On Structural Parameterizations of the Edge Disjoint Paths Problem*

Robert Ganian[1], Sebastian Ordyniak[2], and Ramanujan Sridharan[3]

1 **Algorithms and Complexity Group, TU Wien, Vienna, Austria**
`ganian@ac.tuwien.ac.at`
2 **Algorithms and Complexity Group, TU Wien, Vienna, Austria**
`ordyniak@ac.tuwien.ac.at`
3 **Algorithms and Complexity Group, TU Wien, Vienna, Austria**
`ramanujan@ac.tuwien.ac.at`

Abstract

In this paper we revisit the classical Edge Disjoint Paths (EDP) problem, where one is given an undirected graph G and a set of terminal pairs P and asks whether G contains a set of pairwise edge-disjoint paths connecting every terminal pair in P. Our focus lies on structural parameterizations for the problem that allow for efficient (polynomial-time or fpt) algorithms. As our first result, we answer an open question stated in Fleszar, Mnich, and Spoerhase (2016), by showing that the problem can be solved in polynomial time if the input graph has a feedback vertex set of size one. We also show that EDP parameterized by the treewidth and the maximum degree of the input graph is fixed-parameter tractable.

Having developed two novel algorithms for EDP using structural restrictions on the input graph, we then turn our attention towards the augmented graph, i.e., the graph obtained from the input graph after adding one edge between every terminal pair. In constrast to the input graph, where EDP is known to remain NP-hard even for treewidth two, a result by Zhou et al. (2000) shows that EDP can be solved in non-uniform polynomial time if the augmented graph has constant treewidth; we note that the possible improvement of this result to an fpt-algorithm has remained open since then. We show that this is highly unlikely by establishing the W[1]-hardness of the problem parameterized by the treewidth (and even feedback vertex set) of the augmented graph. Finally, we develop an fpt-algorithm for EDP by exploiting a novel structural parameter of the augmented graph.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity, G.2.1 Combinatorics

Keywords and phrases edge disjoint path problem, feedback vertex set, treewidth, fracture number, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.36

1 Introduction

The Edge Disjoint Paths (EDP) and Node Disjoint Paths (NDP) are fundamental routing graph problems. In the EDP (NDP) problem the input is a graph G, and a set P containing k pairs of vertices and the objective is to decide whether there is a set of k pairwise edge disjoint (respectively vertex disjoint) paths connecting each pair in P. These problems

* The authors acknowledge support by the Austrian Science Fund (FWF, project P26696). Robert Ganian is also affiliated with FI MU, Czech Republic.

and their optimization versions – MaxEDP and MaxNDP – have been at the center of numerous results in structural graph theory, approximation algorithms, and parameterized algorithms [20, 15, 4, 17, 9, 22, 19, 12, 10].

When k is a part of the input, both EDP and NDP are known to be NP-complete [14]. Robertson and Seymour's seminal work in the Graph Minors project [20] provides an $\mathcal{O}(n^3)$ time algorithm for both problems for every fixed value of k. In the realm of *Parameterized Complexity*, their result can be interpreted as *fpt-algorithms* for EDP and NDP parameterized by k. Here, one considers problems associated with a certain numerical parameter k and the central question is whether the problem can be solved in time $f(k) \cdot n^{\mathcal{O}(1)}$ where f is a computable function and n the input size; algorithms with running time of this form are called fpt-algorithms [11, 7, 5].

While the aforementioned research considered the number of paths to be the parameter, another line of research investigates the effect of *structural parameters* of the input graphs on the complexity of these problems. Fleszar, Mnich, and Spoerhase [10] initiated the study of NDP and EDP parameterized by the feedback vertex set number (the size of the smallest feedback vertex set) of the input graph and showed that EDP remains NP-hard even on graphs with feedback vertex set number two. Since EDP is known to be polynomial time solvable on forests [12], this left only the case of feedback vertex set number one open, which they conjectured to be polynomial time solvable. Our first result is a positive resolution of their conjecture.

▶ **Theorem 1.** EDP *can be solved in time* $\mathcal{O}(|P||V(G)|^{\frac{5}{2}})$ *on graphs with feedback vertex set number one.*

A key observation behind the polynomial-time algorithm is that an EDP instance with a feedback vertex set $\{x\}$ is a Yes-instance if and only if, for every tree T of $G - \{x\}$, it is possible to connect all terminal pairs in T either to each other or to x through pairwise edge disjoint paths in T. The main ingredient of the algorithm is then a dynamic programming procedure that determines whether such a set exists for a tree T of $G - \{x\}$.

Continuing to explore structural parameterizations for the input graph of an EDP instance, we then show that even though EDP is NP-complete when the *input graph* has treewidth two, it becomes fixed-parameter tractable if we additionally parameterize by the maximum degree.

▶ **Theorem 2.** EDP *is fixed-parameter tractable parameterized by the treewidth and the maximum degree of the input graph.*

Having explored the algorithmic applications of structural restrictions on the input graph for EDP, we then turn our attention towards similar restrictions on the *augmented graph* of an EDP instance (G, P), i.e., the graph obtained from G after adding an edge between every pair of terminals in P. Whereas EDP is NP-complete even if the input graph has treewidth at most two [19], it can be solved in non-uniform polynomial time if the treewidth of the augmented graph is bounded [22]. It has remained open whether EDP is fixed-parameter tractable parameterized by the treewidth of the augmented graph; interestingly, this has turned out to be the case for the strongly related multicut problems [13]. Surprisingly, we show that this is not the case for EDP, by establishing the W[1]-hardness of the problem parameterized by not only the treewidth but also by the feedback vertex set number of the augmented graph.

▶ **Theorem 3.** *EDP is* W[1]*-hard parameterized by the feedback vertex set number of the augmented graph.*

Motivated by this strong negative result, our next aim was to find natural structural parameterizations for the augmented graph of an EDP instance for which the problem becomes fixed-parameter tractable. Towards this aim, we introduce the *fracture number*, which informally corresponds to the size of a minimum vertex set S such that the size of every component in the graph minus S is small (has size at most $|S|$). We show that EDP is fixed-parameter tractable parameterized by this new parameter.

▶ **Theorem 4.** EDP *is fixed-parameter tractable parameterized by the fracture number of the augmented graph.*

We note that the reduction in [10, Theorem 6] excludes the applicability of the fracture number of the *input graph* by showing that EDP is NP-complete even for instances with fracture number at most three. Finally, we complement Theorem 4 by showing that bounding the number of terminal pairs in each component instead of the its size is not sufficient to obtain fixed-parameter tractability.

2 Preliminaries

2.1 Basic Notation

We use standard terminology for graph theory, see for instance [6]. Given a graph G, we let $V(G)$ denote its vertex set, $E(G)$ its edge set and by $V(E')$ the set of vertices incident with the edges in E', where $E' \subseteq E(G)$. 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 $N_G(X)$. For a vertex set A, we use $G - A$ to denote the graph obtained from G by deleting all vertices in A, 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 a vertex set X is a *feedback vertex set* (*FVS*) if $G - X$ is a forest. We use $[i]$ to denote the set $\{0, 1, \ldots, i\}$. The *feedback vertex set number* of a graph G, denoted by $\mathbf{fvs}(G)$, is the smallest integer k such that G has a feedback vertex set of size k.

2.2 Edge Disjoint Path Problem

In the Edge Disjoint Paths (EDP) problem, one is given an undirected graph G a set P of terminal pairs (i.e., subsets of $V(G)$ of size two) and the question is whether there is there a set of pairwise edge disjoint paths connecting every set of terminal pairs in P. Let (G, P) be an instance of EDP; for brevity, we will sometimes denote a terminal pair $\{s, t\} \in P$ simply as st. For a subgraph H of G, we denote by $P(H)$ the subset of P containing all sets that have a non-empty intersection with $V(H)$ and for $P' \subseteq P$, we denote by $\widetilde{P'}$ the set $\bigcup_{p \in P'} p$. We will assume that, w.l.o.g., each vertex $v \in V(G)$ occurs in at most one terminal pair, each vertex in a terminal pair has degree 1 in G, and each terminal pair is not adjacent to each other; indeed, for any instance without these properties, we can add a new leaf vertex for terminal, attach it to the original terminal, and replace the original terminal with the leaf vertex [22].

▶ **Definition 5** ([22]). The *augmented graph* of (G, P) is the graph G^P obtained from G by adding edges between each terminal pair, i.e., $G^P = (V(G), E(G) \cup P)$.

2.3 Parameterized Complexity

The parameterized complexity paradigm allows a finer analysis of the complexity of problems by associating each problem instance L with a numerical parameter k; the pair (L, k) is then an instance of a parameterized problem. A parameterized problem is *fixed-parameter tractable* (FPT in short) if a given instance (L, k) can be solved in time $\mathcal{O}(f(k) \cdot |L|^{\mathcal{O}(1)}$ where f is an arbitrary computable function; we call algorithms running in this time *fpt-algorithms.* The complexity class W[1] is the analog of NP for parameterized complexity; under established complexity assumptions, problems that are hard for W[1] do not admit fpt-algorithms.

We refer the reader to the respective monographs [11, 7, 5] for an in-depth introduction to parameterized complexity.

2.4 Treewidth

A *tree-decomposition* of a graph $G = (V, E)$ is a pair $(T, \{B_t : t \in V(T)\})$ where $B_t \subseteq V$ for every $t \in V(T)$ and T is a tree such that:

1. for each edge $(u, v) \in E$, there is a $t \in V(T)$ such that $\{u, v\} \subseteq B_t$, and
2. for each vertex $v \in V$, $T[\{\, t \in V(T) \mid v \in B_t \,\}]$ is a non-empty (connected) tree.

The *width* of a tree-decomposition is $\max_{t \in V(T)} |B_t| - 1$. The *treewidth* [16] of G is the minimum width taken over all tree-decompositions of G and it is denoted by $\mathbf{tw}(G)$. We call the elements of $V(T)$ *nodes* and B_t *bags.*

While it is possible to compute the treewidth exactly using an fpt-algorithm [1], the asymptotically best running time is achieved by using the recent state-of-the-art 5-approximation algorithm of Bodlaender et al. [2].

▶ **Fact 6** ([2]). *There exists an algorithm which, given an n-vertex graph G and an integer k, in time $2^{\mathcal{O}(k)} \cdot n$ either outputs a tree-decomposition of G of width at most $5k + 4$ and $\mathcal{O}(n)$ nodes, or correctly determines that $\mathbf{tw}(G) > k$.*

A tree-decomposition $(T, B_t : t \in V(T))$ of a graph G is *nice* if the following conditions hold: (1) T is rooted at a node r such that $|B_r| = \emptyset$, (2) every node of T has at most two children, if a node t of T has two children t_1 and t_2, then $B_t = B_{t_1} = B_{t_2}$; in that case we call t a *join node*, (3) if a node t of T has exactly one child t', then exactly one of the following holds: (3A) $B_t = B_{t'} \cup \{v\}$, in which case we call t an *introduce node* or (3B) $B_t = B_{t'} \setminus \{v\}$ in which case we call t a *forget node*, and (4) if a node t of T is a leaf, then $|B_t| = 1$; we call these *leaf nodes.*

The main advantage of nice tree-decompositions is that they allow the design of much more transparent dynamic programming algorithms, since one only needs to deal with four specific types of nodes. It is well known (and easy to see) that for every fixed k, given a tree-decomposition of a graph $G = (V, E)$ of width at most k and with $\mathcal{O}(|V|)$ nodes, one can construct in linear time a nice tree-decomposition of G with $\mathcal{O}(|V|)$ nodes and width at most k [3]. Given a node t in T, we let Y_t be the set of all vertices contained in the bags of the subtree rooted at t, i.e., $Y_t = B_t \cup \bigcup_{p \text{ is separated from the root by } t} B_p$.

3 Closing the Gap on Graphs of Feedback Vertex Number One

In this section we develop a polynomial-time algorithm for EDP restricted to graphs with feedback vertex set number one. We refer to this particular variant as Simple Edge Disjoint Paths (SEDP): given an EDP instance (G, P) and a FVS $X = \{x\}$, solve (G, P).

Additionally to our standard assumptions about EDP (given in Subsection 2.2), we will assume that: (1) every neighbor of x in G is a leaf in $G - X$, (2) x is not a terminal, i.e., $x \notin \widetilde{P}$, and (3) every tree T in $G - X$ is rooted in a vertex r that is not a terminal. Property

(1) can be ensured by an additional leaf vertex l to any non-leaf neighbor n of x, removing the edge $\{n, x\}$ and adding the edge $\{l, x\}$ to G. Property (2) can be ensured by adding an additional leaf vertex l to x and replacing x with l in P and finally (3) can be ensured by adding a leaf vertex l to any non-terminal vertex r in T and replacing r with l in P.

A key observation behind our algorithm for SEDP is that whether or not an instance $\mathcal{I} = (G, P, X)$ has a solution merely depends on the existence of certain sets of pairwise edge disjoint paths in the trees T in $G - X$. In particular, as we will show in Lemma 8 later on, $\mathcal{I}$ has a solution if and only if every tree T in $G - X$ is $\emptyset$-connected (see Definition 7). The main ingredient of the algorithm is then a bottom-up dynamic programming algorithm that determines whether a tree T in $G-X$ is $\emptyset$-connected. We now define the various connectivity states of subtrees of T that we need to keep track of in the dynamic programming table.

▶ **Definition 7.** Let T be a tree in $G - X$ rooted at r (recall that we can assume that r is not in $\widetilde{P}$), $t \in V(T)$, and let S be a set of pairwise edge disjoint paths in $G[T_t \cup X]$ and $P' \subseteq P(T_t)$, where T_t is the subtree of T rooted at t.

We say that the set S *$\gamma_\emptyset$-connects* P' in $G[T_t \cup X]$ if for every $a \in \widetilde{P'} \cap T_t$, the set S either contains an a-x path disjoint from b, or it contains an a-b path disjoint from x, where $\{a, b\} \in P'$. Moreover, for $\ell \in \{\gamma_X\} \cup P(T_t)$, we say that the set S *ℓ-connects* T_t if S $\gamma_\emptyset$-connects $P(T_t) \setminus \{\ell\}$ and additionally the following conditions hold.

- If $\ell = \gamma_X$ then S also contains a path from t to x.
- If $\ell = p$ for some $p \in P(T_t)$ then:
  - If $p \cap T_t = \{a\}$ then S contains a t-a path disjoint from x.
  - If $p \cap T_t = \{a, b\}$ then S contains a t-a path disjoint from x and a b-x path disjoint from a or S contains a t-b path disjoint from x and an a-x path disjoint from b.

For $\ell \in \{\gamma_\emptyset, \gamma_X\} \cup P(T_t)$, we say that T_t is ℓ-connected if there is a set S which ℓ-connects $P(T_t)$ in $G[T_t \cup X]$.

Informally, a tree T_t is: (1) $\gamma_\emptyset$-connected if all its terminal pairs can be connected in $G[T_t \cup X]$ either to themselves or to x, (2) γ_X-connected if it is $\gamma_\emptyset$-connected and additionally there is a path from its root to x (which can later be used to connect some terminal not in T_t to x via the root of T_t), and (3) γ_p-connected if all but one of its terminals, i.e., one of the terminals in p, can be connected in $G[T_t \cup X]$ either to themselves or to x, and additionally one terminal in p can be connected to the root of T_t (from which it can later be connected to x or the other terminal in p).

▶ **Lemma 8.** *(G, X, P) has a solution if and only if every tree T in $G - X$ is $\gamma_\emptyset$-connected.*

Due to Lemma 8, our algorithm to solve EDP only has to determine whether every tree in $G - X$ is $\gamma_\emptyset$-connected. For a tree T in $G - X$, our algorithm achieves this by computing a set of labels $L(t)$, where $L(t)$ is the set of all labels $\ell \in \{\gamma_\emptyset, \gamma_X\} \cup P(T_t)$ such that T_t is ℓ-connected, via a bottom-up dynamic programming procedure. We begin by arguing that for a leaf vertex l, the value $L(l)$ can be computed in constant time.

▶ **Lemma 9.** *The set $L(l)$ for a leaf vertex l of T can be computed in time $\mathcal{O}(1)$.*

Proof. Since l is a leaf vertex, we conclude that T_l is $\gamma_\emptyset$-connected if and only if either $l \notin \widetilde{P}$ or $l \in \widetilde{P}$ and $(l, x) \in E(G)$. Similarly, T_l is γ_X-connected if and only if $l \notin \widetilde{P}$ and $(l, x) \in E(G)$. Finally, T_l is ℓ-connected for some $\ell \in P(T_l)$ if and only if $l \in \widetilde{P}$. Since all these properties can be checked in constant time, the statement of the lemma follows. ◀

We will next show how to compute $L(t)$ for a non-leaf vertex $t \in V(T)$ with children $t_1, \ldots, t_l$.

▶ **Definition 10.** We define the three sets $V_t^{\neg\gamma_\emptyset} = \{\, t_i \mid \gamma_\emptyset \notin L(t_i) \,\}$, $V_t^{\gamma_X} = \{\, t_i \mid \gamma_X \in L(t_i) \,\}$, and $V_t = \{t_1, \dots, t_l\} \setminus (V_t^{\neg\gamma_\emptyset} \cup V_t^{\gamma_X})$.

That is, $V_t^{\neg\gamma_\emptyset}$ is the set of those children t_i such that T_i is *not* $\gamma_\emptyset$-connected, $V_t^{\gamma_X}$ is the set of those children t_i such that T_i is γ_X-connected and V_t is the set comprising the remaining children. Observe that $\{V_t, V_t^{\neg\gamma_\emptyset}, V_t^{\gamma_X}\}$ forms a partition of $\{t_1, \dots, t_l\}$ and moreover $\gamma_\emptyset \in L(t)$ and $\gamma_X \notin L(t)$ for every $t \in V_t$. Let $H(t)$ be the graph with vertex set $V_t \cup V_t^{\neg\gamma_\emptyset}$ having an edge between t_i and t_j (for $i \neq j$) if and only if $L(t_i) \cap L(t_j) \neq \emptyset$ and not both t_i and t_j are in V_t. The following lemma is crucial to our algorithm, because it provides us with a simple characterization of $L(t)$ for a non-leaf vertex $t \in V(T)$.

▶ **Lemma 11.** *Let t be a non-leaf vertex of T with children $t_1, \dots, t_l$. Then T_t is:*

- *$\gamma_\emptyset$-connected if and only if $L(t') \neq \emptyset$ for every $t' \in \{t_1, \dots, t_l\}$ and $H(t)$ has a matching M such that $|V^{\neg\gamma_\emptyset} \setminus V(M)| \leq |V_t^{\gamma_X}|$,*
- *γ_X-connected if and only if $L(t') \neq \emptyset$ for every $t' \in \{t_1, \dots, t_l\}$ and $H(t)$ has a matching M such that $|V^{\neg\gamma_\emptyset} \setminus V(M)| < |V_t^{\gamma_X}|$,*
- *ℓ-connected (for $\ell \in P(T_t)$) if and only if $L(t') \neq \emptyset$ for every $t' \in \{t_1, \dots, t_l\}$ and there is a t_i with $\ell \in L(t_i)$ such that $H(t) - \{t_i\}$ has a matching M with $|V^{\neg\gamma_\emptyset} \setminus V(M)| \leq |V_t^{\gamma_X}|$.*

The following two lemmas show how the above characterization can be employed to compute $L(t)$ for a non-leaf vertex t of T. Since the matching employed in Lemma 11 needs to maximize the number of vertices covered in $V^{\neg\gamma_\emptyset}$, we first show how such a matching can be computed efficiently.

▶ **Lemma 12.** *There is an algorithm that, given a graph G and a subset S of $V(G)$, computes a matching M maximizing $|V(M) \cap S|$ in time $\mathcal{O}(\sqrt{|V|}|E|)$.*

▶ **Lemma 13.** *Let t be a non-leaf vertex of T with children $t_1, \dots, t_l$. Then $L(t)$ can be computed from $L(t_1), \dots, L(t_l)$ in time $\mathcal{O}(|P(T_t)| l^2 \sqrt{l})$.*

We are now ready to put everything together to decide whether a tree T is $\gamma_\emptyset$-connected.

▶ **Lemma 14.** *Let T be a tree in $G - X$. There is an algorithm that decides whether T is $\gamma_\emptyset$-connected in time $\mathcal{O}(|P(T)||V(T)|^{\frac{5}{2}})$.*

Proof. The algorithm computes the set of labels $L(t)$ for every vertex $t \in V(T)$ using a bottom-up dynamic programming approach. Starting from the leaves of T, for which the set of labels can be computed due to Lemma 9 in constant time, it uses Lemma 13 to compute $L(t)$ for every inner node t of T in time $\mathcal{O}(|P(T_t)| l^2 \sqrt{l})$. The total running time of the algorithm is then the sum of the running time for any inner node of T plus the number of leaves of T, i.e., $\mathcal{O}(|P(T)||V(T)|^{\frac{5}{2}})$. ◀

▶ **Theorem 1.** EDP *can be solved in time $\mathcal{O}(|P||V(G)|^{\frac{5}{2}})$ on graphs with feedback vertex set number one.*

Proof. We first employ Lemma 14 to determine whether every tree T of $G - X$ is $\gamma_\emptyset$-connected. If so we output YES and otherwise NO. Correctness follows from Lemma 8. ◀

4 Treewidth and Maximum Degree

The goal of this section is to obtain an fpt-algorithm for EDP parameterized by the treewidth ω and maximum degree Δ of the input graph.

▶ **Theorem 15.** EDP *can be solved in time* $2^{\mathcal{O}(\Delta\omega^2)} \cdot n$*, where* ω*,* Δ *and* n *are the treewidth, maximum degree and number of vertices of the input graph* G*, respectively.*

Proof Sketch. Let (G, P) be an instance of EDP and let $(T, \mathcal{B})$ be a nice tree-decomposition of G of width at most $k = 5\omega + 4$; recall that such $(T, \mathcal{B})$ can be computed in time $2^{\mathcal{O}(k)}$ by Fact 6. Consider the following leaf-to-root dynamic programming algorithm Å, executed on T. At each bag B_t associated with a node t of T, Å will compute a table $\mathcal{M}_t$ of *records*, which are tuples of the form $\{(\mathsf{used}, \mathsf{give}, \mathsf{single})\}$ where:

- used is a multiset of subsets of B_t of cardinality 2 with each subset occurring at most Δ times,
- give is a mapping from subsets of B_t of cardinality 2 to $[\Delta]$, and
- single is a mapping which maps each terminal $a_i \in Y_t$ such that its counterpart $b_i \notin Y_t$ to an element of B_t.

Before we proceed to describe the steps of the algorithm itself, let us first introduce the semantics of a record. For a fixed t, we will consider the graph G_t obtained from $G[Y_t]$ by removing all edges with both endpoints in B_t (we note that this "pruned" definition of G_t is not strictly necessary for the algorithm, but makes certain steps easier later on). Then $\alpha = \{(\mathsf{used}, \mathsf{give}, \mathsf{single})\} \in \mathcal{M}_t$ if and only if there exists a set of edge disjoint paths Q in G_t and a surjective mapping τ from terminal pairs occurring in Y_t to subsets of B_t of size two with the following properties:

- For each terminal pair ab that occurs in Y_t:
  - Q either contains a path whose endpoints are a and b, or
  - Q contains an a-x_1 path for some $x_1 \in B_t$ and a b-x_2 path for some $x_2 \in B_t$ which is distinct from x_1, and furthermore $\tau(ab) = \{x_1, x_2\} \in$used;
- for each terminal pair ab such that $a \in Y_t$ but $b \notin Y_t$:
  - Q contains a path whose endpoints are a and $x \in B_t$, where $(a, x) \in \mathsf{single}$;
- for each distinct $x_1, x_2 \in B_t$, Q contains precisely $\mathsf{give}(\{x_1, x_2\})$ paths from x_1 to x_2.

In the above case, we say that Q witnesses α. It is important to note that the equivalence between the existence of records and sets Q of pairwise edge disjoint paths only holds because of the bound on the maximum degree. That is because every vertex of G has degree at most Δ, it follows that any set Q of pairwise edge disjoint paths can contain at most Δ paths containing a vertex in the boundary. Moreover, we note that by reversing the above considerations, given a set of edge disjoint paths Q in G_t satisfying a certain set of conditions, we can construct in time $3^{\Delta k}$ a set of records in $\mathcal{M}_t$ that are witnessed by Q (one merely needs to branch over all options of assigning paths in α which end in the boundary: they may either contribute to give or to single or to used). These conditions are that each path either (i) connects a terminal pair, (ii) connects a terminal pair to two vertices in B_t, (iii) connects two vertices in B_t, or (iv) connects a terminal $a \in Y_t$ whose counterpart $b \notin Y_t$ to a vertex in B_t.

Å runs as follows: it begins by computing the records $\mathcal{M}_t$ for each leaf t of T. It then proceeds to compute the records for all remaining nodes in T in a bottom-up fashion, until it computes $\mathcal{M}_r$. Since $B_r = \emptyset$, it follows that (G, P) is a YES-instance if and only if $(\emptyset, \emptyset, \emptyset) \in \mathcal{M}_r$. For each record α, it will keep (for convenience) a set Q_α of edge disjoint paths witnessing α. Observe that while for each specific α there may exist many possible choices of Q_α, all of these interact with B_t in the same way.

We make one last digression before giving the procedures used to compute $\mathcal{M}_t$ for the four types of nodes in nice tree-decompositions. First, observe that the size of one particular record is at most $\Delta k^2 + \Delta k^2 + |\mathsf{single}|$. Since the number of edge disjoint paths in G_t ending in B_t is upper-bounded by Δk, it follows that each record in $\mathcal{M}_t$ satisfies $|\mathsf{single}| \leq \Delta k$ and in particular each such record has size at most $\mathcal{O}(\Delta k^2)$. As a consequence, $|\mathcal{M}_t| \leq 2^{\mathcal{O}(\Delta k^2)}$ for each node t in T, which is crucial to obtain an upper-bound on the running time of Å. It remains to formalize the computation of the records for the four types of nodes of a nice tree decomposition.

Summary and running time

Algorithm Å begins by invoking Fact 6 to compute a tree-decomposition of width at most $k = 5\omega + 4$ and $\mathcal{O}(n)$ nodes, and then converts this into a nice tree-decomposition $(T, \mathcal{B})$ of the same width and also $\mathcal{O}(n)$ nodes. It then proceeds to compute the records $\mathcal{M}_t$ (along with corresponding witnesses) for each node t of T in a leaves-to-root fashion, using the procedures described above. The number of times any procedure is called is upper-bounded by $\mathcal{O}(n)$, and the running time of every procedure is upper-bounded by the worst-case running time of the procedure for forget nodes. There, for each record β in the data table of the child of t, the algorithm takes its witness Q_β and uses branching to construct at most $k^{\mathcal{O}(\Delta k)}$ new witnesses (after the necessary conditions are checked). Each such witness Q_α gives rise to a set of records that can be computed in time $3^{\Delta k}$, which are then added to $\mathcal{M}_t$ (unless they are already there). All in all, the running time of this procedure is upper-bounded by $2^{\mathcal{O}(\Delta k^2)} \cdot k^{\mathcal{O}(\Delta k)} \cdot 3^{\Delta k} = 2^{\mathcal{O}(\Delta k^2)}$, and the run-time of the algorithm follows. ◀

5 Lower Bounds of EDP for Parameters of the Augmented Graph

This section is devoted to a proof of the following theorem.

▶ **Theorem 3.** *EDP is* W[1]*-hard parameterized by the feedback vertex set number of the augmented graph.*

Note that since the feedback vertex set number upper-bounds the treewidth, Theorem 3 complements the result in [22] showing that EDP is solvable in polynomial time for bounded treewidth.

As intermediate steps for the proof of Theorem 3 we establish the W[1]-hardness of several interesting variants of a multidimensional version of the well-known Subset Sum problem as well as several directed and undirected versions of EDP, which we believe are interesting in their own right. Namely, the proof starts by showing W[1]-hardness for the following problem using a reduction from the well-known Multi-Colored Clique (MCC) problem [7].

Multidimensional Subset Sum (MSS)	
Input:	An integer k, a set $S = \{s_1, \dots, s_n\}$ of item-vectors with $s_i \in \mathbb{N}^k$ for every i with $1 \leq i \leq n$ and a target vector $t \in \mathbb{N}^k$.
Parameter:	k
Question:	Is there a subset $S' \subseteq S$ such that $\sum_{s \in S'} s = t$?

Using a reduction from MSS, the proof then continues by establishing W[1]-hardness for the following more relaxed version of MSS.

Multidimensional Relaxed Subset Sum (MRSS)	
Input:	An integer k, a set $S = \{s_1, \dots, s_n\}$ of item-vectors with $s_i \in \mathbb{N}^k$ for every i with $1 \leq i \leq n$, a target vector $t \in \mathbb{N}^k$, and an integer k'.
Parameter:	$k + k'$
Question:	Is there a subset $S' \subseteq S$ with $\lvert S'\rvert \leq k'$ such that $\sum_{s \in S'} s \geq t$?

Next, we reduce from MRSS to the following directed variant of EDP.

Multiple Directed Edge Disjoint Paths (MDEDP)	
Input:	A directed graph G, a set P of ℓ triples (s_i, t_i, n_i) with $1 \leq i \leq \ell$, $s_i, t_i \in V(G)$, and $n_i \in \mathbb{N}$.
Parameter:	$\mathbf{fvs}(\overline{G}) + \lvert P\rvert$
Question:	Is there a set of pairwise edge disjoint paths containing n_i paths from s_i to t_i for every i with $1 \leq i \leq \ell$?

In the above, $\overline{G}$ denotes the underlying undirected graph of a given directed graph G.

Using a known result that allows to reduce the directed EDP problem to the undirected EDP problem given by Vygen [21], we then show that the undirected variant of the above problem, which we refer to as MUEDP, of MDEDP is also W[1]-hard. The final step of the proof is then a straightforward reduction from MUEDP to the EDP problem parameterized by the feedback vertex set number of the augmented graph.

6 An fpt-Algorithm for EDP using the Augmented Graph

In light of Theorem 3, it is natural to ask whether there exist natural structural parameters of the augmented graph which would give rise to fpt-algorithms for EDP but which cannot be used on the input graph. In other words, does considering the augmented graph instead of the input graph provide any sort of advantage in terms of fpt-algorithms? In this section we answer this question affirmatively by showing that EDP is fixed-parameter tractable parameterized by the so-called *fracture number* of the augmented graph. We note that a parameter similar to the fracture number has recently been used to obtain fpt-algorithms for Integer Linear Programming [8].

▶ **Definition 16.** A vertex subset X of a graph H is called a *fracture modulator* if each connected component in $H \setminus X$ contains at most $\lvert X\rvert$ vertices. We denote the size of a minimum-cardinality fracture modulator in H as $\text{frac}(H)$ or the *fracture number* of H.

We begin by making a simple structural observation about fracture modulators.

▶ **Lemma 17.** *Let (G, P) be an instance of* EDP *and let k be the fracture number of its augmented graph. Then there exists a fracture modulator X of G^P of size at most $2k$ such that X does not contain any terminal vertices. Furthermore, such a fracture modulator X can be constructed from any fracture modulator of size at most k in linear time.*

We note that the problem of computing a fracture modulator of size at most k has been recently considered in the context of Integer Linear Programming [8].

▶ **Lemma 18** ([8, Theorems 7 and 8])**.** *There exists an algorithm which takes as input a graph G and an integer k, runs in time at most $\mathcal{O}((k+1)^k \lvert E(G)\rvert)$, and outputs a fracture modulator of cardinality at most k if such exists. Moreover, there is a polynomial-time algorithm that either computes a fracture modulator of size at most $(k+1)k$ or outputs correctly that no fracture modulator of size at most k exists.*

For the rest of this section, let us fix an instance (G, P) of EDP with a fracture modulator X of G^P of cardinality k which does not contain any terminals. Furthermore, since the subdivision of any edge (i.e., replacing an edge by a path of length 2) in (G, P) does not change the validity of the instance, we will assume without loss of generality that $G[X]$ is edgeless; in particular, any edges that may have had both endpoints in X will be subdivided, creating a new connected component of size 1.

Our next step is the definition of *configurations*. These capture one specific way a connected component C of $G^P - X$ may interact with the rest of the instance. It will be useful to observe that for each terminal pair ab there exists precisely one connected component C of $G^P - X$ which contains both of its terminals; we say that *ab occurs* in C. For a connected component C, we let C^+ denote the induced subgraph on $G^P[C \cup X]$.

A *trace* is a tuple $(x_1, \ldots, x_\ell)$ of elements of X. A configuration is a tuple (α, β) where

- α is a multiset of at most k traces, and
- β is a mapping from subsets of X of cardinality 2 to $[k^2]$.

A component C of G^P *admits* a configuration (α, β) if there exists a set of edge disjoint paths $\mathcal{F}$ in C^+ and a surjective mapping τ (called the *assignment*) from α to the terminal pairs that occur in C with the following properties.

- For each terminal pair st that occurs in C:
  - $\mathcal{F}$ either contains a path whose endpoints are s and t, or
  - $\mathcal{F}$ contains an s-x_1 path for some $x_1 \in X$ and a t-x_2 path for some distinct $x_2 \in X$ and there exists a trace $L = (x_1, \ldots, x_2) \in \alpha$ such that $\tau(L) = st$.
- for each distinct $a, b \in X$, $\mathcal{F}$ contains precisely $\beta(\{a, b\})$ paths from a to b.
- $\mathcal{F}$ contains no other paths than the above.

Intuitively, α stores information about how one particular set of edge disjoint paths A which originate in C is routed through the instance: they may either be routed only through C^+ (in which case they don't contribute to α), or they may leave C^+ (in which case α stores the order in which these paths visit vertices of X, i.e., their trace). On the other hand, β stores information about how paths that originate outside of C can potentially be routed through C (in a way which does not interfere with A). Observe that for any particular α there may exist several distinct configurations ((α, β_1), (α, β_2) and so forth); similarly, for any particular β there may exist several distinct configurations ((α_1, β), (α_2, β) and so forth).

If a set $\mathcal{F}$ of edge disjoint paths in C^+ satisfies the conditions specified above for a configuration (α, β), we say that $\mathcal{F}$ *gives rise* to (α, β). Clearly, given $\mathcal{F}$ and (α, β), it is possible to determine whether $\mathcal{F}$ gives rise to (α, β) in time polynomial in $|V(C)|$.

While configurations capture information about how a component can interact with a set of edge disjoint paths, our end goal is to have a way of capturing all important information about a component irrespective of any particular selection of edge disjoint paths. To this end, we introduce the notion of *signatures*. A signature of a component C, denoted $\text{sign}(C)$, is the set of all configurations which C admits. The set of all configurations is denoted by Λ.

▶ **Lemma 19.** *Given a component C, it is possible to compute $sign(C)$ in time at most $k^{\mathcal{O}(k^2)}$. Furthermore, $|sign(C)| \leq |\Lambda| \leq k^{\mathcal{O}(k^2)}$.*

Our next step is the formulation of a clear condition linking configurations of components in $G^P - X$ and solving (G, P). This condition will be of importance later, since it will be checkable by an integer linear program. For a trace α, we say that a, b *occur consecutively* in α if elements a and b occur consecutively in the sequence α (regardless of their order), i.e., $\alpha = (\dots, a, b, \dots)$ or $\alpha = (\dots, b, a, \dots)$. Let $\mathcal{D}$ be the set of connected components of $G^P - X$.

A *configuration selector* is a function which maps each connected component C in $G^P - X$ to a configuration $(\alpha, \beta) \in \mathrm{sign}(C)$. We say that a configuration selector S is *valid* if it satisfies the condition that $\mathrm{dem}(ab) \leq \mathrm{sup}(ab)$ for every $\{a, b\} \subseteq X$, where dem (*demand*) and sup (*supply*) are defined as follows: $\mathrm{dem}(ab)$ is the number of traces in $\bigcup_{C \in \mathcal{D}} S(C)$ where ab occur consecutively and $\mathrm{sup}(ab)$ is the sum of all the values $\beta(a, b)$ in $\bigcup_{C \in \mathcal{D}} S(C)$.

The next, crucial lemma links the existence of a valid configuration selector to the existence of a solution for EDP.

▶ **Lemma 20.** (G, P) *is a* Yes*-instance if and only if there is a valid configuration selector.*

The problem whether there is a valid configuration selector can be easily translated into an integer linear program with a number of variables that can be bounded in terms of k. It then follows from [18] that the problem is fixed-parameter tractable parameterized by k.

▶ **Lemma 21.** *There exists an algorithm which takes as input an* EDP *instance* (G, P) *and a fracture modulator* X *of* G^P *and determines whether there exists a valid configuration selector* S *in time at most* $2^{2^{k^{\mathcal{O}(k^2)}}} \cdot |V(G)|$.

▶ **Theorem 4.** EDP *is fixed-parameter tractable parameterized by the fracture number of the augmented graph.*

Proof. We begin by computing a fracture modulator of the augmented graph by Lemma 18. We then use Lemma 21 to determine whether a valid configuration selector S exists, which by Lemma 20 allows us to solve EDP. ◀

Having established that EDP is fixed-parameter tractable parameterized by the fracture number, let us briefly consider potential extensions of the parameter. In particular, one might be tempted to think that tractability still applies if instead of bounding the size of each component one only bounds the number of terminal pairs in each component. We conclude this section by showing that this is not the case: even if both the deletion set and the number of terminal pairs in each component are bounded by a constant, EDP remains NP-complete.

▶ **Theorem 22.** EDP *is NP-complete even if the augmented graph* G^P *of the instance has a deletion set* D *of size* 6 *such that each component of* $G^P - D$ *contains at most* 1 *terminal pair.*

The proof of Theorem 22 is based on a polynomial reduction from the Multiple Edge Disjoint Paths (MEDP) problem, where given an undirected graph G, three pairs (s_1, t_1), (s_2, t_2), and (s_3, t_3) of terminals and three integers n_1, n_2, and n_3 one asks whether there is a set of pairwise edge disjoint paths containing n_1 paths between s_1 and t_1, n_2 paths between s_2 and t_2, and n_3 paths between s_3 and t_3. MEDP is known to be strongly NP-complete [21, Theorem 4].

7 Conclusion

Our results close a wide gap in the understanding of the complexity landscape of EDP parameterized by structural parameterizations. On the positive side we present three novel algorithms for the classical EDP problem: a polynomial-time algorithm for instances with a FVS of size one, an fpt-algorithm w.r.t. the treewidth and maximum degree of the input graph, and an fpt-algorithm for instances that have a small deletion set into small components in the augmented graph. On the negative side we solve a long-standing open problem concerning the complexity of EDP parameterized by the treewidth of the augmented graph: unlike related multicut problems [13], EDP is W[1]-hard parameterized by the feedback vertex set number of the augmented graph.

References

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

2 Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michal Pilipczuk. A c^k n 5-approximation algorithm for treewidth. *SIAM J. Comput.*, 45(2):317–378, 2016.

3 Hans L. Bodlaender and Ton Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *J. Algorithms*, 21(2):358–402, 1996.

4 Chandra Chekuri, Sanjeev Khanna, and F. Bruce Shepherd. An O(sqrt(n)) approximation and integrality gap for disjoint paths and unsplittable flow. *Theory of Computing*, 2(7):137–146, 2006.

5 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms.* Springer, 2015.

6 Reinhard Diestel. *Graph Theory.* Springer-Verlag, Heidelberg, 4th edition, 2010.

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

8 Pavel Dvorák, Eduard Eiben, Robert Ganian, Dusan Knop, and Sebastian Ordyniak. Solving integer linear programs with a small number of global variables and constraints. In *IJCAI 2017*, pages 607–613, 2017.

9 Alina Ene, Matthias Mnich, Marcin Pilipczuk, and Andrej Risteski. On routing disjoint paths in bounded treewidth graphs. In *Proc. SWAT 2016*, volume 53 of *LIPIcs*, pages 15:1–15:15. Schloss Dagstuhl, 2016.

10 Krzysztof Fleszar, Matthias Mnich, and Joachim Spoerhase. New algorithms for maximum disjoint paths based on tree-likeness. In *Proc. ESA 2016*, pages 42:1–42:17, 2016.

11 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*, volume XIV of *Texts in Theoretical Computer Science. An EATCS Series.* Springer Verlag, Berlin, 2006.

12 Naveen Garg, Vijay V. Vazirani, and Mihalis Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18(1):3–20, 1997.

13 Georg Gottlob and Stephanie Tien Lee. A logical approach to multicut problems. *Inf. Process. Lett.*, 103(4):136–141, 2007.

14 Richard M Karp. On the computational complexity of combinatorial problems. *Networks*, 5(1):45–68, 1975.

15 Ken-ichi Kawarabayashi, Yusuke Kobayashi, and Stephan Kreutzer. An excluded half-integral grid theorem for digraphs and the directed disjoint paths problem. In *Proc. STOC 2014*, pages 70–78. ACM, 2014.

16 T. Kloks. *Treewidth: Computations and Approximations.* Springer Verlag, Berlin, 1994.

17 Stavros G. Kolliopoulos and Clifford Stein. Approximating disjoint-path problems using packing integer programs. *Math. Program.*, 99(1):63–87, 2004.

18 H. W. Lenstra and Jr. Integer programming with a fixed number of variables. *MATH. OPER. RES*, 8(4):538–548, 1983.

19 Takao Nishizeki, Jens Vygen, and Xiao Zhou. The edge-disjoint paths problem is NP-complete for series-parallel graphs. *Discrete Applied Mathematics*, 115(1-3):177–186, 2001.

20 Neil Robertson and Paul D. Seymour. Graph minors XIII. The disjoint paths problem. *J. Comb. Theory, Ser. B*, 63(1):65–110, 1995.

21 Jens Vygen. Np-completeness of some edge-disjoint paths problems. *Discrete Applied Mathematics*, 61(1):83–90, 1995.

22 Xiao Zhou, Syurei Tamura, and Takao Nishizeki. Finding edge-disjoint paths in partial k-trees. *Algorithmica*, 26(1):3–30, 2000.