Edge-Cut Width: An Algorithmically Driven Analogue of Treewidth Based on Edge Cuts

Cornelius Brand, Esra Ceylan, Robert Ganian, Christian Hatschka, and Viktoriia Korchemna

Algorithms and Complexity Group, TU Wien, Vienna, Austria

Abstract. Decompositional parameters such as treewidth are commonly used to obtain fixed-parameter algorithms for NP-hard graph problems. For problems that are W[1]-hard parameterized by treewidth, a natural alternative would be to use a suitable analogue of treewidth that is based on edge cuts instead of vertex separators. While tree-cut width has been coined as such an analogue of treewidth for edge cuts, its algorithmic applications have often led to disappointing results: out of twelve problems where one would hope for fixed-parameter tractability parameterized by an edge-cut based analogue to treewidth, eight were shown to be W[1]-hard parameterized by tree-cut width.

As our main contribution, we develop an edge-cut based analogue to treewidth called edge-cut width. Edge-cut width is, intuitively, based on measuring the density of cycles passing through a spanning tree of the graph. Its benefits include not only a comparatively simple definition, but mainly that it has interesting algorithmic properties: it can be computed by a fixed-parameter algorithm, and it yields fixed-parameter algorithms for all the aforementioned problems where tree-cut width failed to do so.

1 Introduction

While the majority of computational problems on graphs are intractable, in most cases it is possible to exploit the structure of the input graphs to circumvent this intractability. This basic fact has led to the extensive study of a broad hierarchy of decompositional graph parameters (see, e.g., Figure 1 in [3]), where for individual problems of interest the aim is to pinpoint which parameters can be used to develop fixed-parameter algorithms for the problem. Treewidth [31] is by far the most prominent parameter in the hierarchy, and it is known that many problems of interest are fixed-parameter tractable when parameterized by treewidth; some of these problem can even be solved efficiently on more general parameters such as rank-width [13, 30] or other decompositional parameters above treewidth in the hierarchy [4]. However, in this article we will primarily be interested in problems that lie on the other side of this spectrum: those which remain intractable when parameterized by treewidth.

Aside from non-decompositional parameters¹ such as the vertex cover number [10, 12] or feedback edge number [1, 18, 21], the most commonly applied parameters for

Cornelius Brand, Robert Ganian and Viktoriia Korchemna gratefully acknowledge support from the Austria Science Foundation (FWF, Project Y1329).

¹ We view a parameter as decompositional if it is tied to a well-defined graph decomposition; all decompositional parameters are closed under the disjoint union operation of graphs.

problems which are not fixed-parameter tractable with respect to treewidth are tied to the existence of small vertex separators. One example of such a parameter is treedepth [29], which has by now found numerous applications in diverse areas of computer science [17, 23, 28]. An alternative approach is to use a decompositional parameter that is inherently tied to edge-cuts—in particular, tree-cut width [27, 33].

Tree-cut width was discovered by Wollan, who described it as a variation of tree decompositions based on edge cuts instead of vertex separators [33]. But while it is true that "tree-cut decompositions share many of the natural properties of tree decompositions" [27], from the perspective of algorithmic design tree-cut width seems to behave differently than an edge-cut based alternative to treewidth. To illustrate this, we note that tree-cut width is a parameter that lies between treewidth and treewidth plus maximum degree (which may be seen as a "heavy-handed" parameterization that enforces small edge cuts) in the parameter hierarchy [14, 24]. There are numerous problems which are W[1]-hard (and sometimes even NP-hard) w.r.t. treewidth but fixed-parameter tractable w.r.t. the latter parameterization, and the aim would be to have an edge-cut based parameter that can lift this fixed-parameter tractability towards graphs of unbounded degree.

Unfortunately, out of twelve problems with these properties where a tree-cut width parameterization has been pursued so far, only four are fixed-parameter tractable [14, 15] while eight turn out to be W[1]-hard [5, 14, 16, 18, 22]. The most appalling example of the latter case is the well-established EDGE DISJOINT PATHS (EDP) problem: VERTEX DISJOINT PATHS is a classical example of a problem that is FPT parameterized by treewidth, and one should by all means expect a similar outcome for EDP parameterized by the analogue of treewidth based on edge cuts [18, 19]. But if EDP is W[1]-hard parameterized by tree-cut width, what is the algorithmic analogue of treewidth for edge cuts? Here, we attempt to answer to this question through the notion of edge-cut width.

Contribution. Edge-cut width is an edge-cut based decompositional parameter which has a surprisingly streamlined definition: instead of specialized decompositions such as those employed by treewidth, clique-width or tree-cut width, the "decompositions" for edge-cut width are merely spanning trees (or, in case of disconnected graphs, maximum spanning forests). To define edge-cut width of a spanning tree T, we observe that for each edge in G - T there is a unique path in T connecting its endpoints, and the edge-cut width of T is merely the maximum number of such paths that pass through any particular vertex in T; as usual, the edge-cut width of G is then the minimum width of a spanning tree (i.e., decomposition).

After introducing edge-cut width, establishing some basic properties of the parameter and providing an in-depth comparison to tree-cut width, we show that the parameter has surprisingly useful algorithmic properties. As our first task, we focus on the problem of computing edge-cut width along with a suitable decomposition. This is crucial, since we will generally need to compute an edge-cut width decomposition before we can use the parameter to solve problems of interest. As our first algorithmic result, we leverage the connection of edge-cut width to spanning trees of the graph to obtain an explicit fixed-parameter algorithm for computing edge-cut width decompositions. This compares favorably to tree-cut width, for which only an explicit 2-approximation fixed-parameter algorithm [24] and a non-constructive fixed-parameter algorithm [20] are known. Finally, we turn to the algorithmic applications of edge-cut width. Recall that among the twelve problems where a parameterization by tree-cut width had been pursued, eight were shown to be W[1]-hard parameterized by tree-cut width: LIST COLORING [14], PRECOLORING EXTENSION [14], BOOLEAN CONSTRAINT SATISFACTION [14], EDGE DISJOINT PATHS [18], BAYESIAN NETWORK STRUCTURE LEARNING [16], POLY-TREE LEARNING [16], MINIMUM CHANGEOVER COST ARBORESCENCE [22], and MAXIMUM STABLE ROOMMATES WITH TIES AND INCOMPLETE LISTS [5]. Here, we follow up on previous work by showing that *all* of these problems are fixed-parameter tractable when parameterized by edge-cut width. We obtain our algorithms using a new dynamic programming framework for edge-cut width, which can also be adapted for other problems of interest.

Related Work. The origins of edge-cut width lie in the very recent work of Ganian and Korchemna on learning polytrees and Bayesian networks [16], who discovered an equivalent parameter when attempting to lift the fixed-parameter tractability of these problems to a less restrictive parameter than the feedback edge number². That same work also showed that computing edge-cut width can be expressed in Monadic Second Order Logic which implies fixed-parameter tractability, but obtaining an explicit fixedparameter algorithm for computing optimal decompositions was left as an open question.

As far as the authors are aware, there are only four problems for which it is known that fixed-parameter tractability can be lifted from the parameterization by "maximum degree plus treewidth" to tree-cut width. These are CAPACITATED VERTEX COVER [14], CAPACITATED DOMINATING SET [14], IMBALANCE [14] and BOUNDED DEGREE VERTEX DELETION [15]. Additionally, Gozupek et al. [22] showed that the MINIMUM CHANGEOVER COST ARBORESCENCE problem is fixed-parameter tractable when parameterized by a special, restricted version of tree-cut width where one essentially requires the so-called *torsos* to be stars.

2 Preliminaries

We use standard terminology for graph theory, see for instance [7]. 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 $N_G(x)$. For a vertex subset X, the neighborhood of X is defined as $\bigcup_{x \in X} N_G(x) \setminus X$ and denoted by $N_G(X)$; we drop the subscript if the graph is clear from the context. *Contracting* an edge $\{a, b\}$ is the operation of replacing vertices a, b by a new vertex whose neighborhood is $(N(a) \cup N(b)) \setminus \{a, b\}$. For a vertex set A (or edge set B), we use G - A(G - B) to denote the graph obtained from G by deleting all vertices in A (edges in B), and we use G[A] to denote the *subgraph induced on* A, i.e., $G - (V(G) \setminus A)$.

A *forest* is a graph without cycles, and an edge set X is a *feedback edge set* if G - X is a forest. We use [i] to denote the set $\{0, 1, \ldots, i\}$.

Given two graph parameters $\alpha, \beta : G \to \mathbb{N}$, we say that α dominates β if there exists a function p such that for each graph $G, \beta(G) \leq p(\alpha(G))$.

² The authors originally used the name "local feedback edge number".

2.1 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 non-negative integervalued function defined on Σ^* . Then L parameterized by p denotes the parameterized problem $\{(x, p(x)) | x \in L\}$ 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 in short) 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. We call algorithms running in this time *fixed-parameter algorithms*.

Parameterized complexity classes are defined with respect to fpt-reducibility. A parameterized problem P is fpt-reducible to Q if in time $f(k) \cdot |x|^{O(1)}$, 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 $k' \leq g(k)$, where f and g are computable functions depending only on k. Owing to the definition, if P fpt-reduces to Q and Q is fixed-parameter tractable then P is fixed-parameter tractable 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, $FPT \neq W[1]$ under the Exponential Time Hypothesis.

The class W[1] is the analog 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 W[1]-hard, i.e., those to which every problem in W[1] is fpt-reducible. There are many problems shown to be complete for W[1], or equivalently W[1]-complete, including the MULTI-COLORED CLIQUE (MCC) problem [8]. We refer the reader to the respective monographs [6, 8] for an in-depth introduction to parameterized complexity.

2.2 Treewidth

Treewidth [31] is a fundamental graph parameter that has found a multitude of algorithmic applications throughout computer science.

Definition 1. A tree decomposition of a graph G is a pair $(T, \{\beta_t\}_{t \in V(T)})$, where T is a tree, and each node $t \in V(T)$ is associated with a bag $\beta_t \subseteq V(G)$, satisfying the following conditions:

- 1. Every vertex of G appears in some bag of T.
- 2. Every edge of G is contained as a subset in some bag of T.
- 3. For every vertex $v \in V(G)$, the set of nodes $t \in V(T)$ such that $v \in \beta_t$ holds is connected in T.

The width of a tree decomposition is defined as $\max_t |\beta_t| - 1$, and the treewidth $\operatorname{tw}(G)$ of G is defined as the minimum width of any of its tree decompositions.

For our algorithms, it will be useful to make some additional assumptions on the tree decomposition.

Definition 2. A tree decomposition $(T, \{\beta_t\}_{t \in V(T)})$ is called nice if is satisfies the following:

- 1. T has a distinguished root $r \in V(T)$ with $\beta_r = \emptyset$.
- 2. Every node of T has at most two children.
- 3. For every node t of T with two children s, u it holds that $\beta_t = \beta_s = \beta_u$. These nodes are called join-nodes.
- 4. For every node t of T with exactly one child s, there is a vertex $v \in V(G)$ such that either $\beta_t - \beta_u = \{v\}$, in which case we call t an introduce-node, or $\beta_u - \beta_t = \{v\}$, in which case we call t a forget-node. We call v the vertex introduced (resp. forgotten) at t.
- 5. Every node that has no children is called a leaf-node of T and $\beta_t = \emptyset$ must hold.

It is known that every tree decomposition can be converted into a nice one of the same width in linear time.

2.3 Tree-cut Width

The notion of tree-cut decompositions was introduced by Wollan [33], see also [27]. 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 3. A tree-cut decomposition of G is a pair (T, \mathcal{X}) which consists of a rooted 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 node t of T other than the root r, let e(t) = ut be the unique edge incident to t on the path to r. Let T_u and T_t be the two connected components in T - e(t) which contain u and t, respectively. Note that $(\bigcup_{q \in T_u} X_q, \bigcup_{q \in T_t} X_q)$ is a near-partition of V(G), and we use E_t to denote the set of edges with one endpoint in each part. We define the *adhesion* of t (adh(t)) as $|E_t|$; we explicitly set adh(r) = 0 and $E(r) = \emptyset$.

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_ℓ be the connected components of T - t. For each $i = 1, \ldots, \ell$, the vertex set $Z_i \subseteq V(G)$ 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 (this is also called *shrinking* in the literature). Here, the operation of consolidating a vertex set Zinto 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* (also called *dissolving* in the literature) 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 connected graph G and $X \subseteq V(G)$, let the 3-center of (G, X) be the unique graph obtained from G by exhaustively suppressing vertices in $V(G) \setminus X$ of degree at most two. Finally, for a node t of T, we denote by \tilde{H}_t the 3-center of (H_t, X_t) , where H_t is the torso of (T, \mathcal{X}) at t. Let the torso-size tor(t) denote $|\tilde{H}_t|$.

Definition 4. The width of a tree-cut decomposition (T, \mathcal{X}) of G is $\max_{t \in V(T)} \{ \operatorname{adh}(t), \operatorname{tor}(t) \}$. The tree-cut width of G, or $\operatorname{tcw}(G)$ in short, is the minimum width of (T, \mathcal{X}) over all tree-cut decompositions (T, \mathcal{X}) of G.

Without loss of generality, we shall assume that $X_r = \emptyset$. We conclude this subsection with some notation related to tree-cut decompositions. 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 $G[Y_t]$. A node $t \neq r$ in a rooted tree-cut decomposition is *thin* if $adh(t) \leq 2$ and *bold* otherwise.

A tree-cut decomposition (T, \mathcal{X}) 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. Every tree-cut decomposition can be transformed into a nice tree-cut decomposition of the same width in cubic time [14].

For a node t, we let $B_t = \{b \text{ is a child of } t || N(Y_b)| \leq 2 \land N(Y_b) \subseteq X_t\}$ denote the set of thin children of t whose neighborhood is a subset of X_t , and we let $A_t = \{a \text{ is a child of } t | a \notin B_t\}$ be the set of all other children of t. Then $|A_t| \leq 2k + 1$ for every node t in a nice tree-cut decomposition [14].

We refer to previous work [14, 24, 27, 33] for a more detailed comparison of tree-cut width to other parameters. Here, we mention only that tree-cut width is dominated by treewidth and dominates treewidth plus maximum degree, which we denote degtw(G).

Lemma 1 ([14, 27, 33]). For every graph G, $tw(G) \leq 2tcw(G)^2 + 3tcw(G)$ and $tcw(G) \leq 4 \operatorname{degtw}(G)^2$.

3 Edge-Cut Width

Let us begin by considering a maximal spanning forest T of a graph G, and recall that E(G) - T forms a minimum feedback edge set in G; the size of this set is commonly called the *feedback edge number* [1, 18, 21], and it does not depend on the choice of T. We will define our parameter as the maximum number of edges from the feedback edge set that form cycles containing some particular vertex $v \in V(G)$.

Formally, for a graph G and a maximal spanning forest T of G, let the *local feedback* edge set at $v \in V$ be $E_{loc}^{G,T}(v) = \{uw \in E(G) \setminus E(T) \mid \text{the unique path between } u$ and w in T contains v}; we remark that this unique path forms a so-called *fundamental* cycle with the edge uw. The edge-cut width of (G,T) (denoted ecw(G,T)) is then equal to $1 + \max_{v \in V} |E_{loc}^{G,T}(v)|$, and the edge-cut width of G is the smallest edge-cut width among all possible maximal spanning forests of G.

Notice that the definition increments the edge-cut width of T by 1. This "cosmetic" change may seem arbitrary, but it matches the situation for treewidth (where the width is the bag size minus one) and allows trees to have a width of 1. Moreover, defining edge-cut width in this way provides a more concise description of the running times for our algorithms, where the records will usually depend on a set that is one larger than $|E_{loc}^{G,T}(v)|$. We note that the predecessor to edge-cut width, called the *local feedback edge number* [16], was defined without this cosmetic change and hence is equal to edge-cut width minus one.

While it is obvious that ecw(G) is upper-bounded by (and hence dominates) the feedback edge number of G (fen(G)), we observe that graphs of constant ecw(G) can have unbounded feedback edge number—see Figure 1. We also note that Ganian and Korchemna established that edge-cut width is dominated by tree-cut width.



Fig. 1. Example of a graph G with a spanning tree T (marked in red) such that ecw(G) = ecw(G,T) = 3. The feedback edge number of G, i.e., its edge deletion distance to acyclicity, is exactly the number of black edges and can be made arbitrarily large in this fashion while preserving ecw(G) = 3.

Proposition 1 ([16]). For every graph G, $tcw(G) \le ecw(G) \le fen(G) + 1$.

Proof. Let us begin with the second inequality. Consider an arbitrary spanning tree T of G. Then for every $v \in V(G)$, $E_{loc}^T(v)$ is a subset of a feedback edge set corresponding to the spanning tree T, so $|E_{loc}^T(v)| \leq \text{fen}(G)$ and the claim follows.

To establish the first inequality, we will use the notation and definition of tree-cut width from previous work [15, Subsection 2.4]. Let T be the spanning tree of G with ecw(G,T) = ecw(G). We construct a tree-cut decomposition (T, \mathcal{X}) where each bag contains precisely one vertex, notably by setting $X_t = \{t\}$ for each $t \in V(T)$. Fix any node t in T other than root, let u be the parent of t in T. All the edges in $G \setminus ut$ with one endpoint in the rooted subtree T_t and another outside of T_t belong to $E_{loc}^T(t)$, so $adh_T(t) = |cut(t)| \le |E_{loc}^T(t)| \le ecw(G) - 1$.

Let H_t be the torso of (T, \mathcal{X}) in t, then $V(H_t) = \{t, z_1...z_l\}$ where z_i correspond to connected components of $T \setminus t$, $i \in [l]$. In $\tilde{H}(t)$, only z_i with degree at least 3 are preserved. But all such z_i are the endpoints of at least two edges in $|E_{loc}^T(t)|$, so $\operatorname{tor}(t) = |V(\tilde{H}_t)| \leq 1 + |E_{loc}^T(t)| \leq \operatorname{ecw}(G)$. Thus $\operatorname{tcw}(G) \leq \operatorname{ecw}(G)$.

As for the converse, we already have conditional evidence that edge-cut width cannot dominate tree-cut width: BAYESIAN NETWORK STRUCTURE LEARNING is W[1]-hard w.r.t. the latter, but fixed-parameter tractable w.r.t. the former [16]. We conclude our comparisons with a construction that not only establishes this relationship unconditionally, but—more surprisingly—implies that edge-cut width is incomparable to degtw.

Lemma 2. For each $m \in \mathbb{N}$, there exists a graph G_m of degree at most 3, tree-cut width at most 2, and edge-cut width at least m + 1.

Proof. We start from two regular binary trees Y and Y' of depth m, i.e., rooted binary trees where every node except leaves has precisely two children and the path from any leaf to the root contains m edges. We glue Y and Y' together by identifying each leaf of Y with a unique leaf of Y' (see the left part of Figure 2 for an illustration). It remains to show that the resulting graph, which we denote G_m , has the desired properties.



Fig. 2. Left: Graph G_4 , where the roots of Y and Y' are a_3 and a'_3 , the path π is green and B_2 is violet. **Right**: Fragment of the tree-cut decomposition (Y^*, χ) of G_4 .

Consider arbitrary spanning tree T of G_m . There exists a unique path $\pi \subseteq T$ between the roots r and r' of Y and Y'. Observe that $G_m - \pi$ is a disjoint union of m graphs $G_l, l \in [m-1]$. We add to every such G_l two edges which connect it with π and denote the resulting graph by B_l . Then every B_l contains at least one edge that contributes to the local feedback edge set of $q \in V(\pi)$, where q is a leaf in Y and Y'. Indeed, fix any $l \in [m-1]$ and denote by a_l and a'_l the vertices of B_l intersecting π in Y and Y'correspondingly. As T is a tree, T - q is a union of two trees: one containing a_l and another containing a'_l . Hence every vertex ob B_l is connected to precisely one of a_l and a'_l in T - q. In particular, there exists an edge e_l of B_l such that one endpoint of e_l is connected to a_l and another is connected to a'_l in T - q. Then e_l belongs to the local feedback edge set of q. As B_l and $B_{l'}$ don't share edges for any $l \neq l'$, this results in $|E_{\text{local}}^{G_m,T}(q)| \ge m$. Since the inequality holds for any choice of T, we may conclude that $\operatorname{ecw}(G_m) \ge m + 1$.

To compute the tree-cut width of G_m has, consider its tree-cut decomposition (Y^*, χ) where Y^* is a regular binary tree of depth m and χ is defined as follows. Let h: $V(Y^*) \to V(Y)$ and $h': V(Y^*) \to V(Y')$ be bijections such that (1) if y is a leaf of Y^* then h(y) and h'(y) are identified leaves of Y and Y', and (2) if y_1 is a parent of y_2 in Y^* then $h(y_1)$ is a parent of $h(y_2)$ in Y and $h'(y_1)$ is a parent of $h'(y_2)$ in Y'. Further, for every node y of Y^* we define its bag to be $X_y = \{h(y), h'(y)\}$ (see the right part of Figure 2 for the illustration). Observe that the adhesion of every node as well as size of each bag is at most 2, and all the children are thin, therefore, $tcw(G_m) = 2$. \Box

Since it is known that treewidth dominates tree-cut width (see Lemma 1), Lemma 2 implies that edge-cut width does not dominate degtw. Conversely, it is easy to build graphs with unbounded degtw and bounded edge-cut width (e.g., consider the class of stars). Hence, we obtain that edge-cut width is incomparable to degtw. An illustration of the parameter hierarchy including edge-cut width is provided in Figure 3.

Next, we note that even though Lemma 1 and Proposition 1 together imply that $tw(G) \le 2 ecw(G)^2 + 3 ecw(G)$, one can in fact show that the gap is linear. This will also allow us to provide a better running time bound in Section 4.

Lemma 3. For every graph G, $tw(G) \le ecw(G)$.



Fig. 3. Position of edge-cut width in the hierarchy of graph parameters. Here an arrow from parameter β to parameter α represents the fact that α dominates β , i.e., there exists a function p such that for each graph G, $\alpha(G) \leq p(\beta(G))$. We use fen to denote the feedback edge number.

Proof. Let T be the spanning tree of G such that ecw(G) = ecw(G, T). We arbitrarily pick a root r in T and construct the tree decomposition $(T, \{\beta_v\}_{v \in V(T)})$ of G as follows. At first, for every $v \in V(G)$, we add to β_v the vertex v and the parent of v in T (if it exists). Obviously, after this step each vertex v of G appears in some bag and every edge of T is contained as a subset in some bag. Moreover, v appears only in β_v and in the bags of children of v in T, which results in a connected subtree of T.

To complete the construction, we process feedback edges one by one. For every $e \in E(G) \setminus E(T)$, we arbitrarily choose an endpoint u of e = uw and add u to each bag β_v such that $u \in E_{\text{loc}}^{G,T}(v)$. Note that any such step does not violate the connectivity condition. Indeed, we add u to the bags of all vertices which lie on the path between the endpoints of e in T. In particular, the path hits u whose bag β_u initially contained u. Finally, both endpoints of e appear in β_w . In the resulting decomposition, for each $v \in V(G)$ it holds that $|\beta_v| \leq 2 + E_{\text{loc}}^{G,T}(v) \leq 1 + \text{ecw}(G)$. Hence the width of $(T, \{\beta_v\}_{v \in V(T)})$ is at most ecw(G).

Last but not least, we show that—also somewhat surprisingly— edge-cut width is not closed under edge or vertex deletion.

For the edge-deletion case, we refer readers to Figure 4 which illustrates a graph G along with a spanning tree witnessing that $ecw(G) \leq 4$. On the other hand, any spanning tree T of G - ac must contain both edges ab_i and b_ic for some $i \in \{1, 2, 3\}$. We will assume that those edges are ab_1 and b_1c , since the other cases are symmetrical. Then T contains precisely one edge of each pair (ab_2, b_2c) and (ab_3, b_3c) . The other, "missing" edge from each pair contributes to the local feedback edge set of b_1 . Together with two missing edges of 3-cycles that intersect b_1 , this results in $|E_{loc}^{G-ac,T}(b_1)| \geq 4$ and, since similar situation happens for any choice of a spanning tree, we conclude that $ecw(G - ac) \geq 5$. The vertex deletion case can be argued analogously using the graph obtained from G by subdividing the edge ac.

Corollary 1. There exist graphs G and H such that ecw(G - e) > ecw(G) and ecw(H - v) > ecw(H) for some $e \in E(G)$ and $v \in V(H)$.

4 Computing Edge-Cut Width

Before we proceed to the algorithmic applications of edge-cut width, we first consider the question of computing the parameter along with an optimal "decomposition" (i.e., spanning tree). Here, we provide an explicit fixed-parameter algorithm for this task.

By Lemma 3, the treewidth of G can be linearly bounded by ecw(G). The algorithm uses this to perform dynamic programming on a tree decomposition $(T, \{\beta_t\}_{t \in V(T)})$



Fig. 4. Left: Graph G - ac of $ecw(G - ac) \ge 5$. Right: Green tree witnessing that $ecw(G) \le 4$.

of G. For a node $t \in V(T)$, we let Y_t be the union of all bags β_s such that s is either t itself or a descendant of t in T, and let G_t be the subgraph $G[Y_t]$ of G induced by Y_t .

Lemma 4. Given an *n*-vertex graph G of treewidth k and a bound w, it is possible to decide whether G has edge-cut width at most w in time $k^{\mathcal{O}(wk^2)} \cdot n$. If the answer is positive, we can also output a spanning tree of G of edge-cut width at most w.

Using the relation between treewidth and edge-cut width above, we immediately obtain:

Theorem 1. Given a graph G, the edge-cut width ecw(G) can be computed time $ecw(G)^{ecw(G)^3} \cdot n$.

Proof (of Lemma 4). Without loss of generality, we assume that G is connected. Using state-of-the-art approximation algorithms [2, 25], we first compute a "nice" tree decomposition $(T, \{\beta_t\}_{t \in V(T)})$ with root $r \in V(T)$ of width $k = \mathcal{O}(\operatorname{tw}(G))$ in time $2^{\mathcal{O}(k)} \cdot n$.

On a high level, the algorithm relies on the fact that if G has edge-cut width at most w, then at each bag β_t the number of unique paths contributing to the edge-cut width of vertices in β_t is upper-bounded by $|\beta_t|w \le kw$. Otherwise, at least one of the vertices in β_t would lie on more than w cycles. We can use this to branch on how these at most kw edges are routed through the bag.

At each vertex $t \in T$ of the tree decomposition, we store *records* that consist of:

- an acyclic subset F of edges of $G[\beta_t]$,
- a partition C of β_t , and
- two multisets future, past of sequences of vertex-pairs (u, v) from β_t , with the following property:
 - Every vertex of β_t appears on at most w distinct u-v paths, where (u, v) is a pair of vertices in a sequence in future or past.
 - v_i and u_{i+1} are not connected by an edge in β_t .

The semantics of these records are as follows: For every spanning tree of width at most w, the record describes the intersection of the solution with $G[\beta_t]$, and the intersection of every fundamental cycle of this solution with $G[\beta_t]$. We encode the path that a cycle takes through $G[\beta_t]$ via a sequence of vertex pairs that indicate where the path leaves and enters $G[\beta_t]$ from the outside (it may be that these are the same vertex). More precisely, past contains those cycles that correspond to an edge that has already appeared in G_t , whereas future corresponds to those cycles that correspond to an edge not in G_t . In particular, this allows to reconstruct on how many cycles a vertex of β_t lies. The partition C says which vertices of β_t are connected via the solution in G_t .

To be more precise, let $t \in T$ and let S be an acyclic subset of edges of G that has width at most w on G_t (that is, each vertex of S lies on at most w fundamental cycles of S in G_t). We call such S partial solutions at t. Then, we let the t-projection of S be defined as (F, C, future, past), where

- $F = S \cap G[\beta_t].$
- C is a partition of F according to the connected components of S in G_t .
- Let C_e be a fundamental cycle of S in G corresponding to the edge $e \in G S$. Then, there is a sequence $P_e = ((u_1, v_1), \ldots, (u_t, v_t))$ in either future or past of vertex pairs such that the intersection of C_e with S traverses F along the unique $u_i \cdot v_i$ paths in the order they appear in P_e (note that $u_i = v_i$ is possible, in which case the path contains just the vertex u_i).
- For each fundamental cycle C_e of S in G, if $e \in G_t$, then $P_e \in \text{past}$, otherwise, $P_e \in \text{future}$.

Note that P_e can (and often will) be the empty sequence $P_e = \emptyset$. Moreover, we assume that the correspondence between future \cup past and the edges in G - S is bijective, in the sense that if two edges e, e' produce the same sequence $P_e = P_{e'}$, then P_e and $P_{e'}$ occur as two separate copies in future \cup past.

The encoding length of a single record is $\mathcal{O}(wk^2 \log k)$, dominated by the at most kw sequences P_e of k pairs of vertices each, with indices having $\mathcal{O}(\log k)$ bits. Overall, the number of records is hence bounded by $2^{\mathcal{O}(wk^2 \log k)}$.

For each $t \in T$, we store a set of records $\mathcal{R}(t)$ that has the property that $\mathcal{R}(t)$ contains the set of all *t*-projections of spanning trees of width at most w (that is, projections of solutions of the original instance). In addition, we require for every record in $\mathcal{R}(t)$ that there is a partial solution S of G_t of width at most w that agrees with F, \mathcal{C} and past of the record. In this case, we call $\mathcal{R}(t)$ valid. Supposing correctness of this procedure, G is a YES-instance if and only if $(F_r, \mathcal{C}_r, \texttt{past}_r, \texttt{future}_r) \in \mathcal{R}(r)$, with $F_r = \mathcal{C}_r = \texttt{future}_r = \emptyset, \texttt{past}_r = \{\emptyset^{m-n}\}$, and a NO-instance otherwise.

We compute $\mathcal{R}(t)$ bottom-up along the nice tree-decomposition depending on the type of the node t as follows:

At a leaf-node, per convention, $\beta_t = \emptyset$, and since G_t is the empty graph, any spanning tree S has width at most w on G_t . This implies that any t-projection (F, C, past, future) of such S satisfies $F = C = past = \emptyset$, future = $\{\emptyset^{n-m}\}$. It therefore suffices to set $\mathcal{R}(t) = \{(\emptyset, \emptyset, \emptyset, \{\emptyset^{n-m}\})\}$, and this is valid.

At an introduce-node, let the vertex introduced at t be $v \in G$, and let s be the unique child of t in T. By definition, $\beta_t = \beta_s \cup \{v\}$. We assume by inductive hypothesis that $\mathcal{R}(s)$ is valid. Consider now any solution S of width at most w on G_t . This solution will be of width at most w also on G_s . Hence, since $\mathcal{R}(s)$ is assumed valid, there is a record $(F_s, C_s, past_s, future_s)$ corresponding to the s-projection of S.

We first branch over the way that the edges incident with v in $G[\beta_t]$ extend F_s . Call this new set of edges E_v . During this process, we discard any choice of E_v that connects vertices within the same connected component as indicated by C_s .

Furthermore, we discard any choice that implies cycles in the solution via future: If there is an entry in future, that contains two consecutive pairs (u, u'), (w, w') such that u' and w are now in the same component of C (that is, were connected by adding v to G_s), and one of u' or w is not a neighbor of v, then this would imply two u'-w paths: u' and w, but not any of the vertices on the paths u'-v and v-w lie on the fundamental cycle corresponding to the entry in future_s containing (u, u'), (w, w'), yielding two paths: One through the cycle, the other through v via E_v . Therefore, this choice of E_v can be discarded.

Then, for every edge (v, u) incident with v that was not chosen into E_v , there must be a sequence of pairs P in future_s such that the last vertex in the last pair of the sequence P is u, otherwise we may discard E_v (since the corresponding fundamental cycle wasn't reflected in future_s.) We branch over all ways of choosing P_1, \ldots, P_d for each edge e_1, \ldots, e_d incident to v that is not in E_v . For each $i = 1, \ldots, d$, if $P = P_i$ just consists of the single pair (u, u), we add the single pair (v, v) to P, and move Pto past (since the feedback edge (v, u) is now part of G_t). Otherwise, if the first pair (w, w') in P is distinct from (u, u), we add the pair (v, w') to P, remove (w, w') from P, and add P to past.

We now update past and future as follows: If there is a consecutive pair (u, u'), (w, w') in an element of past_s or future_s such that u' and w are neighbors of v, replace the subsequence (u, u'), (w, w') by (u, w'): any other choice of connecting u' and w through a path than directly via v would imply a cycle. In any case, add the resulting sequence to past or future, respectively.

We then branch over the choices of extending fundamental cycles along v: For each pair in a sequence in past or future that contains a neighbor u of v connected via E_v , branch over whether or not to route this fundamental cycle via v by replacing (u, w) by (v, w) or (w, u) by (w, v), respectively.

If during any of the choices for E_v, P_1, \ldots, P_d and the extensions of the fundamental cycles via v, the solution would have to route more than w cycles over any vertex of β_t (as can be checked by tracing out all the pairs in the sequences now contained in future and past), discard the choice. If there is no way to choose the above without exceeding the width bound, discard the entire choice of record and consider the next record in $\mathcal{R}(s)$.

If this is not the case, then, for a choice of E_v (i.e., how to extend F_s), P_1, \ldots, P_d (i.e., how to route the new edges in G_t in past) and a choice of extending the existing cycles in past_s and future_s to in- or exclude v, we branch over how many additional fundamental cycles v outside of G_t will be part of, and add as many copies of the sequence consisting just of (v, v) to future, simultaneously decreasing the multiplicity of \emptyset in future_s by as many, and adding the result to future.

Finally, add $(F_s \cup E_v, C, past, future)$ to $\mathcal{R}(t)$, and consider the next entry of $\mathcal{R}(s)$. Since any partial solution of width at most w on G_t will have to extend its *s*-projection in one of the above ways, this generates all possible *t*-projections (and possibly some additional records with the same F, C, past). In particular, the generated set $\mathcal{R}(t)$ is valid. This completes the description of the introduce step.

The running time of this step is dominated by branching over the sequences P_1, \ldots, P_d . Since $d \leq k$ and there are at most kw sequences in total, we have $(kw)^k = 2^{\mathcal{O}(wk \log k)}$ choices at most, for each of the $2^{\mathcal{O}(wk^2 \log k)}$ records in $\mathcal{R}(s)$, and processing each choice only adds a lower-order term in the running time. Therefore, this step takes time $2^{\mathcal{O}(wk^2 \log k)}$.

At a forget-node, let the vertex forgotten at t be $v \in G$, and let s be the unique child of t in T. By definition, $\beta_t = \beta_s - \{v\}$. We assume by inductive hypothesis that $\mathcal{R}(s)$ is valid, and let $(F_s, \mathcal{C}_s, past_s, future_s) \in \mathcal{R}(s)$.

If $\{v\} \in C$ (that is, v is a single component in the intersection of any solution that projects to the current record with β_t), then discard the choice for the record and consider the next element of $\mathcal{R}(s)$. In this case, the component that contains v in any partial solution conforming with the record could never be completed to form a connected subgraph.

If (v, v) appears as part of a sequence in future_s or past_s, remove (v, v) from the sequence. If, on the other hand, (v, u) is part of any sequence in past_s or future_s for some $u \neq v$, replace (v, u) by (v', u), where v' is the next vertex on the unique v-upath in F_s (and u = v' is possible). In both cases, add the resulting sequence (which is possibly equal to the empty sequence) to future or past, respectively. If the empty sequence would be added to future, discard the current record (since there is no way of closing this fundamental cycle in the future that can involve v).

We remove all edges involving v from F_s to obtain F and update C_s by removing v from all sets it appears in, thereby obtaining C. We add (F, C, past, future) to $\mathcal{R}(t)$. Since $G_t = G_s$, the set of solutions that contribute to the set of t-projections and s-projections doesn't change; we hence only have to update the s-projections to become t-projections, as we did, in order to obtain a valid set $\mathcal{R}(t)$.

The running time of this step is dominated by the running time at the introduce-nodes. At a join-node, let s and s' be the two children of t in T. We consider all pairs of records in $\mathcal{R}(s)$ and $\mathcal{R}(s')$. If $F_s \neq F_{s'}$ or future_s \neq future_{s'}, we discard the current choice. Consider the transitive closures of the reachability relations on β_t as induced by \mathcal{C}_s and $\mathcal{C}_{s'}$, respectively. If their union (as multigraphs) produces a cycle (which could be two parallel edges (u, v) and (u, v) for some $u, v \in \beta_t = \beta_s = \beta_{s'}$), any solution that s-projected and s'-projected to \mathcal{C}_s and $\mathcal{C}_{s'}$, respectively, would be cyclic on G_t . Hence, we may discard this choice of records.

If none of the above happens, we set $past = past_s \cup past_{s'}$ as multisets, and check if this results in any of the vertices of β_t coming to lie on more than w fundamental cycles. If this is the case, we discard the current choice of records. If not, let C be finest common coarsening of the partitions C_s and $C_{s'}$ (that is, the result of merging any two components that share a vertex, and exhausting this process). We let $F = F_s(=F_{s'})$, future = future_s(= future_{S'}), and set $\mathcal{R}(t) = (F, C, past, future)$.

By a similar token as in the previous cases, this produces all possible t-projections of solutions of G_s and $G_{s'}$ that are also solutions for G_t of width at most w, and hence a valid set $\mathcal{R}(t)$.

Since we have to consider pairs of records that differ in past, and past dominates the size of the records, the running time at the join-nodes dominates the running time at the introduce-nodes, and is bounded by $2^{\mathcal{O}(wk^2 \log k)}$.

Overall, the running time of the algorithm is bounded by $2^{\mathcal{O}(wk^2 \log k)} \cdot n$. By keeping one representative of a partial solution of G_t per record at each node t that t-projects to the current record, we can successively build a solution of width at most w.

5 Algorithmic Applications of Edge-Cut Width

Here we obtain algorithms for the following five NP-hard problems (where a sixth problem mentioned in the introduction, PRECOLORING EXTENSION, is a special case of LIST COLORING, and the fixed-parameter tractability of BAYESIAN NETWORK STRUCTURE LEARNING and POLYTREE LEARNING follows from previous work [16]). In all of these, we will parameterize either by the edge-cut width of the input graph or of a suitable graph representation of the input. Recall that all problems are known to be W[1]-hard when parameterized by tree-cut width [5, 14, 18, 22], and here we will show they are all fixed-parameter tractable w.r.t. edge-cut width.

As a unified starting point for all algorithms, we will apply Theorem 1 to compute a minimum-width spanning tree T of the input graph (or the graph representation of the input) G; the running time of Theorem 1 is also an upper-bound for the running time of all algorithms except for MAXSRTI, which has a quadratic dependence on the input size. Let r be an arbitrarily chosen root in T. For each node $v \in V(T)$, we will use T_v to denote the subtree of T rooted at v. Without loss of generality, in all our problems we will assume that G is connected.

The central notion used in our dynamic programming framework is that of a *bound*ary, which fills a similar role as the bags in tree decompositions. Intuitively, the boundary contains all the edges which leave T_v (including the vertices incident to these edges).

Definition 5. For each $v \in V(T)$, the boundary $\partial(v)$ of T_v is the edge-induced subgraph of G induced by those edges which have precisely one endpoint in T_v .

Observe that for each $v \in V(T)$, $|E(\partial(v))| \le ecw(G)$ and $|V(\partial(v))| \le 2 ecw(G)$. It will also sometimes be useful to speak of the graph induced by the vertices that are "below" v in T, and so we set $\mathcal{Y}_v = \{w \mid w \text{ is a descendant of } v \text{ in } T\}$ and $\mathcal{G}_v = G[\mathcal{Y}_v]$; we note that $v \in \mathcal{Y}_v$. Observe that $\partial(v)$ acts as a separator between vertices outside of $\mathcal{Y}_v \cup V(\partial(v))$ and vertices in $\mathcal{Y}_v \setminus V(\partial(v))$

5.1 Edge Disjoint Paths

We start with the classical EDGE DISJOINT PATHS problem, which has been extensively studied in the literature. While its natural counterpart, the VERTEX DISJOINT PATHS problem, is fixed-parameter tractable when parameterized by treewidth, EDGE DISJOINT PATHS is W[1]-hard not only when parameterized by tree-cut width [18] but also by the vertex cover number [11].

EDGE DISJOINT PATHS (EDP)		
Input:	A graph G and a set P of <i>terminal pairs</i> , i.e., a set of subsets of $V(G)$ of size two.	
Question	: Is there a set of pairwise edge disjoint paths connecting every set of terminal pairs in P ?	

A vertex which occurs in a terminal pair is called a *terminal* and a set of pairwise edge disjoint paths connecting every set of terminal pairs in P is called a *solution*.

Theorem 2. EDP *is fixed-parameter tractable when parameterized by the edge-cut width of the input graph.*

Proof. We start by defining the syntax of the records we will use in our dynamic program. For $v \in V(G)$, let a record be a tuple of the form (S, D, R), where:

- $S = \{(t_0, e_0), \dots, (t_i, e_i)\}$ where for each $j \in [i], t_j \in \mathcal{Y}_v$ is a terminal whose counterpart is not in $\mathcal{G}_v, e_j \in E(\partial(v))$, and where each terminal without a partner in \mathcal{Y}_v appears in exactly one pair,
- D, R are sets of unordered pairs of elements from $E(\partial(v))$, and
- each edge of $E(\partial(v))$ may only appear in at most one tuple over all of these sets.

We refer to the edges in S, D, R as *single, donated* and *received* edges, respectively, in accordance with how they will be used in the algorithm. Let $\mathcal{R}(v)$ be a set of records for v. From the syntax, it follows that $|\mathcal{R}(v)| \leq 2^{\mathcal{O}(k \log k)}$ for each $v \in V(G)$.

Let $P_v \subseteq (\mathcal{Y}_v \cup V(\partial(v)))^2$ be a set that can be obtained from P by the following three operations:

- for some $\{a, b\} \in P$ where $a \in \mathcal{Y}_v, b \notin \mathcal{Y}_v$, replacing b by some $c \in V(\partial(v))$, and
- for some $a', b' \in V(\partial(v)) \setminus \mathcal{Y}_v$, adding $\{a', b'\}$ to P_v , and
- for each $\{a, b\} \in P$ where $a, b \notin \mathcal{Y}_v$, remove $\{a, b\}$.

To define a partial solution we need the following graph H_v :

- First, we add $\mathcal{G}_v \cup \partial(v)$ to H_v , where $\mathcal{G}_v \cup \partial(v)$ is the (non-disjoint) union of these two graphs.
- Next, we create for each edge e ∈ E(∂(v)) a pendant vertex v_e adjacent to the endpoint of e that is outside of Y_v. Let V_∂ denote the set of these new vertices.
- Finally, we add edges to $E(H_v)$ such that V_{∂} is a clique.

Let a partial solution at v be a solution to the instance (H_v, P_v) for some P_v defined as above. Obviously, since at the root r we have that $\partial(r)$ is empty, $P_r = P$ and $H_r = G$. Notice that a partial solution at the root is a solution.

Consider then the set W containing all partial solutions at v. The *v*-projection of a partial solution $W \in W$ at v is a record (S_W, D_W, R_W) where:

- $(t, e) \in S_W$ if and only if t is a terminal in \mathcal{Y}_v whose counterpart t' is not in \mathcal{Y}_v and e is the first edge in $E(\partial(v))$ encountered by the t-t' path in W,
- $\{e_i, e_j\} \in D_W$ if and only if there is a path $Q \in W$ with $Q = e_i, e_{i+1}, \ldots, e_{j-1}, e_j$ such that the edges in $Q \setminus \{e_i, e_j\}$ are contained in $E(\mathcal{G}_v)^3$, and
- $\{e_i, e_j\} \in R_W$ if and only if there is some *s*-*t* path $Q \in W$ such that *s*, *t* in \mathcal{Y}_v, e_i is the first edge in $E(\partial(v))$ that occurs in Q, and e_j is the last edge in $E(\partial(v))$ that occurs in Q.

We say that $\mathcal{R}(v)$ is *valid* if and only if it contains all *v*-projections of partial solutions in \mathcal{W} , and in addition, for every record in $\mathcal{R}(v)$, there is a partial solution such that its *v*-projection yields this record.

Observe that if $\mathcal{R}(r) = \emptyset$, then (G, P) is a NO-instance, while if $\mathcal{R}(r) = \{(\emptyset, \emptyset, \emptyset)\}$, then R(r) is a YES-instance. To complete the proof, it now suffices to dynamically compute a set of valid records in a leaf-to-root fashion along T. We note that if at any stage we obtain that a vertex v has no records (i.e., $\mathcal{R}(v) = \emptyset$), we immediately reject.

³ Note that by the syntax, it follows that e_i and e_j are both contained in $\partial(v)$

If v is a leaf, we branch over all possible valid records by setting $R = \emptyset$ and letting D vary over all subsets of $\{\{e_1, e_2\} \mid e_1, e_2 \in E(\partial(v))\}$. In the case that v is a terminal, we additionally let S vary over all subsets of $\{(v, e) \mid e \in E(\partial(v))\}$. We discard choices of S and D where the same edge appears more than once over both sets.

The set $\mathcal{R}(v)$ is trivially valid.

If v is an internal node, we proceed in the following way: First, we bound the number of children of v by our parameter k. Then we branch over all possible combinations of records for the remaining children of v to obtain $\mathcal{R}(v)$.

We reduce the size of the subtree in the following way: Let u be a child of v with $E(\partial(u)) = \{\{u, v\}\}, \text{ i.e., } T_u \text{ has no edge that increases the size of the edge-cut width of <math>v$.

- If there is no terminal pair with precisely one vertex in T_u , then delete T_u along with all terminal pairs with both endpoints in T_u .
- If there is a single terminal pair {s, t} with precisely one vertex, say t, in T_u, then replace t with v and delete T_u along with all terminal pairs with both endpoints in T_u. (We remark that v can be contained in multiple terminal pairs at the same time.)
 Otherwise, we correctly identify that this is a NO-instance.

Since \mathcal{G}_u is connected to the remaining graph by a single edge it can connect only one terminal in \mathcal{Y}_u with a terminal in $V \setminus \mathcal{Y}_u$. After this step there are at most 2(k-1)children left because at most 2(k-1) subtrees rooted at a child of v can contribute to the edge-cut width of v.

Let u_1, \ldots, u_ℓ with $\ell \leq 2(k-1)$ denote the remaining children of v. First, we compute a set $\overline{\mathcal{R}}(v)$, in the same way we would compute $\mathcal{R}(v)$ if v was a leaf. Our goal is to compute $\mathcal{R}(v)$ using the local set $\overline{\mathcal{R}}(v)$ and the partial results $\mathcal{R}(u_1), \ldots, \mathcal{R}(u_\ell)$.

In the following we take one record each out of $\mathcal{R}(v), \mathcal{R}(u_1), \ldots, \mathcal{R}(u_\ell)$ and repeat the following process for each combination of records. First, we observe that each edge can appear in at most two records, because it can connect at most two subtrees.

In the next step, we compute a set D', which contains the longest paths which can be donated by T_v , for each combination of records. For this we look at the *D*-sets in our records from $\overline{\mathcal{R}(v)}, \mathcal{R}(u_1), \ldots, \mathcal{R}(u_\ell)$. We trace out the longest paths along edges in the *D*-sets of these records, which can be done in time $k^{\mathcal{O}(1)}$ (we start at some edge e_i and find its partner e_j in the same *D*-set, then we look for e_j in the other *D*-sets, and so on; in particular, this is not an ordinary longest-path problem).

Now, we resolve each of the pairs $\{e_i, e_j\}$ in R for any of the currently considered records using the paths in D'. Either there is a path in D' connecting e_i and e_j , which means the pair $\{e_i, e_j\}$ can be ignored. Or there are two paths connecting e_i resp. e_j to e'_i resp. $e'_j \in E(\partial(v))$. Then the pair $\{e'_i, e'_j\}$ needs to be added to R'. In case $e_i \in E(\partial(v))$, let $e'_i = e_i$ and similarly for e_j . In either case the used paths are deleted from D'.

Next, we consider each pair $(s, e_i) \in S$ for any of the currently considered records. Let $(s,t) \in P$. If $t \notin \mathcal{Y}_v$ and $e_i \in E(\partial(v))$, then add (s, e_i) to S'. In case $e_i \notin E(\partial(v))$, we use the donated paths in D' to connect e_i to $e'_i \in E(\partial(v))$ and add (s, e'_i) to S'. If $(t, e_i) \in \overline{S}$ for any of the currently considered records, we proceed as if $(e_i, e_i) \in R$.

Note that all steps are deterministic, as each edge can only appear in two sets and therefore there can only be one path starting at any edge e that one could use to traverse the graph.

Afterwards, we need to delete all pairs in D' with e_i or $e_j \notin E(\partial(v))$. Finally, the tuple (S', D', R') is inserted as a record in $\mathcal{R}(v)$.

Correctness follows via induction: The records of the leaves are valid. Assuming $\mathcal{R}(u_1), \ldots, \mathcal{R}(u_\ell)$ are valid, so will be the record set at v: It contains all possible ways in which the partial solutions of the subtrees at u_i could be extended. In particular, this includes the projections of all full solutions, and by construction, every such extension will extend the combination of partial solutions of the subtrees to a partial solution of the subtree at v, showing validity.

As for the running time: We go through each of the *n* vertices, where $|\mathcal{R}_v| \leq 2^{\mathcal{O}(k \log k)}$ for $v \in V$. Moreover, each vertex has at most 2(k-1) children, which makes for $2^{\mathcal{O}(k^2 \log k)}$ combinations when branching, and the number of combinations dominates the time each combination takes to be processed. Hence, the total running time amounts to $2^{\mathcal{O}(k^2 \log k)} \cdot n$.

5.2 List Coloring

The second problem we consider is LIST COLORING [9,14]. It is known that this problem is W[1]-hard parameterized by tree-cut width. A *coloring* col 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 $col(a) \neq col(b)$.

LIST COLORING Input: A graph G = (V, E) and for each vertex $v \in V$ a list L(v) of permitted colors. Question: Does G admit a proper coloring col where for each vertex v it holds $col(v) \in L(v)$?

Theorem 3. LIST COLORING is fixed-parameter tractable when parameterized by the edge-cut width of the input graph.

Proof. We start by defining the syntax of the records we will use in our dynamic program. For $v \in V(G)$, let a record for a vertex v consist of tuples of the form (u, c), where (1) $u \in V(\partial(v)) \cap \mathcal{Y}_v$, (2) $c \in L(u) \cup \{\delta\}$, and (3) each vertex of $V(\partial(v)) \cap \mathcal{Y}_v$ appears exactly once in a record.

To introduce the semantics of the records, consider the set \mathcal{W} containing all partial solutions (i.e., all proper colorings) at v to the instance $(\mathcal{G}_v, (L(u))_{u \in \mathcal{Y}_v})$. The *v*-projection of a partial solution $col \in \mathcal{W}$ is a set $R_{col} = \{(u, c) \mid u \in V(\partial(v)) \cap \mathcal{Y}_v, c \in L(u)\})$ where $(u, c) \in R_{col}$ if and only if col(u) = c.

Let $\mathcal{R}(v)$ be a set of records for v. For two records $R_1, R_2 \in \mathcal{R}(v)$ we say $R_1 \preceq R_2$ if and only if for each $u \in V(\partial(v)) \cap \mathcal{Y}_v$ the following holds:

- Either $(u, c) \in R_1 \cap R_2$ with $c \in L(u)$,

- Or $(u, c) \in R_1$ with $c \in L(u)$ and $(u, \delta) \in R_2$.

We say that $\mathcal{R}(v)$ is *valid* if for each *v*-projection R_{col} of a partial solution $col \in \mathcal{W}$ there is a record $R \in \mathcal{R}(v)$ which satisfies $R_{col} \preceq R$, and in addition, for every record $R \in \mathcal{R}(v)$, there is a partial solution $col \in \mathcal{W}$ such that its *v*-projection fulfills $R_{\text{col}} \preceq R$. Observe that if $\mathcal{R}(r) = \emptyset$, then $(G, (L(v))_{v \in V(G)})$ is a NO-instance, while if $\mathcal{R}(r) = \{\emptyset\}$, then R(r) is a YES-instance.

If a record in $\mathcal{R}(v)$ contains a tuple (u, δ) , then this means that there is always a possible coloring for the vertex u, e.g., if $|L(u)| > d_G(u)$; the symbol δ is introduced specifically to bound |L(v)|. Therefore, it follows that $|\mathcal{R}(v)| \leq 2^{\mathcal{O}(k \log k)}$ for each $v \in V(G)$. To complete the proof, it now suffices to dynamically compute a set of valid records in a leaf-to-root fashion along T.

If v is a leaf, we set $\mathcal{R}(v) = \{\{(v, \delta)\}\}\$ for the case $|L(v)| > d_G(v)$. Otherwise, we branch over all possible colorings of the vertex v, i.e., $\mathcal{R}(v) = \{\{(v, c)\} | c \in L(v)\}\$. Note that the amount of records is always bounded by k, as $d_G(v) \leq k$.

If v is an internal node, we start with reducing the size of the subtree T_v in the following way: Let u be a child of v with $E(\partial(u)) = \{(u, v)\}$.

- If $\mathcal{R}(u) = \{\{(u, c)\}\}\$ with $c \neq \delta$, then remove c from L(v).

- Delete T_u .

After this step there are at most 2(k-1) children of v left. Let u_1, \ldots, u_ℓ with $\ell \leq 2(k-1)$ denote the remaining children of v. First, we compute a set $\overline{\mathcal{R}(v)}$, in the same way we would compute $\mathcal{R}(v)$ if v was a leaf. Our goal is to compute $\mathcal{R}(v)$ using the local set $\overline{\mathcal{R}(v)}$ and the partial results $\mathcal{R}(u_1), \ldots, \mathcal{R}(u_\ell)$. Note that since $d_G(v) \leq 2k-1$ the number of records in $\overline{\mathcal{R}(v)}$ is also bounded by 2k-1.

In the next step we take one record each out of $\mathcal{R}(v), \mathcal{R}(u_1), \ldots, \mathcal{R}(u_\ell)$ and branch over all possible combination of records. Then we check for each combination if the coloring of the vertices in $\mathcal{Y}_{u_1}, \ldots, \mathcal{Y}_{u_\ell}$ can be combined to a proper coloring of the vertices in \mathcal{Y}_v . For this we only need to consider the vertices in $\partial(u_1), \ldots, \partial(u_\ell)$ and check if two neighbors share the same color. If this is not possible, then move on to the next combination of records.

Afterwards, we need to remove all the vertices, which are not in $V(\partial(v)) \cap \mathcal{Y}_v$. The remaining vertices and their colors form a record of T_v .

Since $|\mathcal{R}(v)| \leq 2^{\mathcal{O}(k \log k)}$ and the size of each record is bounded by $\mathcal{O}(k)$, the running time is bounded by $2^{\mathcal{O}(k^2 \log k)} \cdot n$.

5.3 Boolean CSP

Next, we consider the classical constraint satisfaction problem [32]. An instance I of BOOLEAN CSP is a tuple (X, C), where X is a finite set of *variables* and C is a finite set of *constraints*. Each constraint in C is a pair (S, R), where the *constraint scope* S is a non-empty sequence of distinct variables of X, and the *constraint relation* R is a relation over $\{0, 1\}$ (given as a set of tuples) whose arity matches the length of S. An *assignment* is a mapping from the set X of variables to $\{0, 1\}$. An assignment σ satisfies a constraint $C = ((x_1, \ldots, x_n), R)$ if $(\sigma(x_1), \ldots, \sigma(x_n)) \in R$, and σ satisfies the BOOLEAN CSP instance if it satisfies all its constraints. An instance I is satisfiable if it is satisfied by some assignment.

BOOLEAN	CSF
DOODDING	~~-

Input:	A set of variables X and a set of constraints C .
Question:	Is there an assignment $\sigma: X \to \{0,1\}$ such that all constraints in $\mathcal C$ are
	satisfied?

We represent this problem via the *incidence graph*, whose vertex set is $X \cup C$ and which contains an edge between a variable and a constraint if and only if the variable appears in the scope of the constraint.

Theorem 4. BOOLEAN CSP *is fixed-parameter tractable when parameterized by the edge-cut width of the incidence graph.*

Proof. For this problem, we do not need to consider all the vertices in the boundary. Instead, for a vertex $v \in V$, let $B(v) = V(\partial(v)) \cap \mathcal{Y}_v \cap X$. Hence, we will consider only the vertices in the boundary inside of the current subtree, which correspond to variables in the input instance. Note that $|B(v)| \leq |V(\partial(v))| \leq 2k$.

We continue with defining the syntax of the records we will use in our dynamic program. For $v \in V(G)$, let a record for a vertex v be a set of functions of the form $\varphi: B(v) \to \{0, 1\}$. Let $\mathcal{R}(v)$ be a set of records for v. From the syntax, it follows that $|\mathcal{R}(v)| \leq 2^{\mathcal{O}(k)}$ for each $v \in V(G)$. To introduce the semantics of the records, consider the set \mathcal{W} containing all partial solutions (i.e., all assignments of the variables such that every constraint is fulfilled) at v for the instance $(\mathcal{Y}_v \cap X, \mathcal{Y}_v \cap \mathcal{C})$.

The function φ is a *v*-projection of a solution $\sigma \in W$ if and only if $\sigma|_{B(v)} = \varphi$. This means, that the functions in a record represent the assignments of variables, which are compatible with \mathcal{Y}_v .

We say that $\mathcal{R}(v)$ is *valid* if it contains all *v*-projections of partial solutions in \mathcal{W} , and in addition, for every record in $\mathcal{R}(v)$, there is a partial solution such that its *v*-projection yields this record. Observe that if $\mathcal{R}(r) = \emptyset$, then (X, \mathcal{C}) is a NO-instance, while if $\mathcal{R}(r) = \{\emptyset\}$, then R(r) is a YES-instance. To complete the proof, it now suffices to dynamically compute a set of valid records in a leaf-to-root fashion along *T*.

If v is a leaf and $v \in X$, we can remove v in case $d_G(v) = 1$. Otherwise, we set $B(v) = \{v\}$ and all assignments are valid, i.e., $\mathcal{R}(v) = \{\varphi : \{v\} \rightarrow \{0, 1\}\}$.

If v is a leaf and $v \in C$, then $B(v) = \emptyset$, which means $\mathcal{R}(v) = \{\emptyset\}$.

If v is an internal node, we start with bounding the number of children of v. We have to distinguish, if v corresponds to a variable or a constraint. Let u be a child of v.

- For $v \in X$ and $B(u) = \emptyset$, check whether $\mathcal{R}(u)$ allows both values for the variable v. If not we fix the value as seen in the previous case. Afterwards delete u.
- For $v \in C$ and $B(u) = \{u\}$, use $\mathcal{R}(u)$ to check all viable assignments to the root and then remove the unsatisfiable ones from the constraint v. Afterwards delete u.
- If after this we obtain an empty constraint or a conflict with the variable assignment occurs, we know that this is a NO-instance.

After this step there are at most 2(k-1) children left. Let u_1, \ldots, u_ℓ with $\ell \leq 2(k-1)$ denote the remaining children of v. To obtain $\mathcal{R}(v)$, we can brute force all viable combinations of $\mathcal{R}(u_1), \ldots, \mathcal{R}(u_\ell)$.

Since the number of records and the size of each record is bounded by k, this algorithm runs in time $2^{\mathcal{O}(k^2)} \cdot n$.

5.4 Maximum Stable Roommates with Ties and Incomplete Lists

Our fourth problem originates from the area of computational social choice [5]. In this problem we are given a set of *agents* V, where each agent $v \in V$ has a *preference*

 $\mathcal{P}_v = (P_v, \preceq_v)$. The agents $P_v \subseteq V \setminus \{v\}$ are called *acceptable (for v)* and \preceq_v is a linear order on P_v with ties. Let $u, w \in P_v$. If $u \prec_v w$ then we say that v strongly prefers u to w; on the other hand, if $u \prec_v w$ does not hold then we say that v weakly prefers w to u.

We represent this problem via the undirected *acceptability graph* G, which contains a vertex for each agent in V and an edge between two agents if and only if both appear in the preference lists of the other.

A set $M \subseteq E(G)$ is called a *matching* if no two edges in M share an endpoint. If the edge $\{v, w\}$ is contained in M, then we say v is *matched* to w and denote this as M(v) = w and vice versa. In case a vertex v is not incident to any edge in M, then v is *unmatched* resp. $M(v) = \bot$ (where we assume \bot to be less preferable than all acceptable neighbors of v). An edge $\{v, w\} \in E(G) \setminus M$ is *blocking* for M (we also say v, w form a *blocking pair*) if $w \prec_v M(v)$ and $v \prec_w M(w)$. A matching is *stable* if it does not admit a blocking pair.

MAXIMUM STABLE ROOMMATES WITH TIES AND INCOMPLETE LISTS (MAXS-RTI)

Input: A set of agents V, a preference profile $\mathcal{P} = (\mathcal{P}_v)_{v \in V}$, and an integer π . Question: Is there a stable matching of (V, P) of cardinality at least π ?

Theorem 5. MAXSRTI *is fixed-parameter tractable when parameterized by the edgecut width of the acceptability graph.*

Proof. We once again start by defining the syntax of the records. For $v \in V(G)$, let a *signature* at v be a mapping $E(\partial(v)) \rightarrow \{ \text{matched}, \text{unsafe}, \text{safe} \}$. Clearly, the number of signatures at v is upper-bounded by 3^k , where k = ecw(G).

To make it easier to describe the semantics of the records, let us first define the graph \mathcal{H}_v as the non-disjoint union of \mathcal{G}_v and $\partial(v)$; we recall that $\partial(v)$ contains both vertices in \mathcal{G}_v and vertices adjacent to these, and that $E(\partial(v))$ forms an edge-cut separating \mathcal{G}_v from the rest of G.

We are now ready to define the semantics of the records. A matching M in \mathcal{H}_v is called a *partial solution* if there is no blocking edge for M in $E(\mathcal{H}_v)$; in other words, we explicitly forbid the edges in the boundary from forming blocking pairs in partial solutions. Each partial solution M corresponds to a signature signative signation of the edges follows:

- for each $e \in M \cap E(\partial(v))$, sig(e) = matched,
- for each $e = ab \in E(\partial(v)) \setminus M$ such that $a \in \mathcal{Y}_v$ and there exists $ac \in M$ such that $c \prec_a b$, sig(e) = safe, and
- sig(e) = unsafe otherwise.

Intuitively, the signature of M captures the following information about M: which edges in the boundary are matched, and for those which are not matched it stores whether they are "safe" (meaning that the endpoint in \mathcal{Y}_v will never form a blocking pair with that edge), or "unsafe" (meaning that the endpoint in \mathcal{Y}_v could later form a blocking pair with that edge, depending on the preferences and matching of the endpoint outside of \mathcal{Y}_v).

We define $\operatorname{Record}(v)$ to be a mapping from the set of all signatures at v to $\mathbb{N} \cup \{-\infty\}$, where (1) if there is no partial solution corresponding to a signature τ , then $\operatorname{Record}(v)(\tau) \mapsto -\infty$, and otherwise (2) $\operatorname{Record}(v)$ maps τ to the size of the largest partial solution in \mathcal{H}_v whose signature is τ . To avoid any confusion, we remark that when applying addition to the images of $\operatorname{Record}(v)$, we let $-\infty + x = -\infty$ for each $x \in \mathbb{N} \cup \{-\infty\}$.

If we can compute $\operatorname{Record}(r)$ for the root r of a spanning tree T witnessing that $\operatorname{ecw}(G,T) \leq k$, then by definition each partial solution is also a stable matching in the instance. Hence, it suffices to check whether $\operatorname{Record}(r)(\emptyset) \geq \pi$; if this is the case then we output "Yes", and otherwise we can safely output "No". At this point, it suffices to compute $\operatorname{Record}(v)$ for each $v \in V(G)$ in leaf-to-root fashion along T.

If v is a leaf, we first add the mapping $(E(\partial(v)) \mapsto unsafe) \mapsto 0$ to $\operatorname{Record}(v)$, which corresponds to the empty partial solution. Then, for each $vw \in E(\partial(v))$ we construct a signature τ_w which assigns vw to matched, and for each neighbor u of v other than w either assigns vu to safe (if v weakly prefers w to u) or assigns vuto unsafe (if v strongly prefers u to w). For each τ_w constructed in this way, we set $\operatorname{Record}(v)(\tau_w) = 1$.

If v is an internal node, we begin by branching over all edges incident to v, and for each such edge vw we proceed by restricting our attention to all partial solutions which contain vw. We also have a separate branch to deal with all partial solutions where v remains unmatched; we will begin by dealing with this (slightly simpler) case.

Subcase: v remains unmatched. For each child u of v such that $E(\partial(u)) = \{(u, v)\}$, we observe that only partial solutions at u with the signature $uv \mapsto \text{safe}$ can be extended to a partial solution at v; indeed, $uv \mapsto \text{matched}$ would violate our assumption that v remains unmatched, while $uv \mapsto \text{unsafe}$ would, by definition, lead to a blocking pair. For brevity, let us set simple-size to be the sum of all $\text{Record}(u)(uv \mapsto \text{unsafe})$ over all vertices u with a single-edge boundary.

As in the previous algorithms, we observe that at this point only at most 2k children of v remain to be processed, say x_0, \ldots, x_ℓ . We proceed by simultaneously branching over all of the at most 3^k signatures for each of these children, resulting in a total branching factor of 3^{k^2} ; each branch can be represented as a tuple $(\text{sig}_{x_0}, \ldots, \text{sig}_{x_\ell})$. We now discard all tuples that are not well-formed, where a tuple is well-formed if the following conditions hold:

- it contains no signature that maps an edge incident to v to either unsafe or matched (as before, these edges may only be mapped to safe);
- for each edge ab such that $a \in \mathcal{Y}_{x_i}$ and $b \in \mathcal{Y}_{x_j}$, $i, j \in [\ell]$, the signatures of x_i and x_j must either (a) both map that edge to matched, or (b) both map that edge to safe, or (c) map that edge to safe once and unsafe once (signatures must be consistent).

For all remaining tuples, we set branching-size to $\sum_{i \in [\ell]} \text{Record}(x_i)(\text{sig}_{x_i})$. We also identify a unique signature sig^* corresponding to the current branch as follows: each edge in $\partial(v)$ incident to v is mapped to unsafe, and each edge e in $\partial(v)$ not incident to v must have an endpoint in \mathcal{Y}_{x_i} for some x_i and is mapped to $\text{sig}_{x_i}(e)$. At this point, we update $\text{Record}(v)(\text{sig}^*)$ as follows: if the value of $\text{Record}(v)(\text{sig}^*)$ computed so far is greater than simple-size + branching-size then we do nothing, and otherwise we set that value to simple-size+branching-size. We now proceed to the next branch, i.e., choice of neighbor of v.

Subcase: v is matched to w. We will in principle follow the same steps as in the previous subcase, but with a few extra complications. Let us begin by distinguishing whether (1) w itself is a child of v such that $E(\partial(u)) = \{(u, v)\}, (2) w$ is in \mathcal{Y}_{x_i} for some child x_i of v not satisfying this property (including the case where $w = x_i$), or (3) $w \notin \mathcal{Y}_v$. In the first case, we set the child w aside and initiate simple-size = Record(w)($wv \mapsto matched$). In the second case, we will later (in the appropriate branching step) discard all signatures of x_i which do not map wv to matched. In the third case, we will take this into account when constructing sig^{*}.

Next, for each child u of v such that $E(\partial(u)) = \{(u, v)\}$ (other than w, in case (1)), we distinguish whether v weakly prefers w to u, or not. For each u where this holds, we observe that any partial solution at u that does not use v can be safely extended to a partial solution at v—hence, we increase simple-size by max(Record(u)($vu \mapsto unsafe$), Record(u)($vu \mapsto safe$)). On the other hand, for each u where v strongly prefers u to w we observe that a partial solution at u can only be extended to one at v if it matches u in a way which prevents the creation of a blocking pair with v. Hence, in this case, we increase simple-size by Record(u)($vu \mapsto safe$).

In the second step, we once again proceed by simultaneously branching over all of the at most 3^k signatures for the remaining children x_0, \ldots, x_ℓ of v. As before, this results in a total branching factor of 3^{k^2} , and each branch can be represented as a tuple $(sig_{x_0}, \ldots, sig_{x_\ell})$. We now discard all tuples that aren't well-formed, where a tuple is well-formed if the following conditions hold:

- for each edge ab such that $a \in \mathcal{Y}_{x_i}$ and $b \in \mathcal{Y}_{x_j}$, $i, j \in [\ell]$, the signatures of x_i and x_j must either (a) both map that edge to matched, or (b) both map that edge to safe, or (c) map that edge to safe once and unsafe once (signatures must be consistent);
- in case (2), the edge vw is mapped to matched in the appropriate signature;
- the tuple contains no signature that maps any edge incident to v (other than vw) to matched;
- for no edge vz where $z \in \mathcal{Y}_{x_i}$ for some $i \in [\ell]$ such that v strongly prefers z to w, the signature of x_i maps vz to unsafe (as this would create a blocking pair).

For all remaining tuples, we set branching-size to $\sum_{i \in [\ell]} \operatorname{Record}(x_i)(\operatorname{sig}_{x_i})$ in cases (1) and (2); in case (3), we set it to $\sum_{i \in [\ell]} \operatorname{Record}(x_i)(\operatorname{sig}_{x_i}) + 1$. We also identify a unique signature sig^* corresponding to the current branch as follows: each edge $vc \in E(\partial(v))$ is mapped to unsafe if v strongly prefers c to w, and safe otherwise (with the exception of c = w in case (3), where vw must be mapped to matched). Furthermore, each edge e in $\partial(v)$ not incident to v must have an endpoint in \mathcal{Y}_{x_i} for some x_i and is mapped to $\operatorname{sig}_{x_i}(e)$. At this point, we update $\operatorname{Record}(v)(\operatorname{sig}^*)$ as follows: if the value of $\operatorname{Record}(v)(\operatorname{sig}^*)$ computed so far is greater than $\operatorname{simple-size} + \operatorname{branching-size}$. We then proceed to the next branch, i.e., choice of neighbor of v.

The correctness of the algorithm can be shown by induction; it is not difficult to verify that the computation of the records is correct at the leaves, and for non-leaves one uses the assumption that the records of the children are correct. The crucial point is that every partial solution at a child that corresponds to a certain signature can be extended to a partial solution at the parent if the verified conditions hold, which justifies the correctness of adding up the appropriate values for the children. The running time is upper-bounded by $3^{k^2} \cdot n^2$.

5.5 Minimum Changeover Cost Arborescence

The final problem we consider can be found in [22]. An *arborescence* is a directed tree with root r, which contains a directed path from each vertex to r.

Given an arborescence T with root r and an edge $e \in E(T)$ we denote with succ(e) the edge incident to e on the path from v to the root r. For an edge e incident to the root we define succ(e) = e.

A function $cost : X^2 \to \mathbb{N}$ is called a *changeover cost function* if it satisfies the following:

1. $cost(x_1, x_2) = cost(x_2, x_1)$ for each $x_1, x_2 \in X$, and

e

2. cost(x, x) = 0 for each $x \in X$.

The *total changeover costs* of an arborescence T are now defined as

$$\sum_{\in E(T)} \operatorname{cost}(e, \operatorname{succ}(e)).$$

MINIMUM CHANGEOVER COST ARBORESCENCE (MINCCA)

Input: A directed graph G = (V, E), a root $r \in V(G)$, an edge coloring $col : E(G) \to X$, and a changeover cost function $cost : X^2 \to \mathbb{N}$. Question: What is an arborescence of G minimizing the total changeover costs?

The edge-cut width of a directed graph G is the edge-cut width of G where we omit the arc directions.

Theorem 6. MINCCA *is fixed-parameter tractable when parameterized by the edge-cut width of the input graph.*

Proof. We start by defining the syntax of the records we will use in our dynamic program. For $v \in V(G)$, let a record for a vertex v be a tuple of the form (Outgoing, Donate), where:

- Outgoing = $\{(v_0, e_0), \dots, (v_i, e_i)\}$ where for each $j \in [i], v_j \in V(\partial(v)) \cap \mathcal{Y}_v, e_j \in E(\partial(v))$, and
- Donate = { $(v_0, c_0, e_0), \ldots, (v_i, c_i, e_i)$ } where for each $j \in [i], v_j \in V(\partial(v)) \cap \mathcal{Y}_v, c_j \in X$, and $e_j \in E(\partial(v))$.

Moreover, let $f : \mathcal{R}(v) \to \mathbb{N}$ be a function.

Let $\mathcal{R}(v)$ be the set of records for v.

To introduce the semantics of the records, we need the following notion: A *partial* solution at v is a forest of $\mathcal{G}_v \cup \partial(v)$, where for each vertex $u \in \mathcal{Y}_v$ there is a directed path from u to exactly one vertex in $(V(\partial(v)) \setminus \mathcal{Y}_v) \cup \{r\}$. Consider the set \mathcal{W} containing all partial solutions at v. The *v*-projection of a partial solution $S \in \mathcal{W}$ is a tuple (Outgoing_S, Donate_S) where:

- $(u, e) \in \text{Outgoing}_S$ if and only if there is a u-u' path in S with $u' \in V(\partial(v)) \setminus \mathcal{Y}_v$ and e is the first edge on this path which is contained in $E(\partial(v))$, and
- $(u_1, c, e) \in \text{Donate}_S$ if and only if there exists a path $u_0, u_1, u_2, \ldots, u_{i-1}, u_i$ in S with $u_0, u_i \in V(\partial(v)) \setminus \mathcal{Y}_v$ and $e = (u_{i-1}, u_i)$ and $c = \text{col}(u_1, u_2)$.

For a record $R \in \mathcal{R}(v)$ the value f(R) denotes the minimum cost of this record, i.e.,

$$f(R) = \min_{\substack{S \in \mathcal{W}, \\ R = (\texttt{Outgoing}_S, \texttt{Donate}_S)}} \sum_{e \in E(S)} \texttt{cost}(e, \textit{succ}(e))$$

We say that $\mathcal{R}(v)$ is *valid* if it contains all *v*-projections of solutions in \mathcal{W} , and in addition, for every record in $\mathcal{R}(v)$, there is a partial solution such that its *v*-projection yields this record. Observe that if $\mathcal{R}(r) = \emptyset$, then (G, r, col, cost) is a NO-instance, while if $\mathcal{R}(r) = \{(\emptyset, \emptyset, \emptyset)\}$, then R(r) is a YES-instance.

From the syntax and semantics, it follows that $|\mathcal{R}(v)| \leq 2^{\mathcal{O}(k \log k)}$ for each $v \in V(G)$.

To complete the proof, it now suffices to dynamically compute a set of valid records in a leaf-to-root fashion along T.

If v is a leaf, we create the following two records for each edge $e \in E(\partial(v))$ outgoing from v:

 $- \{\{(v, e)\}, \emptyset\},$

 $- \{\{(v, e)\}, \{(v, e, \operatorname{col}(e))\}\}.$

It follows that f(R) = 0 for each $R \in \mathcal{R}(v)$.

If v is an internal node, we start with bounding the number of children of v in order to bound the number of records, which need to be computed. Let V_{del} denote the set of children of v which do not increase the edge-cut width of v, i.e., for each $u \in V_{del}$ it holds $E(\partial(v)) = \{(u, v)\}$. We define the minimum changeover cost of V_{del} as $cost_{del} = \sum_{u \in V_{del}} \min_{R \in \mathcal{R}(v)} f(R)$. Then, we can delete T_u for each $u \in V_{del}$.

After this step there are at most 2(k-1) children left. Let u_1, \ldots, u_ℓ with $\ell \leq 2(k-1)$ denote the remaining children of v. First, we compute a local set $\overline{\mathcal{R}(v)}$, in the same way we would compute the set of records in the leaf case. Note that the number of edges incident to v is bounded by 2k - 1. Hence, $|\overline{\mathcal{R}(v)}| \leq 4k - 2$. Our goal is to compute $\mathcal{R}(v)$ using the local set $\overline{\mathcal{R}(v)}$ and the partial results $\mathcal{R}(u_1), \ldots, \mathcal{R}(u_\ell)$.

In the following we take one record each out of $\mathcal{R}(v), \mathcal{R}(u_1), \ldots, \mathcal{R}(u_\ell)$ and repeat the following process for each combination of records. We proceed similarly as in the proof of EDP (Theorem 2). First, we combine the donated paths by computing a set Donate', which contains the longest paths which can be donated by T_v . For this we look at the Donate-sets in our records from $\overline{\mathcal{R}(v)}, \mathcal{R}(u_1), \ldots, \mathcal{R}(u_\ell)$. We trace out the longest paths along edges for each vertex u in a tuple (u, c, e) in the Donate-sets of these records, which can be done in time $k^{\mathcal{O}(1)}$.

Next, we consider each pair $(u, e) \in \text{Outgoing for any of the currently considered}$ records. If $e \in E(\partial(v))$, then add (u, e) to Outgoing'. In case $e \notin E(\partial(v))$, we use the donated paths in Donate' to connect e to $e' \in E(\partial(v))$, where the sink of e' is in $V(\partial(v)) \setminus \mathcal{Y}_v$, and add (u, e') to Outgoing'.

Afterwards, we need to delete all pairs in Donate' with $u \notin V(\partial(v) \cap \mathcal{Y}_v)$ or $e \notin E(\partial(v))$. Finally, the tuple (Outgoing', Donate') is inserted as a record in $\mathcal{R}(v)$.

Note that all steps are deterministic, as for each vertex there is exactly one outgoing edge.

Let R_1, \ldots, R_ℓ be the records used to compute $R \in \mathcal{R}(v)$. The integer *cost_{conn}* denotes the sum of the changeover cost for connecting an outgoing tuple with a longest donate path. Now, we can determine the minimum cost of $R \in \mathcal{R}(v)$ by computing $f(R) = \min\{f(R), cost_{conn} + cost_{del} + \sum_{j=1}^{\ell} f(R_j)\}$, where we initiate $f(R) = \infty$.

Since the number of records and the size of each record is bounded by k, the running time of this algorithm is $2^{\mathcal{O}(k^2 \log k)} \cdot n$.

6 Conclusion

The parameter developed in this paper, edge-cut width, is aimed at mitigating the algorithmic shortcomings of tree-cut width and filling the role of an "easy-to-use" edge-based alternative to treewidth. We show that edge-cut width essentially has all the desired properties one would wish for as far as algorithmic applications are concerned: it is easy to compute, uses a natural structure as its decomposition, and yields fixed-parameter tractability for all problems that one would hope an edge-based alternative to treewidth could solve.

Last but not least, we note that a preprint exploring a different parameter that is aimed at providing an edge-based alternative to treewidth appeared shortly after the results presented in our paper were obtained [26]. While it is already clear that the two parameters are not equivalent, it would be interesting to explore the relationship between them in future work.

References

- Bentert, M., Haag, R., Hofer, C., Koana, T., Nichterlein, A.: Parameterized complexity of min-power asymmetric connectivity. Theory Comput. Syst. 64(7), 1158–1182 (2020)
- Bodlaender, H.L., Drange, P.G., Dregi, M.S., Fomin, F.V., Lokshtanov, D., Pilipczuk, M.: A c^k n 5-approximation algorithm for treewidth. SIAM J. Comput. 45(2), 317–378 (2016). https://doi.org/10.1137/130947374, https://doi.org/10.1137/130947374
- Bodlaender, H.L., Jansen, B.M.P., Kratsch, S.: Preprocessing for treewidth: A combinatorial analysis through kernelization. SIAM J. Discret. Math. 27(4), 2108–2142 (2013)
- Bonnet, É., Kim, E.J., Thomassé, S., Watrigant, R.: Twin-width I: tractable FO model checking. In: 61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020. pp. 601–612. IEEE (2020)
- Bredereck, R., Heeger, K., Knop, D., Niedermeier, R.: Parameterized complexity of stable roommates with ties and incomplete lists through the lens of graph parameters. In: Lu, P., Zhang, G. (eds.) 30th International Symposium on Algorithms and Computation, ISAAC 2019, December 8-11, 2019, Shanghai University of Finance and Economics, Shanghai, China. LIPIcs, vol. 149, pp. 44:1–44:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019)
- Cygan, M., Fomin, F.V., Kowalik, L., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., Saurabh, S.: Parameterized Algorithms. Springer (2015)
- Diestel, R.: Graph Theory, 4th Edition, Graduate texts in mathematics, vol. 173. Springer (2012)

- Downey, R.G., Fellows, M.R.: Fundamentals of Parameterized Complexity. Texts in Computer Science, Springer Verlag (2013)
- Fellows, M.R., Fomin, F.V., Lokshtanov, D., Rosamond, F., Saurabh, S., Szeider, S., Thomassen, C.: On the complexity of some colorful problems parameterized by treewidth. Information and Computation 209(2), 143–153 (2011)
- Fellows, M.R., Lokshtanov, D., Misra, N., Rosamond, F.A., Saurabh, S.: Graph layout problems parameterized by vertex cover. In: ISAAC. pp. 294–305. Lecture Notes in Computer Science, Springer (2008)
- Fleszar, K., Mnich, M., Spoerhase, J.: New algorithms for maximum disjoint paths based on tree-likeness. Math. Program. 171(1-2), 433–461 (2018)
- Ganian, R.: Improving vertex cover as a graph parameter. Discret. Math. Theor. Comput. Sci. 17(2), 77–100 (2015), http://dmtcs.episciences.org/2136
- Ganian, R., ený, P.H.: On parse trees and Myhill-Nerode-type tools for handling graphs of bounded rank-width. Discr. Appl. Math. 158(7), 851–867 (2010)
- Ganian, R., Kim, E.J., Szeider, S.: Algorithmic applications of tree-cut width. In: Italiano, G.F., Pighizzini, G., Sannella, D. (eds.) Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9235, pp. 348–360. Springer (2015), to appear in Algorithmica
- Ganian, R., Klute, F., Ordyniak, S.: On structural parameterizations of the bounded-degree vertex deletion problem. Algorithmica 83(1), 297–336 (2021)
- Ganian, R., Korchemna, V.: The complexity of bayesian network learning: Revisiting the superstructure. In: Proceedings of NeurIPS 2021, the Thirty-fifth Conference on Neural Information Processing Systems (2021), to appear
- Ganian, R., Ordyniak, S.: The complexity landscape of decompositional parameters for ILP. Artif. Intell. 257, 61–71 (2018)
- Ganian, R., Ordyniak, S.: The power of cut-based parameters for computing edge-disjoint paths. Algorithmica 83(2), 726–752 (2021)
- Ganian, R., Ordyniak, S., Ramanujan, M.S.: On structural parameterizations of the edge disjoint paths problem. Algorithmica 83(6), 1605–1637 (2021)
- Giannopoulou, A.C., Kwon, O., Raymond, J., Thilikos, D.M.: Lean tree-cut decompositions: Obstructions and algorithms. In: Niedermeier, R., Paul, C. (eds.) 36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019, March 13-16, 2019, Berlin, Germany. LIPIcs, vol. 126, pp. 32:1–32:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019)
- Golovach, P.A., Komusiewicz, C., Kratsch, D., Le, V.B.: Refined notions of parameterized enumeration kernels with applications to matching cut enumeration. J. Comput. Syst. Sci. 123, 76–102 (2022)
- Gözüpek, D., Özkan, S., Paul, C., Sau, I., Shalom, M.: Parameterized complexity of the MINCCA problem on graphs of bounded decomposability. Theor. Comput. Sci. 690, 91–103 (2017)
- 23. Gutin, G.Z., Jones, M., Wahlström, M.: The mixed chinese postman problem parameterized by pathwidth and treedepth. SIAM J. Discret. Math. **30**(4), 2177–2205 (2016)
- Kim, E.J., Oum, S., Paul, C., Sau, I., Thilikos, D.M.: An FPT 2-approximation for tree-cut decomposition. Algorithmica 80(1), 116–135 (2018)
- Korhonen, T.: Single-exponential time 2-approximation algorithm for treewidth. CoRR abs/2104.07463 (2021), https://arxiv.org/abs/2104.07463
- Magne, L., Paul, C., Sharma, A., Thilikos, D.M.: Edge-trewidth: Algorithmic and combinatorial properties. CoRR abs/2112.07524 (2021)
- Marx, D., Wollan, P.: Immersions in highly edge connected graphs. SIAM J. Discrete Math. 28(1), 503–520 (2014)

- Nederlof, J., Pilipczuk, M., Swennenhuis, C.M.F., Wegrzycki, K.: Hamiltonian cycle parameterized by treedepth in single exponential time and polynomial space. In: Adler, I., Müller, H. (eds.) Graph-Theoretic Concepts in Computer Science - 46th International Workshop, WG 2020, Leeds, UK, June 24-26, 2020, Revised Selected Papers. Lecture Notes in Computer Science, vol. 12301, pp. 27–39. Springer (2020)
- 29. Nesetril, J., de Mendez, P.O.: Sparsity Graphs, Structures, and Algorithms, Algorithms and Combinatorics, vol. 28. Springer (2012)
- Oum, S.: Approximating rank-width and clique-width quickly. ACM Transactions on Algorithms 5(1) (2008)
- Robertson, N., Seymour, P.D.: Graph minors. II. Algorithmic aspects of tree-width. J. Algorithms 7(3), 309–322 (1986)
- Samer, M., Szeider, S.: Constraint satisfaction with bounded treewidth revisited. J. of Computer and System Sciences 76(2), 103–114 (2010)
- Wollan, P.: The structure of graphs not admitting a fixed immersion. J. Comb. Theory, Ser. B 110, 47–66 (2015)