ALGORITHMIC APPLICATIONS OF TREE-CUT WIDTH[*]

ROBERT GANIAN[†], EUN JUNG KIM[‡], AND STEFAN SZEIDER[†]

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 that 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. On the other hand, we show that LIST COLORING, PRECOLORING EXTENSION, and BOOLEAN CSP (the last parameterized by the tree-cut width of the incidence graph) are W[1]-hard and hence unlikely to be FPT when parameterized by tree-cut width.

Key words. tree-cut width, immersion, parameterized complexity, structural parameters, integer linear programming

MSC codes. 05C85, 68Q25

DOI. 10.1137/20M137478X

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,[1] the Graph Immersion Theorem [38]. This verified a conjecture by Nash-Williams [35, 36]. 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 [41], 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 [41, Theorem 15]. Marx and Wollan [34] 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.

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

[1]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.

function of treewidth, but it can be much larger than treewidth if the maximum degree is unbounded (see Subsection 2.5 for a comparison of tree-cut width to other parameters). Hence tree-cut width has the potential to facilitate the efficient solution of problems that 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.

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

	Parameter		
Problem	*tw*	*tree-cut width*	*max-degree and tw*
CAPACITATED VERTEX COVER	W[1]-hard[8]	FPT[(Thm 4.9)]	FPT
CAPACITATED DOMINATING SET	W[1]-hard[8]	FPT[(Thm 4.23)]	FPT
IMBALANCE	Open[32]	FPT[(Thm 4.16)]	FPT[32]
LIST COLORING	W[1]-hard[11]	W[1]-hard[(Thm 5.1)]	FPT[(Obs 5)]
PRECOLORING EXTENSION	W[1]-hard[11]	W[1]-hard[(Thm 5.1)]	FPT[(Obs 5)]
BOOLEAN CSP	W[1]-hard[39]	W[1]-hard[(Thm 5.2)]	FPT[39]

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 [41, Lemma 10], the Graph Immersion Theorem together with the fact that immersions testing is FPT [23] gives rise to a nonuniform FPT algorithm for testing whether a graph has tree-cut width at most k. Kim et al. [28] provide a uniform FPT algorithm that 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*[2] 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 decompositions. In Section 4, we introduce a general three-stage framework for designing 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

---

[2]We call them "nice," as they serve a purpose similar to that of the nice tree decompositions [29], although the definitions are completely unrelated.

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, [7]. 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 $N_G[x]$ of x is defined as $N_G(x) \cup \{x\}$. For a vertex subset X, the (open) neighborhood of X is defined as $\bigcup_{x \in X} N_G(x) \setminus X$ 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 by $deg_G(v)$, and a vertex of degree 1 is called a *pendant vertex*. When the graph we refer to is clear, we drop the lower index G from the notation. We use $[i]$ to denote the set $\{0, 1, \dots, i\}$.

2.2. Parameterized complexity. A *parameterized problem* P is a subset of $\Sigma^* \times \mathbb{N}$ for some finite alphabet Σ. Let $L \subseteq \Sigma^*$ be a classical decision problem for a finite alphabet, and let p be a nonnegative integer-valued function defined on Σ^*. Then L *parameterized by* p denotes the parameterized problem $\{ (x, p(x)) \mid x \in L \}$ where $x \in \Sigma^*$. For a problem instance $(x, k) \in \Sigma^* \times \mathbb{N}$, we call x the main part and k the parameter. A parameterized problem P is fixed-parameter tractable (FPT) if a given instance (x, k) can be solved in time $f(k) \cdot |x|^{\mathcal{O}(1)}$ where f is an arbitrary computable function of k.

Parameterized complexity classes are defined with respect to *fpt-reducibility*. A parameterized problem P is *fpt-reducible* to Q if in time $f(k) \cdot |x|^{\mathcal{O}(1)}$ one can transform an instance (x, k) of P into an instance (x', k') of Q such that $(x, k) \in P$ if and only if $(x', k') \in Q$, and $k' \le g(k)$, where f and g are computable functions depending only on k. We also use the related notion of *FPT Turing reductions* [14, 9, 6]. For two parameterized problems P and Q, an FPT Turing reduction from P to Q is an FPT algorithm which decides whether (x, k) is in P, provided that there is an oracle access to Q for all instances (x', k') of Q with $k' \le g(k)$ for some computable function g. Informally speaking, a problem P admits an FPT Turing reduction to Q if one can solve an instance of P by an FPT algorithm which has access to an oracle which can solve Q but which may only be used for instances of Q whose parameter is upper-bounded by a function of the parameter of P. Owing to the definition, if P fpt-reduces or FPT Turing–reduces to Q and Q is FPT, then P is FPT as well.

Central to parameterized complexity is the following hierarchy of complexity classes, defined by the closure of canonical problems under fpt-reductions:

$$\mathsf{FPT} \subseteq \mathsf{W}[1] \subseteq \mathsf{W}[2] \subseteq \cdots \subseteq \mathsf{XP}.$$

All inclusions are believed to be strict. In particular, $\mathsf{FPT} \neq \mathsf{W}[1]$ under the Exponential Time Hypothesis [26].

The class $\mathsf{W}[1]$ is the analogue of NP in parameterized complexity. A major goal in parameterized complexity is to distinguish between parameterized problems which are in FPT and those which are $\mathsf{W}[1]$-hard, i.e., those to which every problem in $\mathsf{W}[1]$ is

fpt-reducible. There are many problems shown to be complete for $\mathsf{W}[1]$, or equivalently $\mathsf{W}[1]$-complete, including the MULTI-COLORED CLIQUE (MCC) problem [9].

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 2.1 (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* $c \times x$ *and satisfies all q inequalities given by A and b, specifically satisfying* $A \cdot x \geq b$. *The number of variables p is the parameter.*

Lenstra [31] showed that p-ILP, together with its optimization variant p-OPT-ILP (defined above), is in FPT. His running time was subsequently improved by Kannan [27] and Frank and Tardos [16] (see also [12]).

THEOREM 2.2 ([31, 27, 16, 12]). *p-OPT-ILP can be solved using* $\mathcal{O}(p^{2.5p+o(p)} \cdot L)$ *arithmetic operations in space polynomial in L, with 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 [41]; see also [34]. 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 2.3. *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 Υ_u and Υ_v be the two connected components in $T - e$ which contain u and v, respectively. Note that $(\bigcup_{t \in \Upsilon_u} X_t, \bigcup_{t \in \Upsilon_v} X_t)$ is a partition of $V(G)$, and we use $\mathsf{cut}(e)$ 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 ($\mathsf{adh}_T(t)$ or $\mathsf{adh}(t)$ in brief) as $|\mathsf{cut}(e(t))|$; if t is the root, we set $\mathsf{adh}_T(t) = 0$.

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* $\mathsf{tor}(t)$ denote $|\tilde{H}_t|$.

DEFINITION 2.4. *We let the* width *of a tree-cut decomposition $(T, \mathcal{X})$ of G be* $\max_{t \in V(T)}\{\mathsf{adh}(t), \mathsf{tor}(t)\}$. *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.*

See Figure 1 for an example.

We conclude this subsection with some notation related to rooted 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 $\mathsf{adh}(t) \leq 2$ and *bold* otherwise.
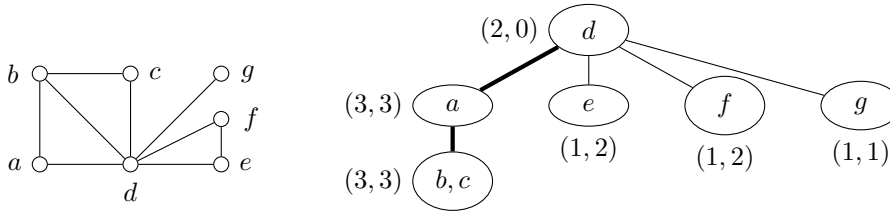

FIG. 1. *A graph G and a width-3 tree-cut decomposition of G, including the torso-size (left value) and adhesion (right value) of each node.*

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**) [7], and *treedepth* (**td**) [37]. We also compare to the maximum over treewidth and maximum degree, which we refer to as *degree treewidth* (**degtw**).

PROPOSITION 2.5 ([41, 34]). *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 a new explicit bound on the relationship between treewidth and tree-cut width and show that it is incomparable with pathwidth and treedepth. Since the proof of Proposition 2.6 below relies on Theorem 3.4 and notation introduced later, we postpone its proof to Section 4.

PROPOSITION 2.6. *For any graph G, it holds that $\mathbf{tw}(G) \leq 2\mathbf{tcw}(G)^2 + 3\mathbf{tcw}(G)$.*

PROPOSITION 2.7. *There is a graph class $\mathcal{H}_1$ of bounded pathwidth and treedepth but unbounded tree-cut width, and there is a graph class $\mathcal{H}_2$ of bounded tree-cut width but unbounded pathwidth and treedepth.*

Proof. Let $\mathcal{H}_1$ be the class of full ternary trees. It is easy to see that, for each ternary tree T_n of depth n, $\mathbf{pw}(T_n) \in \Theta(n)$ and $\mathbf{td}(T_n) \in \Theta(n)$ [40]. On the other hand, the tree-cut width of each T_n is bounded by a constant by Proposition 2.5.
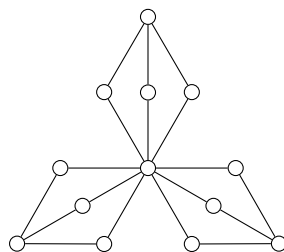

FIG. 2. *The graph S_3.*

Let $\mathcal{H}_2$ be the class of graphs S_n obtained from a star with n leaves $z_1, \ldots, z_n$ by replacing each edge with n subdivided edges (see Figure 2 for an example). It is

easy to see that $\mathbf{pw}(S_n) = 2 = \mathbf{td}(S_n) - 1$. We verify that $\mathbf{tcw}(S_n) \geq n$: suppose $\mathbf{tcw}(S_n) \leq n - 1$. Then any two leaves z_i and z_j, $i \neq j$, must be contained in the same bag in any tree-cut decomposition of width at most $n - 1$, as they are connected by n edge-disjoint paths. This means there exists a bag t containing all z_i's in any such tree-cut decomposition, which, however, implies that $\mathsf{tor}(t) \geq n$. □
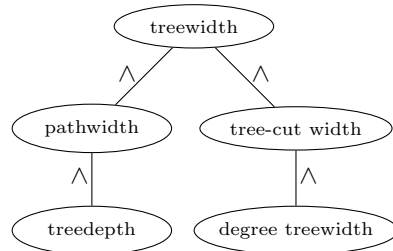

Fig. 3. *Relationships between selected graph parameters (A>B means that every graph class of bounded A is also of bounded B, but there are graph classes of bounded B which are not of bounded A).*

The relationships between graph parameters are illustrated in Figure 3.

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 3.1. *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 3.1 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., $\mathsf{adh}(t) \leq 2$ 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. In order to construct a tree-cut decomposition with the claimed property, we introduce a rerouting operation in which we pick a bad node and relocate it, that is, change the parent of the selected bad node.

> 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 selection criteria of a new parent in REROUTING(t) ensure that no new bad node is created at the depth of $p(t)$ or at a higher depth. This, together with the priority given to a bad node of minimum depth in TOP-DOWN REROUTING, implies that the tree gradually becomes free from bad nodes in a top-down manner.

LEMMA 3.2. *The procedure REROUTING(t) does not increase the width of the tree-cut decomposition.*

Proof. Let b be the bad neighbor of t chosen by REROUTING(t), and let $(T', \mathcal{X})$ be the tree-cut decomposition obtained from the tree-cut decomposition $(T, \mathcal{X})$ by REROUTING(t). We will show that for each $z \in V(T)$, it holds that

(1) $\mathsf{adh}_T(z) \geq \mathsf{adh}_{T'}(z)$, and

(2) $\mathsf{tor}_T(z) \geq \mathsf{tor}_{T'}(z)$,

from which the lemma follows.

Let us first consider Claim (1). Let P be the set of edges on the path in T between b and $p(t)$. Slightly abusing the notation, we also denote the path in T' connecting b and $p(t)$ by P. Then, for any edge p of T' which is not on P, it holds that $\mathsf{cut}_T(p) = \mathsf{cut}_{T'}(p)$, and hence Claim (1) holds for all nodes z which are not on P and also for $z = p(t)$. As for the remaining nodes z on P, note that $z \neq t$ and it holds that every edge between X_b and X_t lies in $\mathsf{cut}_T(e(z)) \setminus \mathsf{cut}_{T'}(e(z))$. Furthermore, by the thinness of t and the fact that X_t has a neighbor in X_b, there may exist at most one edge e' such that $e' \in \mathsf{cut}_{T'}(e(z)) \setminus \mathsf{cut}_T(e(z))$, and hence either $\mathsf{adh}_T(z) = \mathsf{adh}_{T'}(z)$ or $\mathsf{adh}_T(z) = \mathsf{adh}_{T'}(z) + 1$. Finally, note that $\mathsf{adh}_T(t) = \mathsf{adh}_{T'}(t)$. Thus, Claim (1) holds.

Now we consider Claim (2). From Claim (1) it follows that Claim (2) may only be violated if $N_T(z) \subset N_{T'}(z)$, which is only the case for $z = b$. However, it is easy to verify that $\mathsf{tor}_T(b) \geq \mathsf{tor}_{T'}(b)$ since $\{t\} = N_{T'}(b) \setminus N_T(b)$ and t is thin. $\qquad \square$

LEMMA 3.3. TOP-DOWN REROUTING *terminates after performing* REROUTING *at most* $2|T|$ *times, where* $(T, \mathcal{X})$ *is the initial tree-cut decomposition.*

Proof. The key observation is that once a bad node t is rerouted to be a child of its bad neighbor b, then b will remain as an ancestor of t in the tree-cut decompositions obtained by the subsequent TOP-DOWN REROUTING and thus b will not become a bad neighbor of t again. Indeed, the only way that b ceases to be a parent of t is when t is rerouted again as a bad node. Yet, the bad neighbor of t is a descendant of b and thus after the (second) rerouting t remains as a descendant of b. Furthermore, this implies that once a bad node t stops being a bad node, t will never turn into a bad node in the subsequent TOP-DOWN REROUTING.

To see that TOP-DOWN REROUTING terminates in at most $2|T|$ steps, consider maintaining the following auxiliary binary relation over the nodes of T during the procedure. Let (t, b) be a *bad pair* if t is a bad node of T, b is a bad neighbor of t, and there is an edge of G between X_t and X_b. Let B be the binary relation containing all bad pairs. We update the binary relation B as we perform TOP-DOWN REROUTING so that B always consists of the bad pairs in the current tree-cut decomposition. When a bad neighbor b of t becomes the parent of t by rerouting, we eliminate (t, b) from B. Therefore, the number of pairs ever eliminated from B throughout TOP-DOWN REROUTING provides an upper bound on the number of steps. Also the number of pairs eliminated from B equals the number of pairs which were ever inserted into B. The latter is bounded by $2|T|$ because the number of pairs which ever enter B with the first entry t is at most two (due to $\mathsf{adh}(t) \leq 2$) and any pair which was removed from B will never enter it again. It follows that TOP-DOWN REROUTING terminates in at most $2|T|$ steps. $\qquad \square$

Proof of Lemma 3.1. By definition, the output of TOP-DOWN REROUTING is a nice tree-cut decomposition. The lemma then follows from Lemmas 3.3 and 3.2. $\qquad \square$

Theorem 3.4 below builds upon Lemma 3.1 by additionally giving a bound on the size of the decomposition.

THEOREM 3.4. *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 quadratic time.*

Proof. For brevity, we say that a node t is empty if $X_t = \emptyset$. Consider a rooted width-k tree-cut decomposition $(T, \mathcal{X})$ of G. We can assume that each leaf t of T is nonempty since otherwise t may be deleted from T without changing the width.

Consider an empty nonleaf node t which has a single child t_1. Let $(T', \mathcal{X})$ be the tree-cut decomposition obtained by suppressing the node t. We claim that the width of $(T', \mathcal{X})$ is at most k. Indeed, $\mathsf{adh}_{T'}(t_1) = \mathsf{adh}_T(t_1) = \mathsf{adh}_T(t)$ and $\mathsf{adh}_{T'}(u) = \mathsf{adh}_T(u)$ for every other node $u \in V(T')$. Furthermore, $\mathsf{tor}_T(t_1) = \mathsf{tor}_{T'}(t_1)$ and, for the parent p of t (if it exists), we also have $\mathsf{tor}_T(p) = \mathsf{tor}_{T'}(p)$. It is then easily observed that the torso-size of all other nodes remains unchanged.

After exhaustively applying the contraction specified above, we arrive at a rooted tree-cut decomposition $(T'', \mathcal{X}'')$ of G where each empty node has at least two children. Hence, the number of empty nodes in T'' is upper-bounded by the number of nonempty nodes, which implies that $|T''| \le 2|V(G)|$. $(T'', \mathcal{X}'')$ can then be transformed into a nice tree-cut decomposition without having to increase the number of nodes by Lemma 3.1. $\qquad\square$

4. FPT agorithms. 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.

If the above holds for a problem $\mathcal{P}$, then $\mathcal{P}$ admits an FPT algorithm parameterized by tree-cut width. The **inductive step** usually represents the most complex part of the algorithm. Our notion of nice tree-cut decompositions provides an important handle on this crucial step, as we formalize below.

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

Proof. We partition the nodes in A_t into two sets: A_t' contains all thin nodes in A_t, and A_t'' contains all the bold nodes in A_t. We claim that $|A_t'| \le k$ and $|A_t''| \le k+1$, which will establish the statement. The inequality $|A_t'| \le k$ is easy to see. Indeed, recall that $N(Y_b) \subseteq X_t \cup (V(G) \setminus Y_t)$ for every $b \in A_t'$ since $(T, \mathcal{X})$ is nice. Furthermore, each $b \in A_t'$ satisfies $N(Y_b) \cap (V(G) \setminus Y_t) \ne \emptyset$ since otherwise b would have been included in B_t. Therefore, each $b \in A_t'$ contributes at least one to the value $\mathsf{adh}(t)$. From $\mathsf{adh}(t) \le k$, the inequality follows.

To prove $|A_t''| \le k + 1$, suppose $|A_t''| = \ell \ge k + 2$ for the sake of contradiction. Consider the torso H_t at t. For each $b \in A_t \cup B_t$, let z_b be the vertex of

H_t obtained by consolidating the vertex set Y_b in G and let z_{top} be the vertex of H_t obtained by consolidating the vertex set $V(G) \setminus Y_t$. Fix a sequence of suppressing vertices of degree at most two which yields a sequence of intermediate graphs $H_t = H_t^{(0)}, H_t^{(1)}, \ldots, H_t^{(m)} = \tilde{H}_t$. In this sequence, it is assumed that we suppress a vertex z_b with $b \in A'_t \cup B_t$, namely which represents a thin node, whenever this is possible.

Observe that no vertex representing a bold node (i.e., a node of A''_t) is adjacent with a vertex representing a thin node (i.e., a node of $A'_t \cup B_t$) in any graph appearing in the suppression sequence. This observation is valid on $H_t^{(0)} = H_t$ because the tree-cut decomposition at hand is nice and thus in H_t the vertex set $\{z_{top}\} \cup X_t$ separates the vertices obtained from $A'_t \cup B_t$ and those obtained from A''_t. By induction hypothesis, if a vertex z_q representing a bold node q becomes adjacent with a vertex $z_{q'}$ representing a thin node first time in the suppression sequence, say at $H_t^{(\ell)}$, this is due to the suppression of z_{top} in $H_t^{(\ell-1)}$. Then, however, we should have suppressed $z_{q'}$ in $H_t^{(\ell)}$ before z_{top} because $z_{q'}$ has degree at most two in $H_t^{(\ell)}$ (suppression does not increase the degree of any vertex) and we prioritize suppressing $z_{q'}$ over z_{top}.

Let us choose $b', b'' \in A''_t$ so that $z_{b'}$ and $z_{b''}$ are the first and the second (distinct) vertices among z_b for all $b \in A''_t$ whose degree strictly decreases in this sequence. Such b' and b'' must exist since at least two vertices z_b, where $b \in A''_t$, do not appear in $\tilde{H}_t$, and any z_b may only be removed by suppression. Let $a', a'' \in A_t \cup B_t \cup \{top\}$, and let $0 \le i < j \le m$ be such that the first decrease in the degrees of $z_{b'}$ and $z_{b''}$ is due to the suppressions of $z_{a'} \in V(H_t^{(i)})$ and of $z_{a''} \in V(H_t^{(j)})$. We observe that the respective degree of $z_{a'}$ and $z_{a''}$ are exactly one in $V(H_t^{(i)})$ and in $V(H_t^{(j)})$ since otherwise the degree of $z_{b'}$ and $z_{b''}$ would not decrease. Moreover, $z_{a'}$ and $z_{b'}$ are adjacent in $V(H_t^{(i)})$, and $z_{a''}$ and $z_{b''}$ are adjacent in $V(H_t^{(j)})$

By the previous observation that $z_{b'}$ is nonadjacent with any vertex representing a thin node in $V(H_t^{(i)})$ and $V(H_t^{(j)})$, we conclude that $a' = top$. When z_{top} is suppressed in $V(H_t^{(i)})$ and accordingly decreases the degree of $z_{b'}$, this means that in $V(H_t^{(i)})$ all the vertices obtained from thin nodes $A'_t \cup B_t$ have been eliminated in the previous steps. Therefore, all vertices in $V(H_t^{(i+1)}) \setminus X_t$ are those obtained from A''_t. It follows that $a'' = b'$.

Notice that the suppressing of $z_{a'}$, namely z_{top}, decreases the degree of $z_{b'}$ by one. That is, the degree of $z_{b'}$ in $H_t^{(i+1)}$ remains at least two. Now that suppressing $z_{b'}$ in $H_t^{(j)}$ strictly decreases the degree of $z_{b''}$, the degree of $z_{b'}$ in $H^{(j)}$ equals one. This implies that there is a vertex, say z_{a^*}, whose suppression further decreased the degree of $z_{b'}$ between the sequence of $H_t^{(i+1)}$ and $H_t^{(j)}$. However, then $a^* \in A''_t$, which contradicts our choice of b'' and $H_t^{(j)}$. This proves $|A''_t| = \ell \le k + 1$. $\qquad\square$

In the remainder of this section, we employ this high-level framework on the design of FPT algorithms parameterized by tree-cut width 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 which was originally introduced and studied in the classical setting [5, 24]. More recently, its parameterized complexity has also been studied in combination with various parameters [25, 8]. Unlike its uncapacitated variant, CAPACITATED VERTEX COVER is known to be W[1]-hard when parameterized by treewidth [8].

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.

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

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 $\beta_t : \mathcal{Q}_t \to \mathbb{N}_0 \cup \{\infty\}$ such that for every H in $\mathcal{Q}_t$, $\beta_t(H) = \mathsf{cvc}(H, Y_t) - a_t$ (whereas ∞ acts as an absorbing element and $\infty - \infty = \infty$).

DEFINITION 4.2. $\mathcal{D}(t) = (a_t, \beta_t)$.

Next, we show that the number of possible functions β_t is bounded. To this end, we make use of the following lemma.

LEMMA 4.3. *Let $G = (V, E)$ be any capacitated graph, and let G' be obtained from G by adding a pendant vertex x (with arbitrary capacity). Let $U \subseteq V$. Then either $\mathsf{cvc}(G', U) = \mathsf{cvc}(G, U)$ or $\mathsf{cvc}(G', U) = \mathsf{cvc}(G, U) + 1$ or $\mathsf{cvc}(G', U) = \infty$.*

Proof. We use induction on $|E|$. For $|E| = 0$, the lemma is easily seen to be true, as $\mathsf{cvc}(G, U) = 0$ and $\mathsf{cvc}(G', U) \in \{0, 1, \infty\}$.

For the inductive step, let $\{z\} = N(x)$, $c = \mathsf{cvc}(G, U)$, and $c' = \mathsf{cvc}(G', U)$. If $c = \infty$, then clearly also $c' = \infty$, so we may assume that $c \neq \infty$; let $C \subseteq U$ be a capacitated vertex cover of G of cardinality c. We distinguish the following cases:

1. if $z \notin U$, then $c' = \infty$;
2. if $c(z) = 0$, then $c' = \infty$;
3. otherwise, if $z \notin C$, then $C' = C \cup \{z\}$ is a capacitated vertex cover of G' and thus $c \leq c' \leq c + 1$.

The only case that remains is $z \in C$. It could occur that C itself witnesses $c' = c$, i.e., it is possible to allocate edges to C so that z has free capacity for the new edge xz. If z does not have free capacity to accommodate xz within C but $c' \neq \infty$, then there must exist a capacitated vertex cover C' which "frees up" at least one edge which is mapped to z in C; specifically, there must exist an edge $e = zq$ which is allocated to z in C and to q in C'. Notice that if $q \notin C$, we can add q to C and allocate qz to q, hence allowing xz to be allocated to z; the lemma would hold. Thus, we may assume $q \in C$.

We now construct a capacitated graph H from G, the capacitated vertex cover C, and an arbitrary witness f of C as follows. H is obtained from $G - z$ by attaching a new pendant vertex (with capacity 0) to every $v \in V(H)$ such that $v \in N(z)$ and $f(vz) = v$. Since at least one neighbor of z, specifically q, will not receive a new pendant vertex in this construction, it follows that $|E(H)| < |E(G)|$. Now we apply the inductive assumption on the capacitated graph H, the capacitated graph H' obtained from H by adding a pendant vertex z' adjacent to q, and $U_H = U \setminus \{z\}$. Observe that $c = \mathsf{cvc}(H, U_H) + 1$ (since one can add z to any capacitated vertex cover of size $\mathsf{cvc}(H, U_H)$ and obtain a capacitated vertex cover for G). Furthermore, $C' \cup (N(z) \cap U) \setminus \{z\}$ witnesses $\mathsf{cvc}(H', U_H) \neq \infty$, and so there are two final possibilities to consider:

1. $\mathsf{cvc}(H', U_H) = \mathsf{cvc}(H, U_H)$, in which case $c' = c$ since we can allocate the edge zq to q in G without increasing the size of the vertex cover.
2. $\mathsf{cvc}(H', U_H) = \mathsf{cvc}(H, U_H) + 1$, in which case $c' = c + 1 = \mathsf{cvc}(H', U_H) + 1$ since we can allocate the edge zq to q in G at a cost of 1 additional vertex in the cover.

To conclude the proof, we make explicit the construction of a capacitated vertex cover demonstrating the above claims. We start with a capacitated vertex cover K of H' witnessing $\mathsf{cvc}(H', U_H) = \mathsf{cvc}(H, U_H)$ (or $\mathsf{cvc}(H', U_H) = \mathsf{cvc}(H, U_H) + 1$) and add to it the vertex z. As for the allocation of edges, we retain the same allocation of all edges not incident to z as in K, allocate zq to q (this is made possible by the construction of H'), and use f to allocate all the remaining edges incident to z (this is made possible by the pendant vertices added during the construction of H). By "saving" on the edge zq, it is now possible to accommodate the new edge xz into z. $\qquad\square$

LEMMA 4.4. *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 $H \in \mathcal{Q}_t$.*

Proof. We actually prove a slightly stronger claim; for every $E' \subseteq K_t$, $\beta_t(H) \in [|E'|] \cup \infty$. Notice that $|K_t| = \mathsf{adh}(t)$ and hence $|E'| \leq k$. We proceed by induction; for $|E'| = 0$, the claim holds by definition.

For the inductive step, let $E'_1 = E'_0 \cup \{e\}$ and $e = ab$, where a is a vertex in Y_t and b is a neighbor of a in $N(Y_t)$, and let H_0 and H_1 be the graphs $(Y_t \cup N(Y_t), E_t \cup E'_0)$ and $(Y_t \cup N(Y_t), E_t \cup E'_1)$, respectively. The inductive claim then follows directly from Lemma 4.3 by making the pendant vertex x adjacent to a in H_0; indeed, observe that adding an edge between a and b has precisely the same effect on the capacitated vertex set as adding a new pendant vertex to a (since we restrict the capacitated vertex set to Y_t). $\qquad\square$

Since the graphs $H \in \mathcal{Q}_t$ that appear in the domain of β_t can be uniquely represented by the choice of $E' \subseteq K_t$, we can store β_t as a subset of $2^{K_t} \times ([k] \cup \{\infty\})$.

4.1.2. Initialization and termination.

LEMMA 4.5. *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 $2^{\mathcal{O}(k \cdot \log k)}$.*

Proof. $a_t = \mathsf{cvc}(G[Y_t], Y_t)$ can be computed in time $2^{\mathcal{O}(k \cdot \log k)}$ by known results [8] since $|Y_t| \leq k$. Furthermore, $|K_t| \leq k$ since $\mathsf{adh}(t) \leq k$, and so $|\mathcal{Q}_t| \leq 2^k$. What remains is to compute β_t by determining $\mathsf{cvc}(H, Y_t)$ for each H in $\mathcal{Q}_t$; for each H, this can be done by using the same $2^{\mathcal{O}(k \cdot \log k)}$ algorithm for the capacitated vertex cover problem while setting the capacities of vertices outside of Y_t to 0. In this way, we can

compute β_t in time $2^k \cdot 2^{\mathcal{O}(k \cdot \log k)}$, from which the lemma follows.   □

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 $a_r \leq d$.*

4.1.3. 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 formalize this problem below.

CVC JOIN
Instance: A tcw-CVC instance consisting of a capacitated graph G and an integer d, a nonleaf node t of a width-k nice tree-cut decomposition $(T, \mathcal{X})$ of G, and $\mathcal{D}(t')$ for each child t' of t.
Parameter: k.
Task: Compute $\mathcal{D}(t)$.

We use a two-step approach to solve CVC JOIN. 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, $\mathsf{adh}(t) = 0$, and $G[X_t]$ is edgeless. Recall that B_t is the set of all thin children t' of node t with $N(Y_{t'}) \subseteq X_t$ and A_t denotes the set of the children of t not in B_t.

For the following lemma, we remark that the linear dependency on n is merely caused by the fact that the instances of REDUCED CVC JOIN obtained in the Turing reduction are of size $\mathcal{O}(n)$. It is conceivable that this linear factor could be avoided with the use of a carefully designed data structure.

LEMMA 4.6. *There is an FPT Turing reduction from CVC JOIN to $2^{\mathcal{O}(k^2)}$ instances of REDUCED CVC JOIN which runs in time $2^{\mathcal{O}(k^2)} \cdot n$.*

Proof. Let $E_x = \{ ab \in E \mid a, b \in X_t \}$ and $E_1 = \{ ab \in E \mid \exists t_a \in A_t : a \in Y_{t_a}, b \in (Y_t \setminus Y_{t_a}) \}$; in other words, E_1 contains edges in Y_t which contribute to the adhesion of some node of A_t. Let E' denote the set of edges with only one endpoint in Y_t. By Lemma 4.1 and the bound of k on the width of $(T, \mathcal{X})$, it follows that $|E_1| \leq 2k^2 + k$ and $|E_x| \leq k^2$. Let $\mathcal{F}$ contain all mappings of edges from $E_x \cup E_1 \cup E'$ to one of their endpoints; formally, $\mathcal{F} = \{ f : (E_x \cup E_1 \cup E') \to V \mid f(ab) = a \text{ or } f(ab) = b \}$. Notice that $|\mathcal{F}| \leq 2^{\mathcal{O}(k^2)}$.

For any $f \in \mathcal{F}$ and $t' \in A_t \cup \{t\}$, we define the *completion* $U_{f,t'}$ as the graph $G[Y_{t'}]$ together with the vertices $N(Y_{t'})$ and each edge e such that $f(e) \in Y_{t'}$; informally, f designates how we want to cover our edges, and the completion is the relevant subproblem of covering edges in such a way within $Y_{t'}$. Let $X(f) = \{ v \in X_t \mid \exists e : f(e) = v \}$. For each f, we compute $\mathsf{Acost}(f) = \sum_{t' \in A_t} \beta_{t'}(U_{f,t'}) + a_{t'}$; informally, Acost contains the minimum cost of a capacitated vertex cover in A_t complying with f (obtained from information stored in the data tables for A_t), while $X(f)$ contains vertices of X_t which must be in the capacitated vertex cover for this choice of f.

We now construct an instance I_f of REDUCED CVC JOIN for each f as follows. We remove all nodes in A_t, remove all edges with an endpoint outside of Y_t, and remove all edges with both endpoints in X_t. We add a single child t'' with $\mathsf{adh}(t'') = 0$ and $\mathcal{D}(t'') = \{\mathsf{Acost}(f), \{G[Y_{t''}] \mapsto 0\}\}$ (to carry over information of the cost of all forgotten nodes in A_t), and for each $v \in X(f)$ we increase the capacity of v by 1 and add a pendant vertex v' adjacent to v with capacity 0 (ensuring that v must be taken into the vertex cover in I_f). Then for each $v \in X(f)$ we reduce its capacity by the number of edges mapped to v by f (if this would reduce its capacity to negative values, we set its capacity to 0 and attach a 0-capacity pendant vertex to v; this ensures the nonexistence of a capacitated vertex cover for I_f). We adapt the tree-cut

decomposition accordingly by adding the new pendant vertices into new bags in B_t. Notice that I_f now has the desired properties of REDUCED CVC JOIN.

Since its adhesion is zero, the solution of I_f contains a single tuple $(a_{I_f}, \{U_{f,t} \mapsto 0\})$. It remains to show how one can use this tuple to obtain the records for the original instance of CVC JOIN. For each graph H in $\mathcal{Q}_t$, we apply brute-force enumeration over $\mathcal{F}$ to compute the set $\mathcal{F}_H$ of all elements of $\mathcal{F}$ such that $U_{f,t} \cong H$ (intuitively, this corresponds to identifying all the ways for covering the missing edges). We proceed by selecting a function $f \in \mathcal{F}_H$ such that a_{I_f} is minimized and denote this a_{I_f} by $\mathsf{cost}(H)$. For $H = G[Y_t]$, we then set $a_t = \mathsf{cost}(H)$, and we construct β_t by setting, for each H in $\mathcal{Q}_t$ and each $U_{f,t}$, $\beta_t : H \mapsto (\mathsf{cost}(H) - a_t)$.

To argue the claimed running time, observe that one can compute the values of $\mathsf{Acost}(f)$ for all $f \in \mathcal{F}$ in time $2^{\mathcal{O}(k^2)}$ and that the instance I_f for each $f \in \mathcal{F}$ obtained by trivially modifying the graph and the tree-cut decomposition can be constructed in $\mathcal{O}(n)$ time.

We argue the correctness of our reduction by arguing the correctness of the computed value a_t; the same argument then applies analogously also to the correctness of the construction of β_t. Assume that the value a_t computed above is greater than $\mathsf{cvc}(G[Y_t], Y_t) = a'$. Then there exists a capacitated vertex cover C of $G[Y_t]$ of cardinality a' and a mapping of edges f_C which witnesses C. Let $f \in \mathcal{F}$ be the (unique) restriction of f_C to the set of edges mapped by functions in $\mathcal{F}$, and for each $t' \in A_t$ we let $C_{t'}$ be the subset of C which intersects $Y_{t'}$. By the correctness of the data tables for A_t, it holds that $\sum_{t' \in A_t} |C_{t'}| = \mathsf{Acost}(f)$. Furthermore, each $v \in X(f)$ must occur in C by f_C, and also in any solution to the instance I_f due to the added pendant vertex. We would hence necessarily conclude that $a_{I_f} > C \setminus (X(f) \cup \bigcup_{t' \in A_t} C_{t'})$, which, however, contradicts the correctness of the solution to I_f.

On the other hand, assume that $a_t < a'$. Then we can construct a capacitated vertex cover C of $G[Y_t]$ of cardinality a_t from $X(f), f$, and the partial vertex covers which witness the correctness of the solution to I_f. □

LEMMA 4.7. *There exists an algorithm which solves* REDUCED CVC JOIN *in time* $k^{\mathcal{O}(k^2)} \cdot (|B_t| + 1)$.

Proof. Since $K_t = \emptyset$, it suffices to compute $\mathsf{cvc}(G[Y_t], Y_t)$ and output $\mathcal{D}(t) = \{\mathsf{cvc}(G[Y_t], Y_t), G[Y_t] \mapsto 0\}$. We branch over all the at most 2^k possible sets $X' \subseteq X_t$, representing possible intersections of the capacitated vertex cover with X_t. For each such X', we construct an ILP formulation which computes the minimum capacitated vertex cover in $G[Y_t]$ which intersects with X_t in X'. We begin by having a constant c_x contain the capacity of x for every $x \in X'$ and let $c_x = 0$ for $x \in X_t \setminus X'$.

We define a relation $\equiv$ on B_t. For $t_1, t_2 \in B_t$, we say $t_1 \equiv t_2$ if (i) $N(Y_{t_1}) = N(Y_{t_2})$, and (ii) there exists a bijection ϕ from $\mathcal{Q}_{t_1}$ to $\mathcal{Q}_{t_2}$ which satisfies the following two conditions for every $H_1 \in \mathcal{Q}_{t_1}$ and $H_2 \in \mathcal{Q}_{t_2}$ with $H_2 = \phi(H_1)$:
  1. $deg_{H_1}(x) = deg_{H_2}(x)$ for every $x \in N(Y_{t_1})$.
  2. $\beta_{t_1}(H_1) = \beta_{t_2}(H_2)$.
It is easy to see that $\equiv$ defines an equivalence relation. We denote by $[\equiv]$ the set of equivalence classes of $\equiv$ and note that $|[\equiv]| \leq \mathcal{O}(k^2)$. Indeed, there are only $\mathcal{O}(k^2)$ subsets of X_t containing at most two vertices. Moreover, Lemma 4.3 guarantees $\beta_t(H) \in [2] \cup \{\infty\}$ for $H \in \mathcal{Q}_t$ for any $t \in B_t$. Hence, the number of ways that $H \in \mathcal{Q}_t$ can be mapped to $\mathbb{N}_0$ under β_t is bounded by a constant, and thus $|[\equiv]| \leq \mathcal{O}(k^2)$.

Given an equivalence class $\nu \in [\equiv]$, let us fix a node $t_\nu \in \nu$ and let H_ν be the graph $(Y_{t_\nu} \cup N(Y_{t_\nu}), E_{t_\nu} \cup K_{t_\nu})$. For $H \in \mathcal{Q}_{t_\nu}$, we create a variable z_H. This variable shall denote the number of nodes t' in ν such that among the edges in $K_{t'}$ exactly

$deg_H(x)$ many edges can be covered by $Y_{t'}$ for each $x \in N(Y_{t'})$ while incurring the cost $\beta_{t_\nu}(H)$. Our ILP instance is as follows, for which an informal explanation is added in the subsequent paragraphs:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{H \in \mathcal{Q}_{t_\nu}, t_\nu \in \nu, \nu \in [\equiv]} \beta_{t_\nu}(H) \cdot z_H + |X'| \\
\text{subject to} \quad & \sum_{H \in \mathcal{Q}_{t_\nu}, t_\nu \in \nu} z_H = |\nu| & \forall \nu \in [\equiv], \\
& \sum_{H \in \mathcal{Q}_{t_\nu}} (deg_{H_\nu}(x) - deg_H(x)) \cdot z_H \leq c_x & \forall x \in X_t,
\end{aligned}
$$

where z_H is a nonnegative integer variable for every $H \in \mathcal{Q}_{t_\nu}$, $\nu \in [\equiv]$.

Notice that due to the definition of $\equiv$ any two nodes $t_1, t_2 \in \nu$ covering the same number of edges incident with $x \in N(Y_{t_\nu})$ will incur the same cost. This justifies expressing the cost $\mathsf{cvc}(G[Y_t], Y_t)$ with an objective function as above. Since $|\mathcal{Q}_{t_\nu}| \leq 4$ and we create the variables z_H for each $H \in \mathcal{Q}_{t_\nu}$ for each equivalence class $\nu \in [\equiv]$, the total number of integer variables created is in $\mathcal{O}(k^2)$, and so an optimal solution can be obtained by Theorem 2.2 in time $k^{\mathcal{O}(k^2)} \cdot |B_t|$.

Overall, the formulation relies on grouping interchangeable children into equivalence classes and uses variables to denote the number of children in each class for which we use the same assignment of edges between X_t and $Y_{t'}$. For instance, if there is only a single edge to a specific $x \in X_t$, variable z_{H_α} (with H_α being the graph containing the edge incident to x) contains the number of children where the edge is mapped to x while z_{H_β} (with H_β being the graph without the edge incident to x) contains the number of children where this edge is mapped to $Y_{t'}$. Constraint 1 ensures that the variables indeed capture a partition of children in i, while constraint 2 ensures that the capacities in X_t are not exceeded. The instance minimizes the increase in cardinality of capacitated vertex covers over all $Y_{t'}$ by multiplying the number of children with their respective value of β based on the chosen assignment of edges.

Let $\mathcal{I}$ be the minimum value of $\sum_{H \in \mathcal{Q}_{t_\nu}, \nu \in [\equiv]} \beta_{t_\nu}(H) \cdot z_H + |X'|$ over all ILP instances described above; if no solution exists or if a solution is larger than the preset (large) constant representing $\beta_{t'}(H) = \infty$, then we set $\mathcal{I} = \infty$. We argue that $\mathsf{cvc}(G[Y_t], Y_t) = \sum_{t' \in B_t} a_{t'} + \mathcal{I}$, from which the proof follows.

First, consider for a contradiction that $\mathsf{cvc}(G[Y_t], Y_t) < \sum_{t' \in B_t} a_{t'} + \mathcal{I}$. Then there exists a capacitated vertex cover in $G[Y_t]$ of cardinality $\mathsf{cvc}(G[Y_t])$. Let X' be its intersection with X_t, and let f be its witness function. Let $U_{f,t'}$ be the graph on the vertex set $Y_{t'} \cup N(Y_{t'})$ with an edge set $E_{t'} \cup \{e \in K_{t'} : f(e) \in Y_{t'}\}$. Note that $U_{f,t'} \in \mathcal{Q}_{t'}$. For each equivalence class $\nu \in [\equiv]$, we again partition ν into sets so that $t_1, t_2 \in \nu$ belong to the same set if and only if $\phi(U_{f,t_1}) = U_{f,t_2}$. Here ϕ is the bijection between any two nodes $t_1, t_2 \in \nu$ ensuring the equivalence of t_1 and t_2. By fixing a node $t_\nu \in \nu$, each set can be uniquely "represented" by some $H \in \mathcal{Q}_{t_\nu}$ due to the bijective mapping. For each $H \in \mathcal{Q}_{t_\nu}$, let $\tilde{z}_H$ denote the cardinality of the set represented by H. One can check that $z_H = \tilde{z}_H$, $\forall H \in \mathcal{Q}_{t_\nu}, \forall \nu \in [\equiv]$, is a feasible solution for the above ILP yielding an objective value $\mathsf{cvc}(G[Y_t], Y_t)$, contradicting the optimality of $\mathcal{I}$.

On the other hand, consider the case $\mathsf{cvc}(G[Y_t], Y_t) > \sum_{t' \in B_t} a_{t'} + \mathcal{I}$. Then one could construct a better capacitated vertex cover for $G[Y_t]$ by having a capacitated vertex cover intersect X_t in the X' used to obtain $\mathcal{I}$ and using a witness function f which maps edges between X_t and B_t based on the partition of the equivalence classes discovered by the ILP formulation in order to reach $\mathcal{I}$. Again, one can check that the

resulting capacitated vertex cover would have cardinality exactly $\sum_{H \in \mathcal{Q}_{t_\nu}, \nu \in [\equiv]} \beta(H) \cdot z_H + |X'|$. □

By combining Lemma 4.6 with Lemma 4.7, we obtain the following.

COROLLARY 4.8. *There exists an algorithm which solves* CVC JOIN *in time* $k^{\mathcal{O}(k^2)} \cdot n$.

THEOREM 4.9. *tcw-CVC can be solved in time* $k^{\mathcal{O}(k^2)} \cdot n^2$.

Proof. We use Theorem 3.4 to transform $(T, \mathcal{X})$ into a nice tree-cut decomposition with at most $2n$ nodes. We then use Lemma 4.5 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 4.8. The total running time per node is at most $n \cdot k^{\mathcal{O}(k^2)}$, and there are $\mathcal{O}(n)$ many nodes. 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]. It was shown to be equivalent to GRAPH CLEANING [21] and was studied in the parameterized setting [32, 12]. The problem is FPT when parameterized by **degtw** [32]. 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 $\triangleleft_R(v)$ and $\triangleright_R(v)$ 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 by $\mathsf{imb}_R(v)$, is then defined as the absolute value of $\triangleright_R(v) - \triangleleft_R(v)$, and the imbalance of R, denoted by imb_R, is equal to $\sum_{v \in V(G)} \mathsf{imb}_R(v)$.

---

tcw-IMBALANCE (tcw-IMB)
Instance: An n-vertex graph $G = (V, E)$ with $|V| = n$, 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 $\mathsf{imb}_R \leq d$.

4.2.1. Data table. 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 $\mathsf{adh}_t(v)$ 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 $\tau(v) \in \{-\infty, -\mathsf{adh}_t(v), -\mathsf{adh}_t(v) + 1, \ldots, \mathsf{adh}_t(v), \infty\}$ for every $v \in \partial(Y_t)$. We then call a tuple of the form (f, τ) an *extract* (of Y_t or, simply put, of t) and let $\mathcal{L}_t$ denote the set of all extracts (for nodes with adhesion at most k) of t; when the node t is clear from the context, we will use $\mathcal{L}$ as shorthand for $\mathcal{L}_t$. The extract $\alpha = (f, \tau)$ is realized in Y_t (by R) if there exists a linear order R of Y_t such that the following hold:

1. R is a linear superorder of f, and
2. for each $v \in \partial(Y_t)$ we have the following:
   - if $\tau(v) \in \mathbb{Z}$, then $\mathsf{imb}_R(v) = \tau(v)$;
   - if $\tau(v) = -\infty$, then $\rhd_R(v) - \lhd_R(v) \leq -\mathsf{adh}_t(v) - 1$;
   - if $\tau(v) = \infty$, then $\rhd_R(v) - \lhd_R(v) \geq \mathsf{adh}_t(v) + 1$.

The cost of a realized extract α, denoted by $c(\alpha)$, is the minimum value of $\sum_{v \in Y_t} \mathsf{imb}_R(v)$ 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 4.10. $\mathcal{D}(t) = (a_t, \beta_t)$, 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 $4e_t$. We justify this below.

LEMMA 4.11. Let $G = (V, E)$ be a graph, and let t be a node in a width-k tree-cut decomposition $(T, \mathcal{X})$ of G. Let α_1, α_2 be two extracts of Y_t such that $c(\alpha_1) > 4e_t + c(\alpha_2)$. Then, for any linear order R_1 of Y_t which realizes α_1 and any linear superorder R (over V) of R_1, it holds that there exists a linear order R' over V such that $\mathsf{imb}_{R'} < \mathsf{imb}_R$.

Proof. Consider any linear order R_2 (over Y_t) which realizes α_2 of cost $c(\alpha_2)$; since $c(\alpha_2) \neq \infty$, there must exist at least one such R_2. Let R be any linear order constructed above; clearly, there exists a unique linear order R^* over $V \setminus Y_t$ which is a linear suborder of R. Consider the linear order R' obtained by the concatenation of R_2 and R^*. Let I^* denote the imbalance of R^* in $G[V \setminus Y_t]$, and recall that $c(\alpha_2)$ denotes the imbalance of R' in $G[Y_t]$. Since the addition of e_t edges can only increase or decrease the imbalance by at most $2e_t$, we obtain that $\mathsf{imb}_{R'} \leq I^* + c(\alpha_2) + 2e_t$ and $\mathsf{imb}_R \geq I^* + c(\alpha_1) - 2e_t$. Altogether we conclude $\mathsf{imb}_{R'} \leq I^* + c(\alpha_2) + 2e_t < I^* + c(\alpha_1) - 2e_t \leq \mathsf{imb}_R$. $\qquad \square$

The following observation establishes a bound on the size of our records. Moreover, it will be useful to note that for a fixed value of the adhesion one can efficiently enumerate all the functions that may appear in the records (this will facilitate the processing of thin nodes).

OBSERVATION 2. *The cardinality of $\mathcal{L}$ is bounded by $k^{\mathcal{O}(k)}$. Moreover, for each node t with adhesion κ, the number of possible functions β_t is bounded by $\kappa^{\kappa^{\mathcal{O}(\kappa)}}$, and these may be enumerated in the same time.*

4.2.2. Initialization and termination.

LEMMA 4.12. *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 $k^{\mathcal{O}(k)}$.*

Proof. We can branch over all at most k^k linear orders of Y_t, and for each we can compute its imbalance in $G[Y_t]$ in time $\mathcal{O}(k)$. This already gives sufficient information to construct $\mathcal{D}(t)$. $\qquad \square$

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 $a_r \leq d$.*

4.2.3. 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. We formalize this problem below.

> IMB JOIN
> *Instance*: A tcw-IMB instance (G, d), a nonleaf node t of a width-k nice tree-cut decomposition $(T, \mathcal{X})$ of G, and $\mathcal{D}(t')$ for each child t' of t.
> *Parameter*: k.
> *Task*: Compute $\mathcal{D}(t)$.

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.

> REDUCED IMB JOIN
> *Instance*: A tcw-IMB instance consisting of a graph G and an integer d, a root t of a nice tree-cut decomposition $(T, \mathcal{X})$ of G such that $A_t = \emptyset$ and $G[X_t]$ is edgeless, $\mathcal{D}(t')$ for each child t' of t, a linear order f of X_t, a mapping $\omega : X_t \to \mathbb{Z}$, and a set ζ of linear constraints on the value of $\rhd(v) - \lhd(v)$ for a subset of X_t.
> *Parameter*: $k = |X_t|$.
> *Task*: Compute the minimum value of $\left(\sum_{v \in Y_t \setminus X_t} \mathsf{imb}_R(v) \right) + \left(\sum_{v \in X_t} | \rhd_R (v) - \lhd_R(v) + \omega(v)| \right)$ over all linear superorders R of f over Y_t which satisfy ζ, or correctly determine that no such R exists.

LEMMA 4.13. *There is an FPT Turing reduction from* IMB JOIN *to* $k^{\mathcal{O}(k^2)}$ *instances of* REDUCED IMB JOIN *which runs in time* $k^{\mathcal{O}(k^2)} \cdot n$.

Proof. Our goal is to use the data contained in each $\mathcal{D}(t')$ to preprocess all information in the nodes A_t, leaving us with a set of REDUCED IMB JOIN instances. We branch over the at most $k^{\mathcal{O}(k)}$ extracts (of t) in $\mathcal{L}$. The remainder of the proof shows how to compute $c(\alpha)$ for each extract $\alpha = (f, \tau) \in \mathcal{L}$ using an oracle which solves REDUCED IMB JOIN. Indeed, it follows from the definition of $\mathcal{D}(t)$ that this information is sufficient to output $\mathcal{D}(t)$ (in $\mathcal{O}(k)$ time).

Let $Z = \{ v \in Y_t \mid v \in X_t$ or $\exists t' \in A_t : v \in \partial(Y_{t'}) \}$; by Lemma 4.1 and the bound on the width of $(T, \mathcal{X})$, it follows that $|Z| \leq 2k^2 + 2k$. Let $\mathcal{J}$ then denote the set containing all the at most $k^{\mathcal{O}(k^2)}$ linear orders of Z. Next, we prune $\mathcal{J}$ to make sure it is compatible with our α; specifically, we prune $\mathcal{J}$ by removing any linear order which is not a linear superorder of f. We now branch over $\mathcal{J}$; let j be a fixed linear order in $\mathcal{J}$.

For each $t' \in A_t$, we compute the set $\delta(t')$ of extracts (of t') which are "compatible" with our chosen extract $\alpha = (f, \tau)$ (of t) and our chosen j. If $Y_{t'} \cap \partial(Y_t) = \emptyset$, then $\delta(t') = \mathcal{L}$. However, if $Y_{t'} \cap \partial(Y_t) =: Q \neq \emptyset$, then we let $\delta(t')$ contain an extract $(f', \tau') \in \mathcal{L}$ if and only if the following holds. For every $v \in Q$ such that $\tau(v) = \infty$, it holds that either $\tau'(v) = \infty$ or $\tau'(v) + \rhd_j(v) - \lhd_j(v) \geq \mathsf{adh}_t(v) + 1$. For every $v \in Q : \tau(v) = -\infty$, it holds that either $\tau'(v) = -\infty$ or $\tau'(v) + \rhd_j(v) - \lhd_j(v) \leq -\mathsf{adh}_t(v) - 1$. For every $v \in Q$ such that $\tau(v) = i$, where $i \neq \infty$, it holds that $i = \tau'(v) + \rhd_j(v) - \lhd_j(v)$.

Finally, we branch over all choices of compatible extracts for each $t' \in A_t$. Specifically, we branch over all choices of λ, where $\lambda(t') = \alpha' \in \delta(t')$ for every $t' \in A_t$. For each λ, we compute the total imbalance imb_λ in A_t, specifically the sum of the imbalances of vertices in $Y_{t'}$ for every $t' \in A_t$ excluding edges which have an endpoint outside of Y_t. This can be computed as follows: $\mathsf{imb}_\lambda = \sum_{t' \in A_t} (a_{t'} + \beta(\lambda(t'))) + K$,

where K is the total adjustment of the imbalance caused by edges between X_t and some $Y_{t'}$ (note that K may be negative). K can be computed by the following simple procedure. Initialize with $K := 0$ and $\tau'' := \tau'$, and for each edge ab where $a \in X_t$ and $b \in Y_{t'}$ such that $\lambda(t') = (f', \tau')$ we proceed by the following cases:

- if $\tau''(b) > 0$ and $j(a) < j(b)$, then $K := K - 1$ and $\tau''(b) := \tau''(b) - 1$;
- if $\tau''(b) \geq 0$ and $j(b) < j(a)$, then $K := K + 1$ and $\tau''(b) := \tau''(b) + 1$;
- if $\tau''(b) \leq 0$ and $j(a) < j(b)$, then $K := K + 1$ and $\tau''(b) := \tau''(b) - 1$;
- if $\tau''(b) < 0$ and $j(b) < j(a)$, then $K := K - 1$ and $\tau''(b) := \tau''(b) + 1$.

The same procedure as the one above can be used to also compute the value $\rhd_j(v) - \lhd_j(v)$ for each $v \in X_t$, which we store as $\omega(v)$. From τ, we obtain a set of constraints ζ which will guarantee that the solution to REDUCED IMB JOIN will be compatible with α. Specifically, for each $v \in X_t \cap \partial(Y_t)$ such that $\tau(v) = \infty$, we set $\zeta(v) \geq \mathsf{adh}_t(v) + 1$; if $\tau(v) = -\infty$, then we set $\zeta(v) \leq -\mathsf{adh}_t(v) - 1$; and if $\tau(v) \in \mathbb{Z}$, then we set $\zeta(v) = \tau(v)$. As for f, we obtain it as the unique linear suborder of j restricted to X_t. This completes the construction of our REDUCED IMB JOIN instance $I_{\alpha,j,\lambda}$.

In total, we have branched over $k^{\mathcal{O}(k)}$ extracts for t, $k^{\mathcal{O}(k^2)}$ linear orders of Z, and $k^{\mathcal{O}(k)}$ choices of λ, resulting in a total of $k^{\mathcal{O}(k^2)}$ instances. What remains now is to show how the solution of each instance $I_{\alpha,j,\lambda}$ can be used to solve IMB JOIN. Let $i_{\alpha,j,\lambda}$ denote the output of $I_{\alpha,j,\lambda}$. Then we set $c(\alpha)$ to the minimum value of $i_{\alpha,j,\lambda} + \mathsf{imb}_\lambda$ over all choices of α, j, λ.

In conclusion, we argue correctness. For a contradiction, assume that there exist some α, j, λ such that $i_{\alpha,j,\lambda} + \mathsf{imb}_\lambda < c(\alpha)$. By our construction, there then exists a linear order R_1 of $X_t \cup \bigcup_{t' \in A_t} v \in Y_{t'}$ such that the imbalance of all vertices in $\bigcup_{t' \in A_t}$ (induced by edges of $G[Y_t]$ with at least one endpoint in $\bigcup_{t' \in A_t} v \in Y_{t'}$) is equal to imb_λ, and furthermore this R_1 satisfies the "requirements" of the extract α on the imbalance of vertices in $\partial(Y_t) \setminus X_t$. By the construction of the instance $I_{\alpha,j,\lambda}$, there also exists a linear order R_2 of $X_t \cup \bigcup_{t' \in B_t} v \in Y_{t'}$ where the order of vertices in X_t is the same as in R_1, and hence it is possible to merge R_1 and R_2 into a linear order R over Y_t by using vertices in X_t as "anchoring points" (R preserves the order of the anchoring points and the order of vertices within R_1 and of vertices within R_2, and the order between a nonanchoring vertex in R_1 and a nonanchoring vertex in R_2 is irrelevant). One can straightforwardly verify that the linear constraints ζ in $I_{\alpha,j,\lambda}$ ensure that R realizes the extract α. The cost of R is then equal to the imb_λ plus the imbalance of vertices in X_t and the imbalance of vertices in $\bigcup_{t' \in B_t} Y_{t'}$. The mapping ω transfers the information on the imbalance of vertices in X_t caused by edges in $G[X_t \cup \bigcup_{t' \in A_t} Y_{t'}]$ into $I_{\alpha,j,\lambda}$, and from there on one can verify that the imbalance of R in vertices in $X_t \cup \bigcup_{t' \in B_t} Y_{t'}$ sums up to $i_{\alpha,j,\lambda}$. Hence, this linear order R contradicts the value of $c(\alpha)$.

On the other hand, assume $i_{\alpha,j,\lambda} + \mathsf{imb}_\lambda > c(\alpha)$ for all j, λ. Let R be a linear order which realizes α; by reversing the merging procedure outlined in the previous paragraph, one can decompose R into R_1 (over $X_t \bigcup_{t' \in A_t} v \in Y_{t'}$) and R_2 (over $X_t \cup \bigcup_{t' \in B_t} v \in Y_{t'}$). Let j be the unique linear suborder of R_1 over Z, and let λ be the unique tuple of extracts capturing R_1 on individual children $t' \in A_t$. Then, by our assumption on the size of $c(\alpha)$, either the imbalance of R_1 over $\bigcup_{t' \in A_t} v \in Y_{t'}$ is greater than imb_λ, or the imbalance of R_2 over $X_t \cup \bigcup_{t' \in B_t} v \in Y_{t'}$ (additionally counting edges between X_t and $\bigcup_{t' \in A_t}$) is greater than $i_{\alpha,j,\lambda}$. However, the first can be ruled out by the computation of imb_λ from λ, j, α, while the second is impossible since one can plug in R_2 into $I_{\alpha,j,\lambda}$ to achieve a solution value of $i_{\alpha,j,\lambda}$.

We conclude that $i_{\alpha,j,\lambda} + \mathsf{imb}_\lambda = c(\alpha)$, and hence by iterating over all values of α and given the correct value of $i_{\alpha,j,\lambda}$ it is possible to construct $\mathcal{D}(t)$ in time $k^{\mathcal{O}(k^2)} \cdot |V(G)|$. $\square$

LEMMA 4.14. *There exists an algorithm which solves* REDUCED IMB JOIN *in time* $k^{\mathcal{O}(k^4)} \cdot (|B_t| + 1)$.

Proof. We again use an ILP formulation, but this time we first need to get rid of the absolute values over $\rhd_R(v) - \lhd_R(v) + \omega(v)$. Since the number of vertices v in X_t is bounded by k, we can exhaustively branch over whether each $\rhd_R(v) - \lhd_R(v) + \omega(v)$ ends up being negative or nonnegative and force this choice in our ILP instance by suitable constraints. Formally, let us branch over all the at most 2^k possible functions $\phi : v \in X_t \to \{1, -1\}$. For each fixed ϕ, we construct an ILP instance I_ϕ which outputs the minimum value of $\big(\sum_{v \in Y_t \setminus X_t} \mathsf{imb}_R(v)\big) + \big(\sum_{v \in X_t} \phi(v) \cdot (\rhd_R(v) - \lhd_R(v) + \omega(v))\big)$ (over all R which satisfy the required conditions); let this value be i_ϕ. Then the solution of REDUCED IMB JOIN can be correctly computed as $\min_{\phi : v \in X_t \to \{1,-1\}} i_\phi$.

Before proceeding, we mention a few important observations. First, each vertex $v \in Y_{t' \in B_t}$ can be placed in one of the at most $k + 1$ intervals in the linear order between the placement of vertices in X_t. Second, since there are no edges between any $Y_{t' \in B_t}$ and $Y_{t'' \in B_t}$, the order among vertices in different children of t cannot change the value of i_ϕ; only the order between the neighbors of X_t and X_t itself matters. Third, for each $t' \in B_t$, we know that the minimum sum of imbalances in $Y_{t'}$ over all linear orders is $a_{t'}$, and by Lemma 4.11 it suffices to only consider extracts γ of $Y_{t'}$ such that $\beta_t(\gamma) \neq \infty$.

We now describe the construction of an ILP instance which outputs i_ϕ, as well as its relation to a linear order R, so as to facilitate the correctness argument. Let $\mathcal{L}'_0, \mathcal{L}'_1, \mathcal{L}'_2$ denote the set of all extracts of nodes t' where $|\partial(Y_{t'})| = 0, 1, 2$, respectively. Observe that while each of the sets $\mathcal{L}'_1, \mathcal{L}'_2$ may contain a number of extracts that cannot be bounded by any function of our parameter (since each node formally has its own, unique extract) we may identify and define equivalences that group the extracts of nodes which can affect their neighborhood in exactly the same way. In particular, analogously as in the proof of Lemma 4.7 we say that extract $(f_a, \tau_a) \in \mathcal{L}'_i$ of a node a is *equivalent* to extract $(f_b, \tau_b) \in \mathcal{L}'_i$ of a node b, $i \in \{1, 2\}$ if there exists a bijective mapping ϕ from $\partial(Y_a)$ to $\partial(Y_b)$ such that (1) $N(v) \setminus Y_a = N(\phi(v)) \setminus Y_b$ for every $v \in \partial(Y_a)$, (2) $u <_{f_a} v$ if and only if $\phi(u) <_{f_b} \phi(v)$ for every $u, v \in \partial(Y_a)$, and (3) $\tau_a(v) = \tau_b(\phi(v))$ for every $v \in \partial(Y_a)$. In other words, two extracts are equivalent if there is a canonical renaming function between vertices in the boundary of Y_a and Y_b which preserves the neighborhoods outside and transforms the extract (f_a, τ_a) into (f_b, τ_b). We can now set $\mathcal{L}_0 = \mathcal{L}'_0$, while $\mathcal{L}_1 \subseteq \mathcal{L}'_1$ and $\mathcal{L}_2 \subseteq \mathcal{L}'_2$ are obtained by keeping precisely one representative extract for each equivalence class that appears in $\mathcal{L}_1$ and $\mathcal{L}_2$, respectively (in other words, all but one arbitrarily selected extract in each equivalence class is deleted). It can be observed that the size of each such $\mathcal{L}_j$ is upper-bounded by $\mathcal{O}(k^2)$.

Next, we let the set S_0 contain all the constantly many mappings $\beta_0 : \mathcal{L}_0 \to [0] \cup \{\infty\}$. Similarly, we let $S_{1,1}$ contain all $\beta_1 : \mathcal{L}_1 \to [4] \cup \{\infty\}$ (extracts in S_1 can appear in nodes of adhesion 1 or 2), and we let $S_{1,2}$ contain all $\beta_1 : \mathcal{L}_1 \to [8] \cup \{\infty\}$. For the last case, we let S_2 contain all $\beta_2 : \mathcal{L}_2 \to [8] \cup \{\infty\}$. We let

- $\&_{1,1} = S_{1,1} \times X_t$,
- $\&_{1,2} = S_{1,2} \times X_t \times X_t$,
- $\&_{2,1} = S_2 \times X_t$,
- $\&_{2,2} = S_2 \times X_t \times X_t$.

The set $\& = S_0 \cup \&_{1,1} \cup \&_{1,2} \cup \&_{2,2} \cup \&_{2,1}$ will serve as the set of all possible "types" of nodes $t' \in B_t$, while the set $S = S_0 \cup S_{1,1} \cup S_{1,2} \cup S_2$ will contain all possible mappings β. For each $\beta \in S_{1,2} \cup S_2$, nodes t' have two neighbors in X_t; these will in general not be symmetric with respect to β, and hence we distinguish them by denoting one as $N_1(t')$ and the other as $N_2(t')$. Before proceeding, note that $|\&| \in \mathcal{O}(k^2)$.

For each $\pi \in \&$, we introduce variables which capture information on the distribution of the (at most two) vertices in $\partial(Y_{t'})$ among the $k + 1$ intervals of f. We formalize for the most general case of 2 vertices in $\partial(Y_{t'})$ and 2 neighbors in X_t, i.e., for $\&_{2,2}$; the remaining three cases are simplifications of this one and require fewer variables. For each $\pi = (\beta, x_1, x_2) \in \&_{2,2}$, we define $Q_\pi = \{\, t' \in B_t \mid \exists a_{t'} : \mathcal{D}(t') = (a_{t'}, \beta), N_1(t') = x_1, N_2(t') = x_2 \,\}$. We let $q_\pi = |Q_\pi|$; i.e., q_π is the number of nodes of type π. For a node t', let $N(x_1) \cap \partial(Y_{t'})$ be denoted by y_1, and analogously $N(x_2) \cap \partial(Y_{t'}) = y_2$. Now we introduce (for this π) $(k + 1) \cdot (k + 2)$ new variables:

- for each $0 \leq i, j \leq k$, if $i \neq j$, we introduce a variable $g_i^{h_j^\pi}$, which captures the number of nodes t' of type π such that y_1 occurs after exactly i vertices of X_t in R and y_2 occurs after exactly j vertices of X_t in R;
- if $i = j$, we introduce two new variables $g_i^{<h_j^\pi}$ and $g_i^{>h_j^\pi}$, which capture the number of pairs of vertices such that both y_1 and y_2 occur after exactly i vertices of X_t and y_1 occurs, respectively, before or after y_2.

In total, we have introduced $\mathcal{O}(k^4)$ variables. Next, we insert the following constraints:

1. For each $\pi \in \&$, we make sure that the variables indeed correspond to a partition of the set Q_π. Specifically, we make sure that each variable is greater than or equal to 0 and that the sum of all variables associated with π is equal to q_π.

2. We add a constraint for each linear constraint in ζ on the value of $\rhd(v) - \lhd(v)$, where $v \in X_t$. This value of $\rhd(v) - \lhd(v)$ can be expressed by a linear combination over our variables, since each variable is associated with a fixed position (left or right) from each vertex in X_t and a type, which in turn contains information on whether the variable is adjacent or not to each $v \in X_t$.

3. We make sure that we only consider parameter values consistent with ϕ. Specifically, if $\phi(v) = 1$ (nonnegative), then we need to ensure that $\rhd_R(v) - \lhd_R(v) + \omega(v) \geq 0$, while in the other case we need to ensure that $\rhd_R(v) - \lhd_R(v) + \omega(v) < 0$. Each of these conditions can be expressed as a linear constraint over our variables, similarly as when encoding ζ.

The constructed ILP instance can be solved in time at most $k^{\mathcal{O}(k^4)} \cdot (|B_t| + 1)$ by Theorem 2.2.

Let a denote $\sum_{t' \in A_t} a_{t'}$. For each variable var corresponding to a placement of a type π in the at most 2 intervals, one can compute from $\beta \in \pi$ and the order between y_1, y_2, x_1, x_2 a value $\mathsf{cost}(var)$ in $[8] \cup \infty$ which expresses the additional imbalance in $Y_{t'}$ caused by this particular placement. Then our instance will minimize the expression $val(\phi) = a + \sum_{v \in X_t} \phi \cdot (\rhd_R(v) - \lhd_R(v) + \omega(v)) + \sum_{var} \mathsf{cost}(var)$. After branching through all possible values of ϕ, we output the minimum over all computed $val(\phi)$. $\Box$

COROLLARY 4.15. *There exists an algorithm which solves* IMB JOIN *in time* $k^{\mathcal{O}(k^4)} \cdot n$.

Now the proof of the theorem below is then analogous to the proof of Theorem 4.9.

THEOREM 4.16. *tcw*-IMB *can be solved in time* $k^{\mathcal{O}(k^4)} \cdot n^2$.

Proof. We use Theorem 3.4 to transform $(T, \mathcal{X})$ into a nice tree-cut decomposition

with at most $2n$ nodes. We then use Lemma 4.12 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 4.15. The total running time per node is dominated by $n \cdot k^{\mathcal{O}(k^4)}$, and there are at most $2n$ nodes. Once we obtain $\mathcal{D}(r)$, we can correctly output by Observation 3. $\square$

4.3. Capacitated dominating set. CAPACITATED DOMINATING SET is a generalization of the classical DOMINATING SET problem by the addition of vertex capacities. Aside from its applications (discussed, for instance, in [30]), CAPACITATED DOMINATING SET has been targeted by parameterized complexity research in the past also because it is a useful tool for obtaining W-hardness results [15, 13, 3]. It is known to be W[1]-hard when parameterized by treewidth [8].

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 $c(v)$. Such a mapping f is said to *witness D*, and f is a witness function of D. We formally define the problem below.

> 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.

For a vertex $v \in \partial(Y_t)$, recall that $\mathsf{adh}_t(v) = |N(v) \setminus Y_t|$ and $\mathsf{adh}(t) = \sum_{v \in \partial(Y_t)} \mathsf{adh}_t(v)$.

We now give a high-level description of our algorithm. In a "snapshot" of a capacitated dominating set at a node t of the decomposition, a vertex in $\partial(Y_t)$ may have two possible states: *active* or *passive*. The interpretation is as follows: a vertex $v \in \partial(Y_t)$ is

- active if it will either be in the dominating set or will be dominated by a vertex in Y_t and
- passive if it will be dominated by a vertex outside of Y_t.

We will use a data table $\mathcal{D}(t)$ where each entry stores information about the states of vertices in $\partial(Y_t)$, i.e., the vertices which have neighbors outside of Y_t. Moreover, the snapshot also contains information about the residual capacity of each vertex in $\partial(Y_t)$ that is *active*; in particular, each such vertex v in $\partial(Y_t)$ may have up to $\mathsf{adh}_t(v) \le k$ neighbors outside of Y_t, and hence it is important to distinguish whether the residual capacity of v is at least $\mathsf{adh}_t(v)$ (meaning that v can dominate all of its neighbors outside of Y_t) or precisely some number ℓ in $[\mathsf{adh}_t(v) - 1]$ (meaning that v can only dominate up to ℓ of its neighbors outside of Y_t).[3] We store the information about these residual capacities in the form of an "offset." For each such snapshot, we will keep information about the minimum-size capacitated dominating set that corresponds to that snapshot. We formalize below.

4.3.1. Data table. In this subsection, we define the table $\mathcal{D}(t)$ at node t, but before that we will formalize the notion of a *snapshot* (for t).

DEFINITION 4.17. *A snapshot is a tuple* (active, offset) *where* active $\subseteq \partial(Y_t)$ *and* offset *maps each $v \in$ active to an element of* $[\mathsf{adh}_t(v)]$.

---

[3] Notice that active vertices that are not in the dominating set automatically receive a residual capacity of 0 but are otherwise not distinguished from vertices in the dominating set.

For a node $t \in V(t)$, let the *representation* of a snapshot $\sigma = (active, offset)$ be the graph G_σ obtained from $G[Y_t]$ by (1) deleting all vertices in $\partial(Y_t) \setminus active$ and (2) attaching to each vertex $v \in active$ precisely $offset(v)$ many new pendant vertices; we will assume that each new pendant vertex created in this way has a capacity of 0 and refer to these new pendant vertices as *auxiliary vertices*. The *cost* of the snapshot σ, denoted by $cost(\sigma)$, is then the minimum size of a capacitated dominated set over all capacitated dominating sets of G_σ; any capacitated dominating set of G_σ contains no auxiliary vertex. We call such a minimum capacitated dominating set a σ-CDS, and if no such capacitated dominating set exists, then we set $cost(\sigma) = \bot$. The *base cost* of t, denoted by a_t, is then defined as the cost of the snapshot $(\emptyset, \emptyset)$.

The final ingredient we need before defining our data table $\mathcal{D}(t)$ is a bound on the gap between the cost of an arbitrary snapshot and the base cost.

LEMMA 4.18. *For each snapshot $\sigma = (active, offset)$ it holds that either $cost(\sigma) = \bot$, or $a_t \le cost(\sigma) \le a_t + 2\mathsf{adh}(t)$.*

Proof. For the first inequality, it suffices to see that every σ-CDS is also an $(\emptyset, \emptyset)$-CDS. We prove the second inequality by induction on $|active| + \sum_{v \in active} offset(v)$. The proof is a two-step induction, where the first induction step covers the addition of a vertex and the second induction step covers an increase of the offset. For the base case $(\emptyset, \emptyset)$, we observe that the claim clearly holds for $|active| = 0$ by the definition of a_t.

For the first inductive step, assume that the second inequality holds for some $\sigma = (active, offset)$, and let us consider $\sigma' = (active \cup \{v\}, offset \cup \{v \mapsto 0\})$. Then either there exists a σ'-CDS of the same size as a σ-CDS (notably, when there exists a σ-CDS containing v), or for an arbitrary σ-CDS X we have that $X \cup \{v\}$ is a σ'-CDS. In other words, adding a vertex to *active* (with an initial *offset* of 0) only increases the cost of the snapshot by at most 1.

For the second inductive step, assume that the second inequality holds for some $\sigma = (active, offset)$, and let us consider $\sigma' = (active, offset')$, where $offset'(v) = offset(v) + 1$ for one vertex $v \in active$ and $offset'(w) = offset(w)$ for every other vertex $w \in active$. We now distinguish two cases. If $|N_{G_{\sigma'}}(v) \setminus Y_t| > c(v)$, then clearly no σ'-CDS can exist, i.e., $cost(\sigma') = \bot$. Otherwise, we are left with three subcases that are similar to the first induction step:

- either a σ-CDS is also a σ'-CDS, or
- we can take an arbitrary σ-CDS X not containing v and observe that $X \cup \{v\}$ is a σ'-CDS, or
- we can take an arbitrary σ-CDS X containing v and observe that since X is not a σ'-CDS, v must be used to dominate some vertex, say z. We then observe that $X \cup \{z\}$ is a σ'-CDS.

To summarize, we have shown that for every snapshot σ, increasing the *offset* or *active* will only increase the size of a σ-CDS by 1; since both $|active|$ and the sum of offsets is upper-bounded by $\mathsf{adh}(t)$, the lemma follows. $\quad\square$

We can now define our data table.

DEFINITION 4.19. $\mathcal{D}(t) = (a_t, \beta_t)$, *where β_t maps each snapshot of t to a value in $[2\mathsf{adh}(t)] \cup \{\bot\}$ such that for each snapshot σ it holds that $cost(\sigma) = a_t + \beta_t(\sigma)$ (where $\bot$ acts as an absorbing element).*

The correctness of Definition 4.19 follows from Lemma 4.18.

4.3.2. Initialization and termination.

LEMMA 4.20. *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 $2^{\mathcal{O}(k^2)}$.*

Proof. We can branch over all at most $2^k \cdot 2^k$-many snapshots of t, and for each snapshot σ a σ-CDS can be computed by brute force in time at most $2^{\mathcal{O}(k^2)}$. □

For the next observation, it will be useful to note that the root can always be assumed to contain an empty bag (otherwise, one may attach a new root with this property on top of the original one).

OBSERVATION 4. *Let (G, d) be an instance of tcw-CDS, and let r be the root of a nice tree-cut decomposition of G such that $X_r = \emptyset$ and $\mathcal{D}(r) = (a_r, \beta_r)$. Then (G, d) is a yes-instance if and only if $a_r \leq d$.*

4.3.3. 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. We formalize this problem below.

CDS JOIN
Instance: A tcw-CDS instance (G, d), a nonleaf node t of a width-k nice tree-cut decomposition $(T, \mathcal{X})$ of G, and $\mathcal{D}(t')$ for each child t' of t.
Parameter: k.
Task: Compute $\mathcal{D}(t)$.

For a third time, we will 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 CDS JOIN.

REDUCED CDS JOIN
Instance: A tcw-CDS instance (G, d), a nonleaf root t of a width-k nice tree-cut decomposition $(T, \mathcal{X})$ of G such that $A_t = \emptyset$ and $G[X_t]$ is edgeless, $\mathcal{D}(t')$ for each child t' of t, and auxiliary sets $S \subseteq X_t$, $S' \subseteq V(G)$ with the property that $\forall s' \in S' \exists q \in B_t : Y_q = \{s'\}$.
Parameter: k.
Task: Determine whether G admits a capacitated dominating set D of size at most d such that $D \cap X_t = S$ and $D \cap S' = \emptyset$.

LEMMA 4.21. *There is an FPT Turing reduction from CDS JOIN to $2^{\mathcal{O}(k^2)}$ instances of REDUCED CDS JOIN which runs in time $2^{\mathcal{O}(k^2)} \cdot n \cdot d$.*

Proof. In order to compute $\mathcal{D}(t)$ for a CDS JOIN instance, it suffices to be able to compute the *cost* of each snapshot of t. Since the number of snapshots of t is upper-bounded by $2^{\mathcal{O}(k)}$, we can loop over each snapshot in this time and reduce the problem of computing the cost of each individual snapshot to REDUCED CDS JOIN. So let us consider an arbitrary snapshot $\sigma = (\textit{active}, \textit{offset})$ of t.

As our first step, we remove all vertices outside of Y_t (thus turning t into a root). We then branch over each subset $S \subseteq X_t$ with the aim of identifying the intersection between a σ-CDS and X_t; to this end, we only consider subsets S which are "compatible" with *active*, notably by requiring that $S \cap \partial(Y_t) \subseteq \textit{active}$ (i.e., vertices in S that lie on the boundary cannot be passive). Moreover, each vertex x in $X_t \setminus (\partial(Y_t) \cup S)$ must be dominated by a neighbor in Y_t, and we also branch to determine whether x will be dominated by a specific vertex in S or rather by a vertex in $Y_{t'}$ for some unspecified child t' of t. If x is dominated by some $x' \in S$,

then we delete x and reduce the capacity of x' by 1. Afterwards, we delete all edges with both endpoints in X_t since we have at this point predetermined the precise dominating relations within X_t. For a fixed σ, these branching steps require us to consider at most $2^{\mathcal{O}(k^2)}$ many subcases. Observe that in the current branch every vertex in $X_t \setminus (\partial(Y_t) \cup S)$ now must be dominated by a neighbor in Y_t and that every vertex in $\partial(Y_t) \setminus \mathit{active}$ must be left undominated.

Our next and crucial step is aimed at dealing with children in A_t. In particular, for each $p \in A_t$, we branch over each snapshot $\sigma_p = (\mathit{active}', \mathit{offset}')$ of p and check whether σ_p is compatible with S and σ:

- for each $v \in \partial(Y_t) \cap \partial(Y_p)$, it must hold that if $v \notin \mathit{active}$, then $v \notin \mathit{active}'$;[4]
- if $v \in \mathit{active} \cap \mathit{active}'$, then $\mathit{offset}(v) \leq \mathit{offset}'(v)$.

As a subcase, we now deal with vertices $v \in \partial(Y_t) \cap \partial(Y_p)$ such that $v \in \mathit{active}' \setminus \mathit{active}$. The interpretation here is that v cannot be a dominating vertex (since $v \notin \mathit{active}'$) but must be dominated from a neighbor in $Y_t \setminus Y_p$. Since there are at most k such neighbors of v, we use brute force to determine which neighbor dominates v and reduce that neighbor's capacity by 1.

Next, for each $v \in \partial(Y_p) \setminus \partial(Y_t)$, we distinguish the following two cases. If $v \notin \mathit{active}'$, then v must be dominated by a neighbor in $Y_t \setminus Y_p$; there are at most k such neighbors, and we use brute force to determine which one dominates v, whereas for each choice we reduce the dominating vertex's capacity by 1. If $v \in \mathit{active}'$ and $\mathit{offset}'(v) > 0$, then v may be used to dominate some of its (at most k) neighbors in $Y_t \setminus Y_p$, and we use brute force to determine which vertices it dominates and delete the newly dominated vertices. Moreover, for each $v \in \partial(Y_p) \cap \partial(Y_t)$ such that $\mathit{offset}'(v) - \mathit{offset}(v) = \ell$, v can be used to dominate up to ℓ of its neighbors in $Y_t \setminus Y_p$; we once again brute-force to determine which and delete these newly dominated vertices.

Finally, we need to ensure that a solution for our constructed instance of REDUCED CDS JOIN satisfies the requirements specified by the offset component of σ. To this end, for each $v \in \mathit{active}$ we construct $\mathit{offset}(v)$-many new pendant vertices, attach these to v, and add these to the set S' (ensuring that these $\mathit{offset}(v)$-many vertices must be dominated by v). Since these vertices must also be included in the provided nice tree-cut decomposition of G, we simply add each such pendant to its own separate (thin) leaf adjacent to t.

Let us now consider the set $\mathcal{Y}$ of instances of REDUCED CDS JOIN obtained by branching over one particular snapshot σ of t in the original input instance of CDS JOIN. Assume that the cost of σ is δ, as witnessed by a σ-CDS Z with witness function f. Let us now consider the branch where $S = Z \cap X_t$ and the pairs of vertices in a domination relationship inside X_t reflect the witness function f. Similarly, for each $p \in A_t$, consider the branch where we identified the snapshot σ_p that mimics the behavior of Z—in particular by having active' reflect $Z \cap Y_t$ and offset' reflect f. Then, by the correctness of $\mathcal{D}(p)$ and the optimality of Z, it follows that $|Z \cap Y_p| = a_p + \beta_p(\sigma_p)$. Moreover, $|Z \cap X_t| = |S|$. The intersection $Z \cap (\cup_{q \in B_t} Y_q)$ then represents a capacitated dominating set for an instance $Y \in \mathcal{Y}$ (obtained in our reduction) of size $\delta - |S| - (\sum_{p \in A_t} a_p + \beta_p(\sigma_p))$.

On the other hand, let us consider that for some snapshot σ leading to a set $\mathcal{Y}$ of instances of REDUCED CDS JOIN via branching there is an instance $Y \in \mathcal{Y}$ which admits a capacitated dominating set Z' satisfying the stated properties for S and S' of cardinality d'. After adjusting Z' by taking into account the selection of S and

---

[4]On the other hand, it may happen that $v \in \mathit{active} \setminus \mathit{active}'$ since v might be dominated by a vertex in $Y_t \setminus Y_p$.

the domination relations on X_t that led to Y, and similarly by expanding Z' via the σ_p-CDS used in the branching for the individual nodes $p \in A_t$, we can reverse the arguments used in the previous paragraph and show that there is a σ-CDS in the original instance of CDS JOIN of size at most $|Z'| + |S| + (\sum_{p \in A_t} a_p + \beta_p(\sigma_p))$.

To conclude, we have shown that the original instance of CDS JOIN can be solved by computing a minimum capacitated dominating set for each of the at most $2^{\mathcal{O}(k^2)}$ obtained REDUCED CDS JOIN instances, whereas the construction steps for each obtained instance can be carried out in linear time. Such a minimum set can be computed by performing at most d separate calls asking for a capacitated dominating set with the required properties of size at most d. $\qquad\square$

LEMMA 4.22. REDUCED CDS JOIN *can be solved in time* $2^{\mathcal{O}(k \cdot \log k)}|B_t|$.

Proof. Observe that to solve an instance of REDUCED CDS JOIN it suffices to identify, for each child q of t, a snapshot of a capacitated dominating set for $G[Y_q]$ while taking into account that (1) vertices in $X_t \setminus S$ must be dominated by a neighbor in Y_q for some child q of t, and (2) a vertex x in S can be used to dominate $c(x)$-many vertices in the sets Y_q. We will resolve point (1) by branching and then use an ILP formulation for point (2).

As our first step, let us define an equivalence $\equiv$ which identifies which children of t "behave the same way" as far as the potential interaction between their dominating sets and X_t. Formally, given two children p, q of t such that $\mathcal{D}(p) = (a_p, \beta_p)$ and $\mathcal{D}(q) = (a_q, \beta_q)$, $p \equiv q$ if and only if there exists a bijective renaming function $\iota : \partial(Y_p) \to \partial(Y_q)$ such that

- for each $v \in \partial(Y_p)$, $N(v) \setminus Y_p = N(\iota(v)) \setminus Y_q$ (i.e., neighborhoods in X_t are preserved), and
- for each snapshot σ_p of p, $\beta_p(\sigma_p) = \beta_q(\sigma_q)$, where σ_q is obtained by applying ι componentwise on σ_p (i.e., offsets of snapshots are preserved).

Since the total number of possible snapshots (up to renaming) for thin nodes is upper-bounded by a constant, the number of functions β_q for each child $q \in B_t$ is also upper-bounded by a constant. Hence, there are at most $\mathcal{O}(k^2)$-many equivalence classes of $\equiv$; let us denote these $F_1, \ldots, F_y$, where the number of equivalence classes y is in $\mathcal{O}(k^2)$.

We will now perform exhaustive branching to identify, for each $x \in X_t \setminus S$ (i.e., a vertex in X_t which must be dominated by a child of t), an equivalence class of $\equiv$ containing a node p such that a vertex in $\partial(Y_p)$ will dominate x, along with the used snapshot of p. Since $|X_t| \leq k$ and both the number of equivalence classes as well as the number of snapshots are bounded as above, this amounts to a branching factor of at most $\mathcal{O}(k^{2k}) = 2^{\mathcal{O}(k \cdot \log k)}$. In each branch, we simply check that the selected snapshot of p can indeed dominate x (by checking the value of the *offset* in the selected snapshot σ_p); if not, then we discard the current branch, and otherwise we remove p and reduce d by $a_p + \beta_p(\sigma_p)$, where $\mathcal{D}(p) = (a_p, \beta_p)$. During the branching, we take into account the fact that such p can be used to dominate up to two different vertices in $X \setminus S$—in particular, from the second vertex in $X \setminus S$ onward, we allow for a previously deleted child of t to be picked once again.

The above branching ensures that all vertices in X_t are dominated at this point, and it suffices to define the constraints for an ILP that will identify the number of times a snapshot should be used in each equivalence class of $\equiv$ in order to obtain a capacitated dominating set of size at most d. Let $\Omega = \sum_{p \in B_t; \mathcal{D}(p)=(a_p, \beta_p)} a_p$. For the variables of the ILP, we proceed as follows:

First, for each equivalence class F_i containing $\#_i$-many children of t, let $\sigma^1, \ldots, \sigma^z$

be the snapshots that occur on these children (modulo the bijective renaming function as defined above). For each $i \in [y]$ and $j \in [z]$, the ILP will contain an integer variable s_i^j which captures the number of nodes in F_i for which the sought-after capacitated dominating set D will correspond to the jth snapshot; i.e., for each node $q \in F_i$, the set $D \cap Y_q$ will have the same intersection on $\partial(Y_q)$ as specified by *active*, and the capacities on $\partial(Y_q)$ will correspond to those given in *offset*.

Second, let $S = \{x_1, \ldots, x_\iota\}$ for $\iota \le k$. For each each variable s_i^j which corresponds to a snapshot of an equivalence class whose borders all contain a single vertex, say w, that is adjacent to two distinct vertices $x_a, x_b \in S$ and which must be dominated from X_t (i.e., $w \notin active$), we create variables $x_{a,i}^j$ and $x_{b,i}^j$ which will capture how many of the vertices in the border of the nodes in F_i will be dominated by x_a and by x_b, respectively. These are the only variables that appear in the ILP, and hence (unlike in the previous two problems) it only has constantly-many variables. Let χ_a be a shorthand for $\sum_{x_{a,i}^j \text{ exists and is defined as above}} x_{a,i}^j$.

Before introducing the constraints, for each $x_\ell \in S$ let Q_ℓ be the set of all variables s_i^j corresponding to the snapshots of the equivalence classes which contain precisely *one* vertex on the boundary that (1) lies outside of *active*, that (2) has precisely one neighbor in X_t, and that is x_ℓ; in other words, Q_ℓ contains those variables which correspond to children in B_t that will require x_ℓ to dominate one vertex. Similarly, let U_ℓ be the set of all variables s_i^j corresponding to the snapshots of the equivalence classes which contain precisely *two* vertices on the boundary that (1) both lie outside of *active*, that (2) both have precisely one neighbor in X_t, and that is x_ℓ; in other words, U_ℓ contains those variables which correspond to children in B_t that will require x_ℓ to dominate two vertices.

We now introduce the following constraints in the ILP:

1. Each variable is nonnegative.
2. For each equivalence class F_i, we ensure that variables $s_i^1, \ldots, s_i^z$ represent a correct partitioning of the nodes in F_i, in particular by requiring $\#_i = \sum_{j \in [z]} s_i^j$.
3. For each vertex $x_\ell \in X_t$, we ensure that x_ℓ has sufficient capacity to dominate all the vertices in the borders of children in B_t given the domination requirements of the individual snapshots and the number of times each snapshot occurs in each F_i, in particular by requiring $c(x) \ge \left(\sum_{r \in Q_\ell} r\right) + \left(\sum_{u \in U_\ell} 2u\right) + \chi_\ell$.
4. Finally, for each variable s_i^j which corresponds to a snapshot of an equivalence class whose borders all contain a single vertex, say w, that is adjacent to two distinct vertices $x_a, x_b \in S$, we ensure that the nodes which will reject snapshot σ_j in equivalence class F_i are all dominated by requiring $s_i^j = x_{a,i}^j + x_{b,i}^j$.

Correctness follows via arguments that are analogous to those in the proof of Lemma 4.7. Since the ILP formulation has constant size, it can be solved by Theorem 2.2 in time at most $|B_t|$ and the lemma follows. □

We now have all the ingredients necessary to establish that tcw-CDS is FPT.

THEOREM 4.23. *tcw*-CDS *can be solved in time* $2^{\mathcal{O}(k^2)} \cdot n^2$.

Proof. The proof is analogous to the proofs of Theorems 4.9 and 4.16. □

4.4. Proof of Proposition 2.6. As the final result in this section, we provide a proof for the relationship between treewidth and tree-cut width claimed in Subsection 2.5.

To provide some intuition for the proof, let us first recall that for each thin node t of a nice tree-decomposition with parent $p(t)$, t may belong to $B_{p(t)}$ or $A_{p(t)}$ depending on whether $N(Y_t) \subseteq X_{p(t)}$ holds or not. Let us say that a thin node t is *B-thin* (resp., *A-thin*) if $t \in B_{p(t)}$ (resp., $t \in A_{p(t)}$).

In the proof, we show that a nice tree-cut decomposition $(T, \mathcal{X})$ of width at most k can be converted to a tree-decomposition of width of $2k^2 + 3k$. One intuitive way to obtain a tree-decomposition from $(T, \mathcal{X})$ would be to, for every edge xy of G, add $\{x, y\}$ to all nodes of T lying on the unique path connecting t_x and t_y, namely the nodes whose bags contain x and y. Put equivalently, for each vertex x of G, one can find a minimal subtree $T(x)$ of T whose bags collectively cover $N[x]$ and add x to all nodes of $T(x)$. It is easy to check that the resulting collection of bags together with T form a tree-decomposition. However, this construction can create a bag with an arbitrarily large number of vertices. Specifically, any node t of a nice tree-cut decomposition $(T, \mathcal{X})$ may have unboundedly many B-thin children, and the aforementioned "intuitive" construction would add to t all vertices in $\bigcup_{t' \in B_t} Y_t$ with neighbors in X_t.

To remedy this issue, we use a *truncated* version of $T(x)$ and add x to the nodes of this truncated version of $T(x)$ only. In this way, we intend to avoid adding x to t when $x \in Y_{t'}$ for some B-thin child t' of t even if t belongs to $T(x)$. It turns out that this simple tweak of the previous attempt is sufficient to obtain the desired bound on the treewidth. We formalize the idea in the proof below.

Proof of Proposition 2.6. Let G be an arbitrary graph of tree-cut width k. By Theorem 3.4, we may assume that G has a nice tree-cut decomposition $(T, \mathcal{X})$ with at most $2|V(G)|$ nodes.

From $(T, \mathcal{X})$, we construct a pair $(T, \mathcal{Z})$, where $\mathcal{Z}$ consists of vertex subsets $Z_t \subseteq V(G)$ over all nodes t of T. For every vertex $v \in V(G)$, let $t(v)$ be the node of T whose bag $X_{t(v)}$ contains v and let $T(v)$ be the minimal subtree of T satisfying $N[v] \subseteq \bigcup_{t \in T(v)} X_t$. Clearly, $t(v)$ is contained in $T(v)$. If there is a B-thin node on the unique path from the root of $T(v)$ to $t(v)$, let $\mathsf{top}^*(v)$ be the lowermost B-thin node among such B-thin nodes; if no B-thin node exists between $t(v)$ and $\mathsf{top}(v)$, we set $\mathsf{top}^*(v)$ to be the root of $T(v)$. Now, for every $v \in V(G)$, let T'_v be the subtree of $T(v)$ rooted at $\mathsf{top}^*(v)$. Finally, for each node t of T'_v we add v to Z_t. This completes the construction of $(T, \mathcal{Z})$.

We want to show that $(T, \mathcal{Z})$ is a tree-decomposition of G. First, the construction guarantees that for every $v \in V(G)$ the set of bags containing v is the tree T'_v and thus is connected in T. Next, we shall argue that for every edge uv of G there exists a bag in $\mathcal{Z}$ containing both u and v. For this, it suffices to verify that T'_u and T'_v have at least one node in common; let us assume for a contradiction that this is not the case. If the root of T'_v is an ancestor of T'_u in T, then T'_v must have included $t(u)$ due to the edge uv and thus the root of T'_u. Therefore, if T'_u and T'_v are disjoint, the roots of T'_u and T'_v cannot be in an ancestor-descendant relation. Note that both $T(u)$ and $T(v)$ contain $\{t(u), t(v)\}$, and in particular the least common ancestor of T'_u and T'_v. The fact that $T'_u \neq T(u)$ and $T'_v \neq T(v)$ entails that the roots, i.e., $\mathsf{top}^*(u)$ and $\mathsf{top}^*(v)$ of T'_u and T'_v, respectively, are B-thin nodes. Recall that $t(u)$ and $t(v)$ are, respectively, nodes of T'_u and T'_v, and hence $u \in Y_{\mathsf{top}^*(u)}$ and $v \in Y_{\mathsf{top}^*(v)}$. However, the niceness property of $(T, \mathcal{X})$ implies that $N(Y_{\mathsf{top}^*(u)})$ are included in the bag of the parent of $\mathsf{top}^*(u)$, a contradiction. Therefore, it follows that $(T, \mathcal{Z})$ is a tree-decomposition of G.

To verify the width of $(T, \mathcal{Z})$, consider an arbitrary node t and let us bound the number of vertices v such that $t \in T'_v$, which equals $|Z_t|$. Observe that $t \in T'_v$ holds

only if t is a descendant of $\mathsf{top}^*(v)$ (possibly the same node). This condition does not hold for any vertex in $Y_{t'}$ for $t' \in B_t$, implying that Z_t is disjoint from $\bigcup_{t' \in B_t} Y_t$. Consider $v \in Z_t \setminus X_t$. If $v \notin Y_t$, observe that $t(v)$ is a strict ancestor of t and the reason that v is placed in Z_t is because T'_v contains t, and especially v has a neighbor in Y_t. As such an edge connecting v and its neighbor in Y_t is counted in $\mathsf{adh}(t)$, there are at most $|\mathsf{adh}(t)|$ such vertices v, namely satisfying $v \in Z_t \setminus X_t \setminus Y_t$. If $v \in Y_t$, note that $v \in Y_{t'}$ for some $t' \in A_t$ and $v \in Z_t$ implicates that T'_v contains t. This means that v has a neighbor outside $Y_{t'}$ since otherwise the root of T'_v would have been a descendant of t'. Such an edge connecting v and its neighbor outside $Y'_{t'}$ is counted in $\mathsf{adh}(t')$. With Lemma 4.1, there are at most $2k+1$ nodes in A_t, and hence there are at most $(2k+1) \cdot k$ vertices v such that $v \in (Z_t \setminus X_t) \cap Y_t$. This yields the claimed bound $|Z_t| \le |X_t| + |\mathsf{adh}(t)| + \sum_{t' \in A_t} |\mathsf{adh}(t')| \le 2k^2 + 3k$. $\qquad\square$

5. Lower bounds. We show that List Coloring [10] and Precoloring Extension [2] are W[1]-hard parameterized by tree-cut width, strengthening the known W[1]-hardness results with respect to treewidth [11]. Both problems have been studied extensively in the classical [33, 22] as well as parameterized [11, 17] settings. A *coloring c* is a mapping from the vertex set of a graph to a set of colors; a coloring is *proper* if for every pair of adjacent vertices a, b it holds that $c(a) \ne c(b)$. Problem definitions follow.

---

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 to tcw-List Coloring; the only difference is that in Precoloring Extension lists are restricted to either contain a single color or all possible colors. Before stating the new hardness result, we mention (and prove) the following observation.

Observation 5. List Coloring *and* Precoloring Extension *parameterized by* **degtw** *are FPT*.

Proof. Assume that there exists a solution to an instance of List Coloring, i.e., there exists a mapping *col* from each vertex to a color in $L(v)$; w.l.o.g., we assume that colors are represented by numbers in $\mathbb{N}_0$. From *col*, we can construct a new mapping *col'* such that *col'* maps each vertex v to one of the first **degtw** $+ 1$ elements in $L(v)$ without creating conflicts (this may be done greedily from *col* in an iterative fashion). The mapping *col'* witnesses that there must exist a solution to the instance which only chooses one of the first **degtw** $+ 1$ colors in each list. This allows the construction of a standard dynamic algorithm on the tree-decomposition of the input graph. $\qquad\square$

Theorem 5.1. *tcw-List Coloring and tcw-Precoloring Extension are W[1]-hard.*

Proof. We use the reduction from the W[1]-hard problem Multi-Colored Clique (MCC) to List Coloring described in [11, Theorem 2].

> MULTI-COLORED CLIQUE (MCC)
> *Instance*: A k-partite graph G with k parts $V_1, \ldots, V_k$, each consisting of n vertices.
> *Parameter*: k.
> *Task*: Decide whether there a k-clique in G.

We outline the reduction below. Given an instance $G = (V_1 \cup \cdots \cup V_k, E)$ of MCC, we construct an instance I of LIST COLORING with vertex sets X, Y, whereas for each V_i there is a single $i \in X$ such that $L(i) = V_i$. Then, for each nonedge $\{a \in V_i, b \in V_j\} \notin E$, where $i \neq j$, we construct a vertex $y \in Y$ such that y is adjacent to i and j and $L(y) = \{a, b\}$. Then the choice of color for each $i \in X$ corresponds to a choice of a single vertex from V_i, and the set Y contains "constraints" which prevent the choice of two nonadjacent vertices.

To prove that LIST COLORING is W[1]-hard, it now suffices to prove that the constructed instance I has tree-cut width at most k. To this end, consider the following tree-cut decomposition $(T, \mathcal{X})$ of I: T is a star with center $s \in V(T)$ and $X_s = X$, and for each $y \in Y$ there is a leaf $z \in V(T)$ such that $X_z = \{y\}$. Assume that T is rooted at s. Then $\mathsf{adh}(s) = 0$ and $\mathsf{adh}(z) = 2$ for each leaf z. Hence, $\mathsf{tor}(s) = k$ and $\mathsf{tor}(z) = 3$, from which it follows that I has tree-cut width k.

To show that PRECOLORING EXTENSION is also W[1]-hard, we recall the simple reduction from LIST COLORING to PRECOLORING EXTENSION also described in [11]. From an instance of LIST COLORING, it is possible to construct an instance of PRECOLORING EXTENSION by "modeling" the lists through the addition of precolored pendant vertices (vertices of degree one). Let I' be an instance of PRECOLORING EXTENSION constructed in this manner from an instance I of LIST COLORING described above; let Q contain all the new pendant vertices, i.e., $Q = V(I') \setminus V(i)$.

We show that I' also has a tree-cut decomposition $(T', \mathcal{X}')$ of width k. For vertex sets $X, Y \in I'$, we use $(T, \mathcal{X})$. Then, for each $q \in Q$ adjacent to a vertex $v \in X \cup Y$ such that $v \in X_{t_v}$, we construct a pendant $t_q \in V(T')$ adjacent to t_v such that $X'_{t_q} = \{q\}$. The adhesion and torso-size of vertices in $X \cup Y$ remains the same as in $(T, \mathcal{X})$, while for each $q \in Q$ it holds that $\mathsf{adh}(t_q) = 1$ and $\mathsf{tor}(t_q) = 1$. The theorem follows. □

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** [39]. Formal definitions follow.

An instance I of the CSP is a triple $(X, D, \mathcal{C})$, where X is a finite set of *variables*, D is a finite set of *domain values*, and $\mathcal{C}$ is a finite set of *constraints*. Each constraint in $\mathcal{C}$ is a pair (S, R), where S, the *constraint scope*, is a nonempty sequence of distinct variables of X, and R, the *constraint relation*, is a relation over D (given as a set of tuples) whose arity matches the length of S. A CSP instance $(X, D, \mathcal{C})$ is *Boolean* if $D = \{0, 1\}$. An *assignment* is a mapping from the set X of variables to the domain D. An assignment τ *satisfies* a constraint $C = ((x_1, \ldots, x_n), R)$ if $(\tau(x_1), \ldots, \tau(x_n)) \in R$, and τ satisfies the CSP instance if it satisfies all its constraints. An instance I is *satisfiable* if it is satisfied by some assignment.

The *incidence graph* G_I of CSP instance $I = (V, D, \mathcal{C})$ is the bipartite graph whose vertex set is formed by $V \cup \mathcal{C}$ and where a constraint $C = (S, R) \in \mathcal{C}$ is incident exactly to all the variables in S.

> 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.

tcw-BOOLEAN-CSP denotes tcw-CSP restricted to Boolean CSP instances. We note that BOOLEAN-CSP parameterized by **degtw** is FPT [39].

THEOREM 5.2. *tcw*-BOOLEAN CSP *is* W[1]-*hard*.

Proof. We give a reduction from MCC. Let (G, k) be an instance of MCC with $V(G) = \bigcup_{i=1}^{n} V_i$, where $V_i = \{v_{i,1}, \ldots, v_{i,n}\}$ for $1 \leq i \leq k$. We first construct a CSP instance $I = (X, D, \mathcal{C})$ and will show later how to make it Boolean. We let $X = \{x_{i,j} \mid 1 \leq i, j \leq k\}$ and $D = \{1, \ldots, n\}$. For $1 \leq i \leq k$, the set $\mathcal{C}$ contains the constraint $C_i^{=}((x_{i,1}, \ldots, x_{i,k}), \{(0, \ldots, 0), (1, \ldots, 1), \ldots, (n, \ldots, n)\})$, which enforces that the values of all the variables $x_{i,1}, \ldots, x_{i,k}$ are the same. Furthermore, for each $1 \leq i < j \leq k$ the set $\mathcal{C}$ contains the constraint $C_{i,j}^{E} = ((x_{i,j}, x_{j,i}), \{(a, b) \in [n] \times [n] \mid v_{i,a}v_{j,b} \in E(G)\})$ which encodes the incidence relation between V_i and V_j. It follows by construction that I is satisfiable if and only if G contains a k-clique.

Next, we obtain from I a Boolean CSP instance $I' = (X', \{0, 1\}, \mathcal{C}')$ by replacing each variable $x_{i,j}$ with n Boolean variables $x_{i,j}^{(1)}, \ldots, x_{i,j}^{(n)}$. Intuitively, assigning $x_{i,j}$ the value a corresponds to assigning $x_{i,j}^{(a)}$ the value 1 and all other $x_{i,j}^{(b)}$ for $b \neq a$ the value 0 (instead of this unary encoding we could have used a more succinct binary encoding, but the unary encoding is simpler and suffices for our purposes). Consequently, each constraint $C_i^{=}$ of I (of arity n) gives rise to a constraint $C_i^{='}$ of I (or arity n^2), and each constraint $C_{i,j}^{E}$ of I (or arity 2) gives rise to a constraint $C_{i,j}^{E'}$ of I' (of arity $2n$). By construction, I is satisfiable if and only if I' is satisfiable.

In order to complete the reduction and the proof of the theorem, it remains to construct a tree-cut decomposition of the incidence graph $G_{I'}$ of I' of a width that is bounded by a function of k. We observe that I' has exactly $k' = k + \binom{k}{2}$ many constraints and each variable appears in the scopes of exactly two constraints. Hence, one side of the bipartite graph $G_{I'}$ consists of k' many vertices and the other side only of vertices of degree 2. This already implies that the tree-cut width of $G_{I'}$ is at most k', as we can take the following tree-cut decomposition $(T, \mathcal{X})$ of $G_{I'}$. T is a star with center $s \in V(T)$ where $X_s = \mathcal{C}$, and for each variable $x_{i,j}^{(a)}$ there is leaf $t_{i,j}^{(a)} \in V(T)$ with $X_{t_{i,j}^{(a)}} = \{x_{i,j}^{(a)}\}$. $\square$

6. Concluding notes. We have provided the first algorithmic applications of the new graph parameter tree-cut width, considering various hard combinatorial problems. In some cases, we could establish fixed-parameter tractability, and in other 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. In fact, following their introduction these decompositions have already been used to obtain algorithms for problems such as BOUNDED DEGREE VERTEX DELETION [18], STABLE ROOMMATES WITH TIES AND INCOMPLETE LISTS [4], and EDGE DISJOINT PATHS [20]. Surprisingly, while the third problem is XP when parameterized by tree-cut width, it remains W[1]-hard under this parameterization [20]—a stark contrast to the behavior of VERTEX DISJOINT PATHS parameterized by treewidth.

We remark that the quadratic dependency on n in our algorithms can almost certainly be reduced to linear by using a carefully designed data structure or avoiding the use of Turing reductions. In particular, in all three cases the quadratic dependency is caused by the fact that the Turing reductions used in the proofs of Lemmas 4.6, 4.13, and 4.21 produce subinstances of size $\mathcal{O}(n)$; other than that, these reductions run in time that depends only on k.

While we do not yet have an exact fixed-parameter algorithm for computing tree-cut width, the 2-approximation algorithm of Kim et al. [28] is sufficient to establish fixed-parameter tractability of various problems of interest. Moreover, there is also a SAT encoding which can compute the tree-cut width exactly for graphs with dozens of vertices [19].

Acknowledgment. The authors also gratefully acknowledge the detailed feedback provided by the reviewers.

REFERENCES

[1] T. C. BIEDL, T. M. CHAN, Y. GANJALI, M. T. HAJIAGHAYI, AND D. R. WOOD, *Balanced vertex-orderings of graphs*, Discrete Appl. Math., 148 (2005), pp. 27–48.

[2] M. BIRÓ, M. HUJTER, AND Z. TUZA, *Precoloring extension. I. Interval graphs*, Discrete Math., 100 (1992), pp. 267–279.

[3] H. L. BODLAENDER, D. LOKSHTANOV, AND E. PENNINKX, *Planar capacitated dominating set is W[1]-hard*, in Parameterized and Exact Computation, Lecture Notes in Comput. Sci. 5917, Springer, Berlin, 2009, pp. 50–60.

[4] R. BREDERECK, K. HEEGER, D. KNOP, AND R. NIEDERMEIER, *Parameterized complexity of stable roommates with ties and incomplete lists through the lens of graph parameters*, in Proceedings of the 30th International Symposium on Algorithms and Computation (ISAAC 2019), Shanghai University of Finance and Economics, Shanghai, China, LIPIcs. Leibniz Int. Proc. Inform. 149, P. Lu and G. Zhang, eds., Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Wadern, Germany, 2019, 44.

[5] J. CHUZHOY AND J. NAOR, *Covering problems with hard capacities*, in Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science (FOCS 2002), Vancouver, BC, Canada, 2002, pp. 481–489.

[6] M. CYGAN, F. V. FOMIN, L. KOWALIK, D. LOKSHTANOV, D. MARX, M. PILIPCZUK, M. PILIPCZUK, AND S. SAURABH, *Parameterized Algorithms*, Springer, Cham, 2015.

[7] R. DIESTEL, *Graph Theory*, 2nd ed., Grad. Texts Math. 173, Springer-Verlag, New York, 2000.

[8] M. DOM, D. LOKSHTANOV, S. SAURABH, AND Y. VILLANGER, *Capacitated domination and covering: A parameterized perspective*, in Parameterized and Exact Computation, Third International Workshop (IWPEC 2008), Victoria, BC, Canada, 2008, Lecture Notes in Comput. Sci. 5018, M. Grohe and R. Niedermeier, eds., Springer, Berlin, 2008, pp. 78–90.

[9] R. G. DOWNEY AND M. R. FELLOWS, *Fundamentals of Parameterized Complexity*, Texts Comput. Sci., Springer, London, 2013.

[10] P. ERDŐS, A. L. RUBIN, AND H. TAYLOR, *Choosability in graphs*, in Proceedings of the West Coast Conference on Combinatorics, Graph Theory and Computing (Humboldt State University, Arcata, CA, 1979), Congress. Numer. XXVI, Utilitas Mathematica, Winnepeg, MB, Canada, 1979, pp. 125–157.

[11] 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*, Inform. and Comput., 209 (2011), pp. 143–153.

[12] M. R. FELLOWS, D. LOKSHTANOV, N. MISRA, F. A. ROSAMOND, AND S. SAURABH, *Graph layout problems parameterized by vertex cover*, in Algorithms and Computation, 19th International Symposium (ISAAC 2008), Gold Coast, Australia, Lecture Notes in Comput. Sci. 5369, S. Hong, H. Nagamochi, and T. Fukunaga, eds., Springer, Berlin, 2008, pp. 294–305.

[13] J. FIALA, P. A. GOLOVACH, AND J. KRATOCHVÍL, *Parameterized complexity of coloring problems: Treewidth versus vertex cover*, Theoret. Comput. Sci., 412 (2011), pp. 2513–2523.

[14] J. FLUM AND M. GROHE, *Parameterized Complexity Theory*, Texts Theoret. Comput. Sci. EATCS Ser., Springer-Verlag, Berlin, 2006.

[15] F. V. FOMIN, P. A. GOLOVACH, D. LOKSHTANOV, AND S. SAURABH, *Clique-width: On the price*

of generality, in Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2009), New York, NY, 2009, C. Mathieu, ed., SIAM, Philadelphia, 2009, pp. 825–834, https://doi.org/10.1137/1.9781611973068.90.

[16] A. FRANK AND É. TARDOS, *An application of simultaneous diophantine approximation in combinatorial optimization*, Combinatorica, 7 (1987), pp. 49–65.

[17] R. GANIAN, *Twin-cover: Beyond vertex cover in parameterized algorithmics*, in Parameterized and Exact Computation, 6th International Symposium (IPEC 2011), Saarbrücken, Germany, Lecture Notes in Comput. Sci. 7112, Springer, Heidelberg, 2012, pp. 259–271.

[18] R. GANIAN, F. KLUTE, AND S. ORDYNIAK, *On structural parameterizations of the bounded-degree vertex deletion problem*, Algorithmica, 83 (2021), pp. 297–336.

[19] R. GANIAN, N. LODHA, S. ORDYNIAK, AND S. SZEIDER, *SAT-encodings for treecut width and treedepth*, in Proceedings of the Twenty-First Meeting on Algorithm Engineering and Experiments (ALENEX 2019), San Diego, CA, 2019, S. G. Kobourov and H. Meyerhenke, eds., SIAM, Philadelphia, 2019, pp. 117–129, https://doi.org/10.1137/1.9781611975499.10.

[20] R. GANIAN AND S. ORDYNIAK, *The power of cut-based parameters for computing edge-disjoint paths*, Algorithmica, 83 (2021), pp. 726–752.

[21] S. GASPERS, M. MESSINGER, R. J. NOWAKOWSKI, AND P. PRALAT, *Clean the graph before you draw it!*, Inform. Process. Lett., 109 (2009), pp. 463–467.

[22] S. GRAVIER, D. KOBLER, AND W. KUBIAK, *Complexity of list coloring problems with a fixed total number of colors*, Discrete Appl. Math., 117 (2002), pp. 65–79.

[23] 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, ACM, New York, 2011, pp. 479–488.

[24] S. GUHA, R. HASSIN, S. KHULLER, AND E. OR, *Capacitated vertex covering*, J. Algorithms, 48 (2003), pp. 257–270.

[25] J. GUO, R. NIEDERMEIER, AND S. WERNICKE, *Parameterized complexity of vertex cover variants*, Theory Comput. Syst., 41 (2007), pp. 501–520.

[26] R. IMPAGLIAZZO, R. PATURI, AND F. ZANE, *Which problems have strongly exponential complexity?*, J. Comput. System Sci., 63 (2001), pp. 512–530.

[27] R. KANNAN, *Minkowski's convex body theorem and integer programming*, Math. Oper. Res., 12 (1987), pp. 415–440.

[28] E. J. KIM, S. OUM, C. PAUL, I. SAU, AND D. M. THILIKOS, *An FPT 2-approximation for tree-cut decomposition*, Algorithmica, 80 (2018), pp. 116–135, https://doi.org/10.1007/s00453-016-0245-5.

[29] T. KLOKS, *Treewidth: Computations and Approximations*, Springer-Verlag, Berlin, 1994.

[30] F. KUHN AND T. MOSCIBRODA, *Distributed approximation of capacitated dominating sets*, Theory Comput. Syst., 47 (2010), pp. 811–836.

[31] H. LENSTRA, *Integer programming with a fixed number of variables*, Math. Oper. Res., 8 (1983), pp. 538–548.

[32] D. LOKSHTANOV, N. MISRA, AND S. SAURABH, *Imbalance is fixed parameter tractable*, Inform. Process. Lett., 113 (2013), pp. 714–718.

[33] D. MARX, *NP-completeness of list coloring and precoloring extension on the edges of planar graphs*, J. Graph Theory, 49 (2005), pp. 313–324.

[34] D. MARX AND P. WOLLAN, *Immersions in highly edge connected graphs*, SIAM J. Discrete Math., 28 (2014), pp. 503–520, https://doi.org/10.1137/130924056.

[35] C. S. J. A. NASH-WILLIAMS, *On well-quasi-ordering trees*, in Theory of Graphs and Its Applications, Academia Publishing House of The Czech Academy of Sciences, Prague, 1964, pp. 83–84.

[36] C. S. J. A. NASH-WILLIAMS, *On well-quasi-ordering infinite trees*, Proc. Cambridge Philos. Soc., 61 (1965), pp. 697–720.

[37] J. NEŠETŘIL AND P. O. DE MENDEZ, *Tree-depth, subgraph coloring and homomorphism bounds*, European J. Combin., 27 (2006), pp. 1024–1041.

[38] N. ROBERTSON AND P. D. SEYMOUR, *Graph minors. XIII. The disjoint paths problem*, J. Combin. Theory Ser. B, 63 (1995), pp. 65–110.

[39] M. SAMER AND S. SZEIDER, *Constraint satisfaction with bounded treewidth revisited*, J. Comput. System Sci., 76 (2010), pp. 103–114.

[40] A. TAKAHASHI, S. UENO, AND Y. KAJITANI, *Minimal acyclic forbidden minors for the family of graphs with bounded path-width*, Discrete Math., 127 (1994), pp. 293–304, https://doi.org/10.1016/0012-365X(94)90092-2.

[41] P. WOLLAN, *The structure of graphs not admitting a fixed immersion*, J. Combin. Theory Ser. B, 110 (2015), pp. 47–66.