linear order R of V such that $imb_R \leq d$.

Data Table, Initialization and Termination. Let $A \subseteq B$ be sets and let f_A, f_B be linear orders of A, B respectively. We say that f_A is a *linear suborder* of f_B if the elements of A occur in the same order in f_A as in f_B (similarly, f_B is a *linear superorder* of f_A). The information we remember in our data tables can be informally summarized as follows. First, we remember the minimum imbalance which can be achieved by any linear order in Y_t. Second, for each linear order f of vertices which have neighbors outside of Y_t *and* for a restriction on the imbalance on these vertices, we remember how much the imbalance grows when considering only linear superorders of f which satisfy these restrictions. The crucial ingredient is that the restrictions mentioned above are "weak" and we only care about linear superorders of f which do not increase over the optimum "too much"; this allows the second, crucial part of our data tables to remain bounded in k.

For brevity, for $v \in Y_t$ we let $n_t(v)$ denote $|N_{V \setminus Y_t}(v)|$, i.e., the number of neighbors of v outside Y_t. Let f be a linear order of $\partial(Y_t)$ and let τ be a mapping such that $\tau(v \in \partial(Y_t)) \in \{-\infty, -n_t(v), -n_t(v)+1, \ldots, n_t(v), \infty\}$. We then call a tuple of the form (f, τ) an *extract* (of Y_t), and let $\mathcal{L}$ denote the set of all extracts (for nodes with adhesion at most k). The extract $\alpha = (f, \tau)$ is realized in Y_t (by R) if there exists a linear order R of Y_t such that

1. R is a linear superorder of f, and
2. for each $v \in \partial(Y_t)$:
   - if $\tau(v) \in \mathbb{Z}$ then $imb_R(v) = \tau(v)$,
   - if $\tau(v) = -\infty$ then $\triangleright_R(v) - \triangleleft_R(v) < -n_t(v) - 1$,
   - if $\tau(v) = \infty$ then $\triangleright_R(v) - \triangleleft_R(v) > n_t(v) + 1$.

The cost of a realized extract α, denoted $c(\alpha)$, is the minimum value of $\sum_{v \in Y_t} imb_R(v)$ over all R which realize α (notice that edges with only one endpoint in Y_t do not contribute to $c(\alpha)$). If α is not realized in Y_t, we let $c(\alpha) = \infty$. We store the following information in our data table: the cost of a minimum extract realized in Y_t, and the cost of every extract whose cost is not much larger than the minimum cost. We formalize below; let e_t denote the number of edges with one endpoint in Y_t.

Definition 5. *$\mathcal{D}(t) = (a_t, \beta_t)$ where $a_t = \min_{\alpha \in \mathcal{L}} c(\alpha)$ and $\beta_t : \mathcal{L} \to \mathbb{N}_0 \cup \{\infty\}$ such that $\beta_t(\alpha) = c(\alpha) - a_t$ if $c(\alpha) - a_t \leq 4e_t$ and $\beta_t(\alpha) = \infty$ otherwise.*

Notice that we are deliberately discarding information about the cost of extracts whose cost exceeds the optimum by over $4e_t$.

Observation 2 *The cardinality of $\mathcal{L}$ is bounded by $k^{O(k)}$, and hence the number of possible functions β_t is bounded by $k^{O(k^2)}$. Additionally, these may be enumerated in the same time.*

Lemma 7. *Let t be a leaf in a nice tree-cut decomposition $(T, \mathcal{X})$ of a graph G, and let k be the width of $(T, \mathcal{X})$. Then $\mathcal{D}(t)$ can be computed in time $k^{O(k)}$.*

Observation 3 *Let (G, d) be an instance of tcw-*IMB *and let r be the root of a nice tree-cut decomposition of G. Then (G, d) is a yes-instance if and only if $a_r \leq d$.*

Inductive Step. What remains is to show how to compute $\mathcal{D}(t)$ for a node t once $\mathcal{D}(t')$ is known for each child t' of t. Once again, we use the two-step approach of first reducing to a "simpler" problem and then applying a suitable ILP encoding. We call the problem we reduce to Reduced IMB Join.

Lemma 8. *There is an FPT Turing reduction from* IMB Join *to* $k^{O(k^2)}$ *instances of* Reduced IMB Join *which runs in time* $k^{O(k^2)} \cdot (|B_t| + 1)$.

Proof (Sketch). We branch over all linear orders of $f(k)$-many "important vertices" in A_t, over all possible extracts in $\mathcal{D}(t)$, and over all extracts in A_t which are compatible with the above. This gives sufficient information to compute the imbalance of vertices in X_t along with constraints on the placement of β_t, which yield an instance of Reduced IMB Join. □

Lemma 9. *There exists an algorithm which solves* Reduced IMB Join *in time* $k^{O(k^4)} \cdot (|B_t| + 1)$.

Proof (Sketch). The k absolute values can be translated into 2^k-many ILP instances by branching on whether they end up being positive or negative. Vertices in X_t separate the linear order into $k + 1$ "regions", and we can again partition B_t into types. Variables express how many children of type i have a border vertex placed in region j. □

The proof of the theorem below is then analogous to the proof of Theorem 3.

Theorem 4. *tcw-*IMB *can be solved in time* $k^{O(k^4)} \cdot n + |T|^3$.

4.3 Capacitated Dominating Set

Capacitated Dominating Set is a generalization of the classical Dominating Set problem by the addition of vertex capacities. It is known to be W[1]-hard when parameterized by treewidth [4].

Let $G = (V, E)$ be a capacitated graph with a capacity function $c : V(G) \to \mathbb{N}_0$. We say that $D \subseteq V(G)$ is a *capacitated dominating set* of G if there exists a mapping $f : V \setminus D \to D$ which maps every vertex to one of its neighbors so that the total number of vertices mapped by f to any $v \in D$ does not exceed $c(v)$.

> tcw-Capacitated Dominating Set (tcw-CDS)
> *Instance*: A capacitated graph G on n vertices together with a width-k tree-cut decomposition $(T, \mathcal{X})$ of G, and an integer d.
> *Parameter*: k.
> *Task*: Decide whether there exists a capacitated dominating set D of G containing at most d vertices.

The methods used to solve tcw-CDS are similar to those used to prove Theorem 3 and 4, and hence we only provide a high-level description of our approach here. For tcw-CDS, the table $\mathcal{D}(t)$ stores information about whether vertices in X_t occur in a dominating set, the residual capacities in $\partial(Y_t)$, and the size of a minimum capacitated dominating set which has these properties. The following steps are then analogous to those above.

Theorem 5. *tcw-*CDS *can be solved in time* $k^{O(k^2)} \cdot n + |T|^3$.

5 Lower Bounds

We show that List Coloring [6] and Precoloring Extension [2] are W[1]-hard parameterized by tree-cut width, strengthening the known W[1]-hardness results with respect to treewidth [7].

> tcw-List Coloring
> *Instance*: A graph $G = (V, E)$, a width-k tree-cut decomposition $(T, \mathcal{X})$ of G, and for each vertex $v \in V$ a list $L(v)$ of permitted colors.
> *Parameter:* k.
> *Task*: Decide whether there exists a proper vertex coloring c such that $c(v) \in L(v)$ for each $v \in V$.

The tcw-Precoloring Extension problem may be defined analogously as List Coloring; the only difference is that in Precoloring Extension lists are restricted to either contain a single color or all possible colors.

Observation 4 List Coloring *and* Precoloring Extension *parameterized by* **degtw** *are FPT.*

Theorem 6. *tcw-*List Coloring *and tcw-*Precoloring Extension *are* W[1]*-hard.*

We also show that the Constraint Satisfaction Problem (CSP) is W[1]-hard when parameterized by the tree-cut width of the incidence graph, even when restricted to the Boolean domain; this is not the case for **degtw** [20].

> tcw-CSP
> *Instance*: A CSP instance $I = (X, D, \mathcal{C})$ together with a width-k tree-cut decomposition $(T, \mathcal{X})$ of the incidence graph G_I of I.
> *Parameter:* k.
> *Task*: Decide whether I is satisfiable.

Theorem 7. *tcw-*Boolean CSP *is* W[1]*-hard.*

The proofs of Theorems 6 and 7 are based on a reduction from MCC.

6 Concluding Notes

We have provided the first algorithmic applications of the new graph parameter tree-cut width, considering a variety of hard combinatorial problems. In some cases we could establish fixed-parameter tractability, in some cases we could establish W[1]-hardness, staking off the potentials and limits of this parameter (see Table 1). The FPT algorithms make use of our new notion of nice tree-cut decompositions, which we believe to be of independent interest. We hope that our results and methods stimulate further work on problems parameterized by tree-cut width, which will result in a more refined parameterized complexity landscape; natural candidate problems include further graph layout problems or the General Factor Problem.

References

1. T. Biedl, T. Chan, Y. Ganjali, M. T. Hajiaghayi, and D. R. Wood. Balanced vertex-orderings of graphs. *DAM*, 148(1):27 – 48, 2005.
2. M. Biró, M. Hujter, and Z. Tuza. Precoloring extension. i. interval graphs. *Discrete Mathematics*, 100(1-3):267–279, 1992.
3. R. Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer Verlag, New York, 2nd edition, 2000.
4. M. Dom, D. Lokshtanov, S. Saurabh, and Y. Villanger. Capacitated domination and covering: A parameterized perspective. In *IWPEC*, Lecture Notes in Computer Science, pages 78–90. Springer Verlag, 2008.
5. R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer Verlag, 2013.
6. P. Erdős, A. L. Rubin, and H. Taylor. Choosability in graphs. *Congressus Numerantium*, 26:125–157, 1979.
7. M. R. Fellows, F. V. Fomin, D. Lokshtanov, F. Rosamond, S. Saurabh, S. Szeider, and C. Thomassen. On the complexity of some colorful problems parameterized by treewidth. *Information and Computation*, 209(2):143–153, 2011.
8. M. R. Fellows, D. Lokshtanov, N. Misra, F. A. Rosamond, and S. Saurabh. Graph layout problems parameterized by vertex cover. In *ISAAC*, Lecture Notes in Computer Science, pages 294–305. Springer, 2008.
9. A. Frank and É. Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Combinatorica*, 7(1):49–65, 1987.
10. M. Grohe, K.-i. Kawarabayashi, D. Marx, and P. Wollan. Finding topological subgraphs is fixed-parameter tractable. In *STOC'11—Proceedings of the 43rd ACM Symposium on Theory of Computing*, pages 479–488. ACM, New York, 2011.
11. R. Kannan. Minkowski's convex body theorem and integer programming. *Math. Oper. Res.*, 12(3):415–440, 1987.
12. E. Kim, S.-I. Oum, C. Paul, I. Sau, and D. Thilikos. FPT 2-approximation for constructing tree-cut decomposition. Manuscript, 2014.
13. T. Kloks. *Treewidth: Computations and Approximations*. Springer Verlag, Berlin, 1994.
14. H. Lenstra. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8:538–548, 1983.
15. D. Lokshtanov, N. Misra, and S. Saurabh. Imbalance is fixed parameter tractable. *Information Processing Letters*, 113(19-21):714–718, 2013.
16. D. Marx and P. Wollan. Immersions in highly edge connected graphs. *SIAM J. Discrete Math.*, 28(1):503–520, 2014.
17. C. S. J. A. Nash-Williams. On well-quasi-ordering finite trees. *Proc. Cambridge Philos. Soc.*, 59:833–835, 1963.
18. J. Nešetřil and P. O. de Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *European J. Combin.*, 27(6):1024–1041, 2006.
19. N. Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986.
20. M. Samer and S. Szeider. Constraint satisfaction with bounded treewidth revisited. *J. of Computer and System Sciences*, 76(2):103–114, 2010.
21. P. Wollan. The structure of graphs not admitting a fixed immersion. *J. Comb. Theory, Ser. B*, 110:47–66, 2015. Preliminary version available at http://arxiv.org/abs/1302.3867, 2013.