Steiner Tree Parameterized by Multiway Cut and Even Less

Bart M.P. Jansen
Eindhoven University of Technology, The Netherlands

Céline M.F. Swennenhuis
Eindhoven University of Technology, The Netherlands

Abstract

In the Steiner Tree problem we are given an undirected edge-weighted graph as input, along with a set K of vertices called *terminals*. The task is to output a minimum-weight connected subgraph that spans all the terminals. The famous Dreyfus-Wagner algorithm running in $3^{|K|}\text{poly}(n)$ time shows that the problem is fixed-parameter tractable parameterized by the number of terminals. We present fixed-parameter tractable algorithms for Steiner Tree using structurally smaller parameterizations.

Our first result concerns the parameterization by a multiway cut S of the terminals, which is a vertex set S (possibly containing terminals) such that each connected component of $G - S$ contains at most one terminal. We show that Steiner Tree can be solved in $2^{\mathcal{O}(|S| \log |S|)}\text{poly}(n)$ time and polynomial space, where S is a minimum multiway cut for K. The algorithm is based on the insight that, after guessing how an optimal Steiner tree interacts with a multiway cut S, computing a minimum-cost solution of this type can be formulated as minimum-cost bipartite matching.

Our second result concerns a new hybrid parameterization called *K-free treewidth* that simultaneously refines the number of terminals $|K|$ and the treewidth of the input graph. By utilizing recent work on $\mathcal{H}$-Treewidth in order to find a corresponding decomposition of the graph, we give an algorithm that solves Steiner Tree in time $2^{\mathcal{O}(k)}\text{poly}(n)$, where k denotes the K-free treewidth of the input graph. To obtain this running time, we show how the *rank-based* approach for solving Steiner Tree parameterized by treewidth can be extended to work in the setting of K-free treewidth, by exploiting existing algorithms parameterized by $|K|$ to compute the table entries of leaf bags of a tree K-free decomposition.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Fixed parameter tractability

Keywords and phrases fixed-parameter tractability, Steiner Tree, structural parameterization, H-treewidth

Digital Object Identifier 10.4230/LIPIcs.ESA.2024.76

Related Version *Full Version*: `https://arxiv.org/abs/2406.19819`

Funding This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 803421, ReduceSearch).

1 Introduction

Steiner Tree is a famous problem in algorithmic graph theory [9, 22, 26]. In this problem, we are given an undirected edge-weighted graph G and a set K of *terminal vertices* that we need to connect using edges of the graph. The goal is to find a minimum-weight connected subgraph that spans all these terminals, commonly known as a Steiner tree. The problem has a wide array of applications in industry, such as telecommunications, designing integrated circuits, molecular biology, and object detection (e.g., [15, 21, 22, 27, 28]).

Since the Steiner Tree problem is NP-hard, we cannot hope to design an exact polynomial-time algorithm for this problem [19]. However, a popular approach is to bound the running time not just in terms of the input size n, but to also take the influence of a secondary measurement k (referred to as the *parameter*) into account. In particular, we are interested in parameters that allow for fixed-parameter tractable (FPT) algorithms, i.e., algorithms that run in time $f(k) \cdot \mathrm{poly}(n)$ where n is the size of the input, k the parameter, and $f(\cdot)$ some computable function. One of the most natural parameters to consider for Steiner Tree is $|K|$, the number of terminals. The famous algorithm by Dreyfus and Wagner [10] computes a minimum Steiner tree of an n-vertex graph in time $3^{|K|} \cdot \mathrm{poly}(n)$ using exponential space, which was later improved by several authors [13, 23]. In particular, Fomin, Kaski, Lokshtanov, Panolan, and Saurabh [12] present an algorithm that runs in single-exponential time $7.97^{|K|} \cdot \mathrm{poly}(n)$ and uses *polynomial* space. Another commonly-used parameter for Steiner Tree is the *treewidth* $\mathsf{tw}(G)$ of the input graph G, which measures its structural similarity to a tree. A straight-forward dynamic program solves the problem in time $2^{\mathcal{O}(\mathsf{tw}(G) \log \mathsf{tw}(G))}\mathrm{poly}(n)$. Using *representative sets*, a single-exponential running time of $2^{\mathcal{O}(\mathsf{tw}(G))}\mathrm{poly}(n)$ can be obtained (cf. [8]).

For many other combinatorial problems on graphs, including graph modification problems such as Vertex Cover and Odd Cycle Transversal, a lot of effort has been invested into developing FPT algorithms using structurally *smaller* parameterizations than standard measures of solution size or treewidth [1, 2, 11, 14, 17]. But to the best of our knowledge, no previous papers present FPT algorithms for Steiner Tree using parameterizations structurally smaller than the number of terminals. As our main contributions, we identify two relevant terminal-aware refined parameterizations for Steiner Tree and develop corresponding FPT algorithms.

Multiway cut. The first parameter we consider is the size of a (node) multiway cut for the terminals, i.e., a set of vertices $S \subseteq V(G)$ such that every connected component of $G - S$ contains at most one terminal. Since S is allowed to contain terminals, any instance has a multiway cut of size $|K| - 1$. In general, the size of a minimum multiway cut can be arbitrarily much smaller than $|K|$. We show that we can solve Steiner Tree in FPT time and *polynomial* space when we parameterize it by the size of a multiway cut for the terminals.

▶ **Theorem 1.1.** *There is a polynomial-space algorithm that, given as input a graph G with weight function $\mathrm{cost}\colon E(G) \to \mathbb{N}$, a set of terminals $K \subseteq V(G)$, and a multiway cut S for K, outputs a minimum-weight Steiner tree in time $2^{\mathcal{O}(|S| \log |S|)}\mathrm{poly}(n)$.*

Our algorithm handles graphs whose weights are encoded in binary, as opposed to some other algorithms in the literature ([23, 24]) whose running time scales linearly with the value of the largest weight. The assumption that a multiway cut S for K is given as input is not restrictive, as a minimum multiway S cut for K can be found in $4^{|S|}\mathrm{poly}(n)$ time and polynomial space[1], due to an algorithm by Chen, Liu, and Lu [6]. Theorem 1.1 improves upon an elementary *exponential-space* algorithm with the same running time that was presented in a master's thesis supervised by the first author [29].

***K*-free treewidth.** The second parameterization we utilize refines the size of a multiway cut. To motivate the parameter, consider the following one-player game on a graph G with terminal vertices K. In each round, one vertex is removed from each connected component. The game ends when each connected component contains at most one terminal. If there is a multiway cut $S = \{s_1, \ldots, s_k\}$ of size k, then the game can be won in at most k rounds by choosing vertex s_i in round i (or earlier, if not all remaining vertices of S belong to the same

[1] The algorithm from Chen, Liu, and Lu [6] is a bounded-depth branching algorithm, branching on important separators, making it a polynomial-space algorithm.

connected component). The minimum number of rounds needed is therefore never larger than the size of a minimum multiway cut. It can be arbitrarily much smaller: if deletions early in the game split the graph into multiple components, these are handled “in parallel” in subsequent rounds of the game.

For technical reasons, it will be convenient to consider the variation of the game that only ends when *all* terminals have been removed from the graph, rather than ending when all terminals are separated. The number of rounds needed to win the latter variation is at most one more than the original: all terminals belong to different components when the original game ends, so at that point one additional round can delete one vertex from each connected component to eliminate all the terminals. Let $\mathsf{ed}_K(G, K)$ (the *elimination distance* to a K-free graph) denote the minimum number of rounds needed to eliminate all terminals from the graph. The previous discussion shows that if S is a multiway cut for K in G, then we have $\mathsf{ed}_K(G, K) \leq |S| + 1 \leq |K|$; hence it is a refined parameterization for Steiner Tree. The exponential dependence of our second algorithm can be bounded in terms of $\mathsf{ed}_K(G, K)$. But there is an even smaller parameterization, called *K-free treewidth*, that can be used to upper-bound the running time of the second algorithm we present. To introduce it, we briefly summarize an analogous range of parameterizations for vertex-deletion problems.

Our refined parameterizations for Steiner Tree are inspired by recent work on parameterized algorithms for $\mathcal{H}$-Deletion, which asks to find a minimum vertex set X in an input graph G that ensures $G - X$ belongs to graph class $\mathcal{H}$. The Odd Cycle Transversal problem is a prime example, which arises by letting $\mathcal{H}$ be the class bip of bipartite graphs. Recent work on $\mathcal{H}$-Deletion [1, 2, 11, 17, 18] has focused on improving parameterizations by the size of the deletion set X to parameterizations in terms of the *elimination distance* $\mathsf{ed}_{\mathcal{H}}(G)$ *to* $\mathcal{H}$, which is the minimum number of rounds needed to obtain a graph in $\mathcal{H}$ when removing one vertex from each connected component in each round. One of the results from this direction of work shows that Odd Cycle Transversal can be solved in time $2^{\mathcal{O}(k)}\text{poly}(n)$, where $k = \mathsf{ed}_{\mathsf{bip}}(G)$ [18]. Through work of Bulian and Dawar [5], it is known that the concept of elimination distance is related to the *treedepth* [25] of a graph G: the treedepth is the minimum number of rounds needed to eliminate *all* vertices.

The famous graph parameter *treewidth* is never larger than treedepth, but can be much smaller. Eiben et al. [11] proposed the notion of *$\mathcal{H}$-treewidth*, where $\mathcal{H}$ is a class of graphs. Roughly speaking, the $\mathcal{H}$-treewidth $\mathsf{tw}_{\mathcal{H}}(G)$ of a graph G can be defined in terms of the minimum cost of a tree decomposition of a certain kind, in which (potentially large) leaf bags that induce a subgraph belonging to $\mathcal{H}$ do not contribute to the cost. Hence $\mathcal{H}$-treewidth captures how efficiently a graph can be decomposed into subgraphs belonging to $\mathcal{H}$ along small separators in a treelike manner. It is known [17, Lemma 2.4] that $\mathsf{tw}_{\mathcal{H}}(G) \leq \mathsf{ed}_{\mathcal{H}}(G)$ for all graphs G, so that the resulting parameterization refines the $\mathcal{H}$-elimination distance. At the same time, $\mathsf{tw}_{\mathcal{H}}(G)$ is not larger than the standard treewidth of G, so that the resulting parameterization is a hybrid [2] of standard treewidth and the solution size for $\mathcal{H}$-deletion. In their work on $\mathcal{H}$-treewidth, Jansen, de Kroon and Włordarczyk [17] suggested that it may be interesting to explore variations of this parameter in the context of a set of terminal vertices. This is the route we pursue for Steiner Tree.

We complete the analogy between hybrid parameterizations for $\mathcal{H}$-Deletion and our parameterizations for Steiner Tree by introducing a notion called *K-free treewidth*, denoted $\mathsf{tw}_K(G, K)$ for a graph G with terminal set K. While we defer formal definitions to Section 3, it intuitively captures how efficiently the input graph can be decomposed into terminal-free subgraphs along small separators in a tree-like manner. Using the correspondence between treewidth and treedepth, it follows that $\mathsf{tw}_K(G, K) \leq \mathsf{ed}_K(G, K) \leq |S| + 1 \leq |K|$ where S is a minimum multiway cut. Our second main result shows that the resulting parameterization admits an algorithm whose running time and space usage are single-exponential.

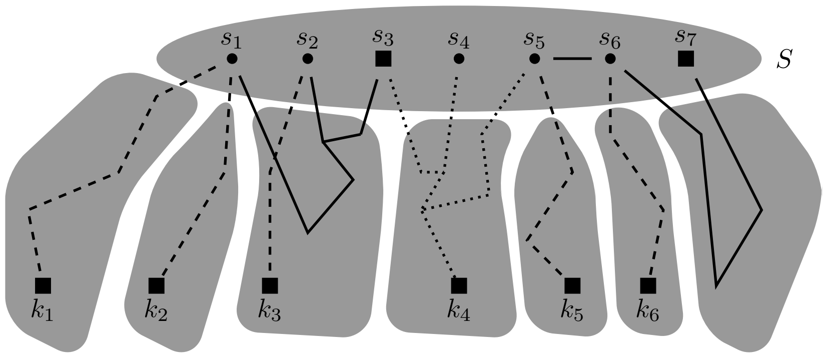

Figure 1 The multiway cut $S = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7\}$ ensures that no two terminals (squares) belong to the same connected component of $G - S$. The gray areas indicate the set S and the components of $G - S$. The lines are a visual representation of a Steiner tree F for the terminals, split at S, such that F is partitioned into three trees of category 1 (solid trees, note the solid tree between s_5 and s_6), one tree of category 2 (dotted trees) and five paths of category 3 (dashed paths).

▶ **Theorem 1.2.** *(★) There is an algorithm that takes as input a graph G with weight function* cost: $E(G) \to \mathbb{N}$*, and a set $K \subseteq V(G)$ of terminals, that computes a minimum-weight Steiner tree in $2^{\mathcal{O}(\mathsf{tw}_K(G,K))}\mathrm{poly}(n)$ time and space.*

Theorem 1.2 shows that optimal solutions to Steiner Tree can still be found efficiently in inputs that can be decomposed into terminal-free (but potentially dense and large) subgraphs along small separators. The single-exponential dependence on $\mathsf{tw}_K(G,K)$ in the running time of our algorithm is optimal under the *Exponential Time Hypothesis* [16], and matches the single-exponential running times of the current-best algorithms parameterized by the number of terminals $|K|$ [12, 24] or treewidth [4]. Hence Theorem 1.2 shows that the generality of the refined parameter tw_K does not incur a significant computational overhead.

Techniques. Both of our algorithms utilize a subroutine to solve Steiner Tree for a small set of terminal vertices. The manner in which these subroutine results are employed differs greatly between the two, however.

We first sketch the main idea behind Theorem 1.1, which aims to find a minimum Steiner tree by utilizing a known multiway cut S. We can partition the edges of any minimum Steiner tree F by splitting F at the vertices of the multiway cut S (see Figure 1), such that any tree F' in the resulting partition intersects with at most one component of $G - S$ and falls into one of the following categories:

1. F' is a minimum-weight Steiner tree for a subset of S,
2. F' is a minimum-weight Steiner tree for a subset of S and exactly one $k \in K$, or
3. F' is a minimum-weight path from a terminal $k \in K$ to some vertex of S.

There can never be two or more vertices from $K \setminus S$ in a single tree F', as these terminals live in different components of $G - S$. Note that the paths of category 3 can be found quickly by computing a shortest path. Moreover, we prove that the Steiner trees of categories 1 and 2 are Steiner trees for a vertex set of size at most $|S| + 1$, hence we can compute the minimum weights of such Steiner trees in polynomial space with the algorithm by Fomin, Kaski, Lokshtanov, Panolan, and Saurabh [12].

We introduce the notion of *S-connecting system* to characterize how a Steiner tree F interacts with the set S. Intuitively, an S-connecting system records which types of trees F' (in terms of the classification above) arise when splitting F at S, and which vertices from $S \cup K$ are connected by these trees. The algorithm will iterate over all $2^{\mathcal{O}(|S|\log|S|)}$ distinct S-connecting systems. For each such system, we can compute an edge-weighted bipartite graph B such that an optimal Steiner tree consistent with the S-connecting system corresponds to a minimum-weight maximum matching in B. The edge-weights in B are weights of optimal Steiner trees of category 1 or 2. By exploiting the fact that each tree F' obtained by splitting an optimal tree F at S involves only few terminals, each bipartite graph B can be computed in single-exponential time.

For Theorem 1.2, the algorithm builds upon the $2^{\mathcal{O}(\mathsf{tw}(G))}\text{poly}(n)$ time algorithm from Bodlaender, Cygan, Kratsch, and Nederlof [4]. Recall that any node x of a rooted tree decomposition can be associated with a subgraph G_x, which contains all subgraphs of its children. Note that its bag $\chi(x) \subseteq V(G)$ is the *boundary* of G_x, i.e., the only vertices of G_x that can have neighbors outside G_x belong to $\chi(x)$. The algorithm goes over the tree decomposition, keeping track of possible *partial solutions*, which are Steiner trees restricted to the subgraph G_x. Instead of storing actual partial solutions, the algorithm stores how these partial solution are connected to the boundary in the form of partitions. In principle, this could lead to $\mathsf{tw}(G)^{\mathcal{O}(\mathsf{tw}(G))}$ different partitions that would need to be stored as any bag $\chi(x)$ is of size at most $\mathsf{tw}(G)+1$ by definition. However, using the rank-based approach [4], one only needs to keep a *representative set* of partitions of size at most $2^{\mathsf{tw}(G)}$; we sketch the main ideas in the following paragraph.

To obtain an algorithm parameterized by $\mathsf{tw}_K(G,K)$, for the parts of the decomposition corresponding to small separators we can use the same techniques as employed for standard treewidth. For the parts of the decomposition that are large, the decomposition ensures us that terminals can only lie on the boundary. Hence, there are only $\mathsf{tw}_K(G,K)+1$ terminals in G_x for such nodes x and we can use the Dreyfus-Wagner algorithm [10] to obtain a fixed-parameter tractable algorithm to compute partial solutions of G_x. Since there might be $\mathsf{tw}_K(G,K)^{\mathcal{O}(\mathsf{tw}_K(G,K))}$ partial solutions, this does not directly yield a single-exponential running time. To obtain Theorem 1.2, we apply the rank-based approach for these nodes, by iteratively increasing the set of vertices of the boundary that are used by considered partial solutions. As far as we know, our algorithm is the first to incorporate advanced dynamic-programming ideas such as the rank-based approach with hybrid graph decompositions. To obtain a decomposition to which we can apply this scheme, we show that a recent FPT 5-approximation to compute $\mathcal{H}$-treewidth [18] can be leveraged for K-free treewidth.

Organization. We give the polynomial-space algorithm of Theorem 1.1 in Section 2. In Section 3 we formally define K-free treewidth and give an FPT 5-approximation. In Section 4 we prove Theorem 1.2, i.e., we give a single-exponential algorithm for Steiner Tree when parameterized by K-free treewidth. We conclude in Section 5.

We use standard terminology for graphs and parameterized algorithms. Terms not defined here can be found in a textbook [7]. We write $[m]$ for $\{1, ..., m\}$ and define $\min\{\emptyset\} = \infty$. The proofs of statements marked by (★) are deferred to the full version due to space constraints.

2 Polynomial-space algorithm parameterized by multiway cut

In this section, we will consider Steiner Tree parameterized by the size of a given multiway cut S for the terminal set K. In other words, each connected component of $G - S$ contains at most one terminal. Note that S can contain terminals.

2.1 S-connecting systems

The following concept is the main focus of this section.

▶ **Definition 2.1** (S-connecting system). *Consider a graph G and $S \subseteq V(G)$. An S-*connecting system *in G is a tuple $(\mathcal{S}, \mathcal{T})$, where $\mathcal{S} = \{S_1, S_2, \dots, S_m\}$ is a collection of subsets of S and $\mathcal{T}$ is a tree, such that:*

1. *$V(\mathcal{T}) = S \cup \{u_1, \dots, u_m\}$, for $m = |\mathcal{S}|$;*
2. *for all $i \in [m]$ we have $S_i = N_{\mathcal{T}}(u_i) \subseteq S$ and $d_{\mathcal{T}}(u_i) > 1$; and*
3. *for all distinct $s, s' \in S$ it holds that if $\{s, s'\} \in E(\mathcal{T})$, then $\{s, s'\} \in E(G)$.*

We will use the following notion of self-reachable to describe that vertices are part of a common connected component.

▶ **Definition 2.2** (Self-reachable). *Consider a graph G and $S \subseteq V(G)$. The set S is* self-reachable *in G if S is contained in a single connected component of G, i.e., when there is a path in G between any pair of vertices $s, s' \in S$.*

Intuitively, if S is a multiway cut for terminal set K in graph G, and F is a Steiner tree for K containing all vertices of S, then there is an S-connecting system $\mathcal{S}$ that represents the interaction between the Steiner tree F and the multiway cut S. The tree $\mathcal{T}$ of this S-connecting system can be obtained from F by contracting each tree of $F - S$ into a single vertex, while removing the degree-1 vertices of the resulting tree that do not belong to S. Definition 2.1 ensures the corresponding set $\mathcal{S}$ is uniquely determined by $\mathcal{T}$. We say that the tree F *realizes* the resulting S-connecting system $(\mathcal{S}, \mathcal{T})$.

The following lemma follows from the definitions by a simple induction. It shows that when H is a subgraph of G in which each set S_i of an S-connecting system ($\mathcal{S} = \{S_1, \dots, S_m\}, \mathcal{T}$) is self-reachable, then the connectivity structure of the tree $\mathcal{T}$ ensures that the entire set S is self-reachable.

▶ **Lemma 2.3.** *Let $(\mathcal{S}, \mathcal{T})$ be an S-connecting system and let $H \subseteq G$ be a subgraph of G such that for all $i \in [m]$, the set S_i is self-reachable in H. Then S is self-reachable in $H' = H \cup \mathcal{T}[S]$.*

Proof. We prove that any pair $\{s, s'\}$ of vertices from S is self-reachable in H, by induction on the distance from s to s' in $\mathcal{T}$. Note that the base case is evidently true, as for distance 0 we have $s = s'$. Consider an arbitrary pair $\{s, s'\}$ and let ℓ be the distance between them in $\mathcal{T}$; the induction hypothesis is that any pair from S whose distance in $\mathcal{T}$ is less than ℓ, is self-reachable in H'.

Let $s = x_0, x_1, \dots, x_\ell = s'$ be a path in $\mathcal{T}$ from s to s'. If $x_{\ell-1} \in S$, we find by induction that $\{s, x_{\ell-1}\}$ is self-reachable in H'. Moreover, $\{x_{\ell-1}, s'\} \in E(\mathcal{T}[S])$ so $\{s'', s'\} \in E(H')$ and we find that $\{x_{\ell-1}, s'\}$ is self-reachable in H'. Hence, $\{s, s'\}$ is self-reachable in H'.

If $x_{\ell-1} \notin S$, then by definition of S-connecting system, we see that $x_{\ell-2} \in S$ as $x_{\ell-1}$ can then only contain vertices from S as its neighbors. Moreover, since we have $x_{\ell-2}, s' \in N_{\mathcal{T}}(x_{\ell-1})$ we find that $x_{\ell-2}, s' \in S_i$ for some $i \in [m]$. By assumption, S_i is self-reachable in H', so in particular $\{x_{\ell-2}, s'\}$ is self-reachable in H'. Again by induction we find that $\{s, x_{\ell-2}\}$ is self-reachable in H'. Hence, $\{s, s'\}$ is self-reachable in H'. ◀

Next, we will prove that there are at most $2^{\mathcal{O}(|S| \log |S|)}$ different S-connecting systems. For this, we first prove the following claim, bounding the size of $\mathcal{S}$.

▶ **Lemma 2.4.** *Let G be a graph and $S \subseteq V(G)$. Any S-connecting system $(\mathcal{S}, \mathcal{T})$ in G satisfies $|\mathcal{S}| \leq |S| - 1$.*

Proof. Let $(\mathcal{S}, \mathcal{T})$ be an S-connecting system. Let $\mathcal{S} = \{S_1, \ldots, S_m\}$; we prove that $m \leq |S| - 1$. Let $V(\mathcal{T}) = S \cup \{u_1, \ldots, u_m\}$ such that $N_{\mathcal{T}}(u_i) = S_i$ for all $i \in [m]$. Root $\mathcal{T}$ at an arbitrary vertex $s^* \in S$. Note that for any $i \in [m]$, u_i is not a leaf of $\mathcal{T}$ and has only neighbors in S by definition of the S-connecting system. Hence, every u_i is a parent of at least one $s \in S \setminus \{s^*\}$. Since every $s \in S$ has at most one parent we find $m \leq |S| - 1$. ◀

▶ **Lemma 2.5.** *For any graph G and vertex set $S \subseteq V(G)$, there are at most $2^{\mathcal{O}(|S| \log |S|)}$ different S-connecting systems.*

Proof. Consider an arbitrary S-connecting system $(\mathcal{S}, \mathcal{T})$ in G. Item 1 of Definition 2.1 ensures that $|V(\mathcal{T})| \leq |S| + |\mathcal{S}|$, while Lemma 2.4 ensures $|\mathcal{S}| \leq |S| - 1$. Hence $|V(\mathcal{T})| \leq 2|S| - 1$. By Cayley's formula, there are n^{n-2} different labeled trees on n vertices. Therefore, the number of different choices for $\mathcal{T}$ is bounded by $\sum_{i=1}^{2|S|-1} i^{i-2} \leq (2|S| - 1)\,(2|S| - 1)^{2|S|-3}$, i.e., by $2^{\mathcal{O}(|S| \log |S|)}$. By Definition 2.1, the collection $\mathcal{S}$ is uniquely determined by $\mathcal{T}$. ◀

2.2 The algorithm

Before we present Theorem 1.1, we describe at a high level how S-connecting systems facilitate a reduction to bipartite matching. The starting observation is that, by trying all possible subsets of the multiway cut S, we may assume that the Steiner tree we are looking for contains all vertices of S. The connectivity pattern of each Steiner tree with respect to S can then be summarized by an S-connecting system. To assemble a Steiner tree that realizes a given S-connecting system, a naïve approach is the following: pick a shortest path from each terminal to S, and for each subset S_i of the S-connecting system, pick a subtree containing S_i to make S_i self-reachable. This approach leads to some redundancy: we might be able to re-use some edges if the path used to connect a terminal t_p to S, shares some edges with a subtree that makes a subset S_i self-reachable.

Due to S being a multiway cut, a tree of $G - S$ that makes a subset S_i self-reachable can live in only one component of $G - S$, and can therefore only involve at most one terminal. For each choice of terminal t_p and subset S_i, we can use the polynomial-space FPT algorithm for Steiner Tree parameterized by $|K|$ [12] to compute the cost of a tree making $\{t_p\} \cup S_i$ self-reachable, and compare it to the shortest-path distance between t_p and the closest vertex of S to see how much we would benefit from combining the task of making t_p reachable from S with the task of making S_i self-reachable. As each terminal can only be involved in making one set S_i self-reachable, we now see a weighted bipartite matching problem appear: we are looking for a pairing of sets S_i with terminals t_p so that the overall *savings*, compared to making each terminal individually reachable from S by a shortest path and separately adding a subtree making each S_i self-reachable into the Steiner tree, are as large as possible.

These ideas are formalized in the proof of the following theorem.

▶ **Theorem 1.1.** *There is a polynomial-space algorithm that, given as input a graph G with weight function* cost$: E(G) \to \mathbb{N}$*, a set of terminals $K \subseteq V(G)$, and a multiway cut S for K, outputs a minimum-weight Steiner tree in time $2^{\mathcal{O}(|S| \log |S|)}\text{poly}(n)$.*

Proof. Consider the multiway cut S. Let $C_1, \ldots, C_q$ denote the vertex sets of the connected components of $G - S$, so that each C_p contains at most one terminal. For a vertex set X, we denote by $\delta(X)$ the set of edges of G that have exactly one endpoint inside X. Hence for $p \in [q]$, each edge of $\delta(C_p)$ has one endpoint in C_p and one endpoint in S. We define $k_p := C_p \cap K$ for all $p \in [q]$.

In our algorithm, we will guess which vertices $S' \subseteq S$ will be used by the Steiner tree. For each such guess for S', we will consider all possible S'-connecting systems $(\mathcal{S}, \mathcal{T})$. For each such system, we create a weighted complete bipartite graph B with partition $V(B) = \{\mathcal{S}\} \cup \{P\}$ where $P = \{(p, j) \colon p \in \{1, \dots, q\}, j \in \{0, \dots, |\mathcal{S}|\}\}$. For each $p \in [q]$ we define G_p as $G[C_p] \cup \delta(C_p)$, i.e., the graph induced on C_p together with the edges between C_p and S. The goal is to find a minimum-weight maximum matching in B, which represents how each set $S_i \in \mathcal{S}$ is self-reachable. If S_i is matched with $(p, 0)$, it indicates that the subtree making S_i self-reachable is contained in G_p, and that k_p is contained in this subtree. If S_i is matched with (p, j) for $j > 0$, it indicates that the subtree making S_i self-reachable is contained in G_p (not necessarily using terminal k_p if it exists). Note that the specific value of j does not carry any meaning, but we need to be able to use G_p multiple (and at most $|\mathcal{S}|$) times to make different sets in $\mathcal{S}$ self-reachable.

To determine the weights of B, we have to solve several Steiner Tree problems in succession. Let $\mathsf{MST}[H, X]$ denote an arbitrary minimum-cost Steiner tree for terminal set X in graph H and let $\mathsf{MST}[H, X] = \emptyset$ if no such Steiner tree exists. Moreover, for $X \subseteq V(G)$ and $k \in V(G)$ we refer to $\mathsf{SP}[X, k]$ as an arbitrary shortest path from k to some vertex of X and we set $\mathsf{SP}[X, k] = \emptyset$ if no such path exists. We extend this notation for the sets k_p defined above, which are either singletons or empty. If $k_p = \{t_p\}$ is a singleton set containing a terminal, we let $\mathsf{SP}[X, k_p] = \mathsf{SP}[X, t_p]$. If $k_p = \emptyset$, then $\mathsf{SP}[X, k_p] = \emptyset$.

Some of the edges of B can have infinite weight, indicating that a certain Steiner tree does not exist. We define a cost function ω for the edges of B as follows. For all $S_i \in \mathcal{S}$ and $p \in [q]$ we set

$$\omega(S_i, (p, 0)) = \begin{cases} \infty & \text{if } \mathsf{SP}[S', k_p] = \emptyset, \\ \mathrm{cost}(\mathsf{MST}[G_p, S_i \cup \{k_p\}]) - \mathrm{cost}(\mathsf{SP}[S', k_p]) & \text{else if } C_p \cap K \neq \emptyset, \\ \infty & \text{otherwise,} \end{cases}$$

and for all $S_i \in \mathcal{S}$, $p \in [q]$, and $j \in [|\mathcal{S}|]$ we set

$$\omega(S_i, (p, j)) = \mathrm{cost}(\mathsf{MST}[G_p, S_i]).$$

Note that each of these values can be computed in polynomial space and $2^{\mathcal{O}(|S'|)}\mathrm{poly}(n)$ time using the algorithm by Fomin, Kaski, Lokshtanov, Panolan, and Saurabh [12]. Since the bipartite graph B is of polynomial size, we can construct B in polynomial space and $2^{\mathcal{O}(|S'|)}\mathrm{poly}(n)$ time. We claim that we can reconstruct a Steiner tree, based on a minimum-weight maximum matching of B, if its weight is finite.

▷ Claim 2.6. (★) Consider $S' \subseteq S$ such that $K \cap S \subseteq S'$, with an S'-connecting system $(\mathcal{S}, \mathcal{T})$. Let M be a minimum-weight maximum matching of B of finite weight. Then a Steiner tree T_M for terminals K can be constructed in $\mathrm{poly}(n)$ time with

$$\mathrm{cost}(T_M) \leq \omega(M) + \mathrm{cost}(\mathcal{T}[S']) + \sum_{k \in K} \mathrm{cost}(\mathsf{SP}[S', k]).$$

We are now ready to present our algorithm. For all $S' \subseteq S$ such that $S \cap K \subseteq S'$, for all S'-connecting systems $(\mathcal{S}, \mathcal{T})$, construct the weighted complete bipartite graph B as above. Then compute a minimum-weight maximum matching M of B. Using Claim 2.6 we construct a Steiner tree if the weight of M is finite in polynomial time. Finally, the algorithm outputs the minimum-weight Steiner tree found during this process.

The algorithm runs in $2^{\mathcal{O}(|S|\log|S|)}\mathrm{poly}(n)$ time and polynomial space: we consider $2^{|S|}$ different sets S', for each there are at most $2^{\mathcal{O}(|S'|\log|S'|)}$ different S'-connecting systems by Lemma 2.5. Computing the weights of B then takes $2^{\mathcal{O}(|S'|)}\mathrm{poly}(n)$ time and polynomial space using the algorithm by Fomin, Kaski, Lokshtanov, Panolan, and Saurabh [12]. Finding a minimum-cost maximum matching of B takes $\mathrm{poly}(n)$ time and space. Constructing a Steiner tree based on M takes only polynomial time and space. It remains to prove that the output of the algorithm is a minimum Steiner tree.

▷ Claim 2.7. (★) The algorithm outputs a minimum-weight Steiner tree.

This concludes the proof of Theorem 1.1. ◀

3 Tree K-free-decompositions

In this section we formally define K-free treewidth as the minimum width of a tree K-free-decomposition. We also show how to compute a 5-approximation in FPT time.

The definition of *K-free treewidth* closely resembles the notion of tree $\mathcal{H}$-decomposition that inspired it [11, 18]. The difference lies in the parts of the decomposition that do not contribute to the cost. Throughout this section, K can be any set of vertices. However, we will use K as the set of Steiner Tree terminals in the rest of this paper.

▶ **Definition 3.1** (Tree K-free-decomposition). *For a graph G and a set $K \subseteq V(G)$, a* tree K-free-decomposition *of graph G is a triple $(\mathbb{T}, \chi, L)$ where $L \subseteq V(G)$, $\mathbb{T}$ is a rooted tree, and $\chi : V(\mathbb{T}) \to 2^{V(G)}$, such that:*

(K.A) *for each $v \in V(G)$, the nodes in $\{x : v \in \chi(x)\}$ form a non-empty connected subtree of $\mathbb{T}$,*
(K.B) *for each edge $uv \in E(G)$, there is a node $x \in V(\mathbb{T})$ with $\{u, v\} \subseteq \chi(x)$,*
(K.C) *for each $v \in L$, there is a unique $x \in V(\mathbb{T})$ with $v \in \chi(x)$, and x is a leaf of $\mathbb{T}$,*
(K.D) *for each node $x \in V(\mathbb{T})$, we have $\chi(x) \cap L \cap K = \emptyset$.*
The width of a tree K-free-decomposition $(\mathbb{T}, \chi, L)$ is defined as $\max\{0, \max_{x \in V(\mathbb{T})} |\chi(x) \setminus L| - 1\}$. The K-free-treewidth of a graph G, denoted $\mathsf{tw}_K(G, K)$, is the minimum width of a tree K-free-decomposition of G.

The first two items in this definition ensure that the pair $(\mathbb{T}, \chi)$ forms a valid (standard) tree decomposition. The vertex set L, which must be disjoint from the terminal set K, corresponds to the vertices in terminal-free subgraphs that are not decomposed further. Each vertex from L occurs in exactly one bag, which is a leaf of $\mathbb{T}$. The vertices from L do not contribute to the cost of the decomposition, which corresponds to the fact that the sets $\chi(t) \cap L$ of leaf nodes t can represent arbitrarily large terminal-free subgraphs. To obtain the definition of $\mathcal{H}$-treewidth for a fixed graph class $\mathcal{H}$, it suffices to omit K from the definition and replace condition **(K.D)** by the requirement that for each node $x \in V(\mathbb{T})$, the graph $G[\chi(x) \cap L]$ belongs to $\mathcal{H}$.

▶ **Theorem 3.2.** *(★) There is an algorithm that takes an input graph G, vertex set $K \subseteq V(G)$, integer k, and either computes a tree K-free-decomposition of width at most $5k + 5$ consisting of $\mathcal{O}(n)$ nodes, or correctly concludes that $\mathsf{tw}_K(G, K) > k$. The algorithm runs in time $2^{\mathcal{O}(k)} \cdot \mathrm{poly}(n)$ and polynomial space.*

Proof sketch. Jansen, de Kroon, and Włodraczyk [18] recently showed that a 5-approximation to $\mathcal{H}$-treewidth can be computed in single-exponential time, provided that $\mathcal{H}$ is a hereditary and union-closed graph class for which $\mathcal{H}$-Deletion has a single-exponential

FPT algorithm parameterized by solution size. We transform the input graph G with its terminal set K into a graph $\hat{G}$ by subdividing each edge of G and attaching a triangle onto each terminal vertex $k_i \in K$ via two new degree-two vertices k_i', k_i''. The subdivisions ensure that the connectivity structure of the graph remains unchanged but turns triangles into 6-cycles. Hence the only triangles of $\hat{G}$ correspond to the terminal vertices K. We show that for the class $\mathcal{H}$ of triangle-free graphs, the $\mathcal{H}$-treewidth of $\hat{G}$ is closely related to the K-free treewidth of G. Since Triangle-Free Deletion parameterized by solution size is FPT via bounded-depth branching, the above-mentioned approach gives a 5-approximation for triangle-free treewidth on $\hat{G}$. Using the correspondence between triangle-free treewidth of $\hat{G}$ and K-free treewidth of G, this yields the desired algorithm. ◀

3.1 Nice tree K-free-decompositions

Many algorithms using standard tree decompositions use *nice* tree decompositions (e.g., [7]). We will use a very similar notion of a *nice tree K-free-decomposition*, where every non-leaf node of a rooted tree decomposition $(\mathbb{T}, \chi)$ is one of five types:

- **join node:** a node x with exactly two child nodes c_1, c_2, with $\chi(x) = \chi(c_1) = \chi(c_2)$.
- **vertex introduce node:** a node x that *introduces a vertex* $v \in V(G)$ has exactly one child node c, such that $\chi(x) = \chi(c) \cup \{v\}$ with $v \notin \chi(c)$.
- **vertex forget node:** a node x that *forgets a vertex* $v \in V(G)$ has exactly one child node c, such that $\chi(x) = \chi(c) \setminus \{v\}$ with $v \in \chi(c)$.
- **edge introduce node:** a node x that *introduces an edge* $\{u, v\}$ has exactly one child node c, such that $\chi(x) = \chi(c)$ and $\{u, v\} \subseteq \chi(x)$.
- **leaf introduce node:** a node x that has exactly one child node c that is a leaf, such that $\chi(x) = \chi(c) \setminus L$.

We say that in a leaf node c, all edges of $G[\chi(c) \setminus L]$ are introduced. In a leaf introduce node x with child node c, all remaining edges (i.e., all edges of $G[\chi(x)]$ except those in $G[\chi(c) \setminus L]$) are introduced. We require that every edge is introduced exactly once in the whole decomposition. We use the notation G_x to denote the subgraph of G with $V(G_x)$ the union of all bags of descendants of x in $\mathbb{T}$ and $E(G_x)$ the edges that are introduced by node x or a descendant of x.

Note that we can transform any tree K-free-decomposition into a *nice* tree K-free-decomposition, in almost exactly the same way as transforming a standard tree decomposition into a nice tree decomposition (cf. [7, Lemma 7.4], [20]). First we apply the usual transformations on the interior of $\mathbb{T}$, i.e.we ensure that $\mathbb{T}$ is binary and introduce / forget at most one vertex per bag. Note that we only introduce edges that are not part of $G[\chi(x)]$ for a leaf $x \in V(\mathbb{T})$. Then, we will do the following transformation for each leaf $c \in V(\mathbb{T})$: Remove the edge between c and its parent p and add a node x as a child of p and a parent of c. Set $\chi(x) = \chi(c) \setminus L$. Now x is a leaf introduce node, introducing the leaf c. This ensures all the properties of a nice tree decomposition without increasing the width of the decomposition.

4 Single-exponential time algorithm parameterized by K-free treewidth

In order to obtain a single-exponential algorithm even though the number of partitions of a size-k set is super-exponential in k, we use the rank-based approach by Bodlaender, Cygan, Kratsch, and Nederlof [4]. The main idea behind this approach is to derive single-exponential rank bounds for matrices that encode which pairs of "solutions" combine into a full solution. These bounds imply that any set of partial solutions can be trimmed down to a *representative* subset of single-exponential size while providing the following guarantee: if the original set contained a partial solution that can be extended to an optimal full solution, then the representative set still contains such a partial solution.

We start making these ideas concrete for Steiner Tree. Given a node x of a tree K-free decomposition $(\mathbb{T}, \chi, L)$ of the input graph G obtained via Theorem 3.2, we will work bottom-up in the decomposition tree. For each node $x \in V(\mathbb{T})$ we compute a set of partial solutions in the associated graph G_x defined in Section 3.1, using the previously computed data for the children of x. During this process, we decide which partial solutions in G_x to keep based on their behaviour on the boundary, i.e., on $\chi(x)$. Roughly speaking, a partial solution corresponds to a subgraph F of G_x containing all vertices of $K \cap V(G_x)$. The subgraph F does not have to be connected, but if it is disconnected then each connected component contains a vertex of $\chi(x)$. The behavior of F can be summarized by a partition of the vertices of $\chi(x) \cap V(F)$ based on their spread over connected components of F: two vertices $u, v \in \chi(x) \cap V(F)$ are in the same set of the partition if and only if they are in a common connected component of F. To work with representative sets of partial solutions, we therefore need some terminology to work with partitions. We view a partition of a set U as a set of disjoint subsets of U whose union is U.

▶ **Definition 4.1.** *For a finite set U we use $\Pi(U)$ to denote the set of all partitions of U. For $P, Q \in \Pi(U)$ we denote by $P \sqcup Q$ the* join *of partitions P and Q in the partition lattice, which can be obtained as follows. Let G_P (resp. G_Q) be a graph on vertex set U whose connected components partition U according to P (resp. Q). Then $P \sqcup Q$ corresponds to the partition of U formed by the connected components of the union $G_P \cup G_Q$. This implies that for every $A \in P \cup Q$ there exists $A' \in P \sqcup Q$ such that $A \subseteq A'$.*

Using this notation, we can introduce the concept of a set $\mathcal{R}$ of partial solutions *representing* another $\mathcal{A}$. To facilitate the discussion of our algorithm, we will work with partial solutions of two types. The most intuitive form consists of subgraphs F of G_x as described above. A more succinct representation of the essential information can be obtained by encoding each such subgraph F as a pair (P, w), where P is the partition it induces on the boundary vertices and w is the integer giving the total cost of the edges in F. From this perspective, the objects stored in a set of partial solutions consist of pairs (P, w). The following definition captures the idea of two partial solutions together forming a full solution: they merge together to provide a subgraph consisting of a single connected component whenever the partition representing the combined connectivity information consists of a single set containing the entire boundary.

▶ **Definition 4.2** (Definition 3.4 from [4]). *For a universe U and two sets $\mathcal{A}, \mathcal{R} \subseteq \{(P, w) : P \in \Pi(U) \wedge w \in \mathbb{N}\}$, we say that $\mathcal{R}$* represents *$\mathcal{A}$ if for all $Q \in \Pi(U)$ we have*

$$\min\{w \colon (P, w) \in \mathcal{A} \wedge P \sqcup Q = \{U\}\} = \min\{w \colon (P, w) \in \mathcal{R} \wedge P \sqcup Q = \{U\}\}.$$

Bodlaender et al. [4] showed that a representative set $\mathcal{R}$ can be computed by finding a basis of a suitable matrix of rank $2^{|U|-1}$. We can therefore bound the size of $\mathcal{R}$ by $2^{|U|-1}$.

▶ **Proposition 4.3** (Theorem 3.7 from [4]). *There is an algorithm that given a universe U and a set $\mathcal{A} \subseteq \{(P, w) : P \in \Pi(U) \wedge w \in \mathbb{N}\}$ in time $2^{\mathcal{O}(|U|)}\text{poly}(|\mathcal{A}|)$ finds a set $\mathcal{R} \subseteq \mathcal{A}$ of size at most $2^{|U|-1}$, that represents $\mathcal{A}$.*

We slightly adjust this proposition to also work for storing subgraphs (i.e., partial solutions) as representative sets, instead of partitions. This facilitates the presentation of the remainder of the algorithm. For this purpose, we define how to map subgraphs to partitions.

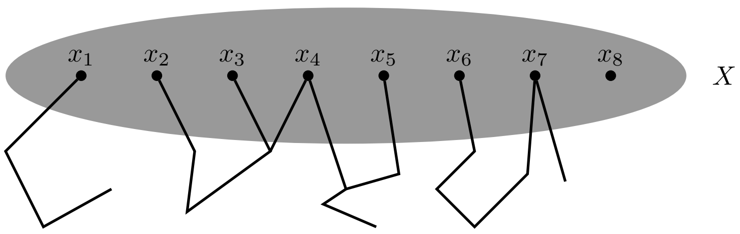

Figure 2 In this example we have $X = \{x_1, \ldots, x_8\}$. The lines are a visual representation of a subgraph F of G, the graph G itself is not drawn. For this subgraph F we have $\pi_F(X) = \{\{x_1\}, \{x_2, x_3, x_4, x_5\}, \{x_6, x_7\}, \{x_8\}\}$.

▶ **Definition 4.4.** *For a graph G, a set $X \subseteq V(G)$, and $F \subseteq G$ a subgraph of G, let $\pi_F(X)$ be the partition of X such that:*

- *any $u \in X \setminus V(F)$, we have $\{u\} \in \pi_F(X)$,*
- *for any $u, v \in X \cap V(F)$, vertices u and v are in the same set of $\pi_F(X)$ if and only if u and v belong to a common connected component of F.*

See Figure 2 for an example of a subgraph F and the partition $\pi_F(X)$. Observe that if C is a connected component of F and $H = F \setminus C$, then $\pi_F(X) = \pi_H(X) \sqcup \pi_C(X)$; we will use this fact later. Next, we adjust the definition of representing to work for subgraphs. We use $\{F \subseteq G\}$ to denote the set of all subgraphs of a graph G.

▶ **Definition 4.5.** *Given an edge-weighted graph G, a vertex set $Z \subseteq V(G)$, and two sets of subgraphs $\mathcal{B}, \mathcal{R} \subseteq \{F \subseteq G\}$, we say that $\mathcal{R}$* represents *$\mathcal{B}$* on *Z if for all $Q \in \Pi(Z)$ we have*

$$\min\{\text{cost}(F) \colon F \in \mathcal{B} \wedge \pi_F(Z) \sqcup Q = \{Z\}\} = \min\{\text{cost}(F) \colon F \in \mathcal{R} \wedge \pi_F(Z) \sqcup Q = \{Z\}\}. \quad (1)$$

The given definitions support the following straight-forward extension of Proposition 4.3, which facilitates computing representative subsets of partial solutions stored as subgraphs. The computation works with respect to *any* choice of vertex set Z as boundary. The fact that Z is not necessarily a separator in G is crucial for its later use.

▶ **Corollary 4.6.** *Consider an edge-weighted graph G and a subset of the vertices $Z \subseteq V(G)$. For a given set of subgraphs $\mathcal{B} \subseteq \{F \subseteq G\}$, we can find in $2^{\mathcal{O}(|Z|)}\text{poly}(|\mathcal{B}|)$ time a set $\mathcal{R} \subseteq \mathcal{B}$ of size at most $2^{|Z|-1}$ that represents $\mathcal{B}$ on Z.*

We now present the main ingredient for applying the rank-based approach for K-free treewidth. It will be used to compute the table entries of the leaf-introduce vertices of the tree K-free decomposition.

▶ **Lemma 4.7.** *Given as input an edge-weighted n-vertex graph G and two disjoint subsets of vertices $Y \subseteq V(G)$ and $Z \subseteq V(G)$ with $Z \neq \emptyset$, we can compute in $2^{\mathcal{O}(|Z|)} \cdot \text{poly}(n)$ time a set $\mathcal{R}(Z) \subseteq \mathcal{F}(Z) = \{F \subseteq G[Z \cup Y]\}$ such that:*

- *$|\mathcal{R}(Z)| \leq 2^{|Z|-1}$, and*
- *$\mathcal{R}(Z)$ represents $\mathcal{F}(Z)$ on Z.*

Proof. We want to find a representative set for all possible subgraphs in $G[Y \cup Z]$. For this purpose, we build $\mathcal{F}(Z)$ by increasing the size of Z, while maintaining the bounded size of $\mathcal{F}(Z)$. Order the elements of Z arbitrarily, i.e., let $Z = \{z_1, \ldots, z_k\}$. Let $Z_i = \{z_1, \ldots, z_i\}$ be the first i elements of Z and define $\mathcal{F}(Z_i) = \{F \subseteq G[Z_i \cup Y]\}$. We will iteratively compute sets $\mathcal{R}(Z_i) \subseteq \mathcal{F}(Z_i)$ with the following properties:

- $|\mathcal{R}(Z_i)| \leq 2^{|Z|-1}$, and
- $\mathcal{R}(Z_i)$ represents $\mathcal{F}(Z_i)$ on Z.

We set $\mathcal{R}(\emptyset) = \{\emptyset\}$. To compute $\mathcal{R}(Z_i)$ from $\mathcal{R}(Z_{i-1})$ we execute the following steps.

1. Initialize $\mathcal{B}(Z_i) = \mathcal{R}(Z_{i-1})$.
2. For all $F \in \mathcal{R}(Z_{i-1})$, for all $T \subseteq Z_i$ such that $z_i \in T$, compute a minimum-cost Steiner tree F_T for terminals T in $G[Z_i \cup Y]$ in $2^{\mathcal{O}(|Z|)}\text{poly}(n)$ time using the Dreyfus-Wagner algorithm [10]. Add $F \cup F_T$ to $\mathcal{B}(Z_i)$.
3. Compute a representative set $\mathcal{R}(Z_i) \subseteq \mathcal{B}(Z_i)$ with $|\mathcal{R}(Z_i)| \leq 2^{|Z|-1}$ using Corollary 4.6.

The resulting set $\mathcal{R}(Z_k) = \mathcal{R}(Z)$ is given as the output. The main work happens in Step 2. For each of the $|\mathcal{R}(Z_{i-1})| \leq 2^{|Z|-1}$ choices for F, there are at most $2^{|Z|}$ choices for T. Hence we invoke the Dreyfus-Wagner algorithm $2^{\mathcal{O}(|Z|)}$ times for a terminal set of size $|T| \leq |Z|$. The resulting set $\mathcal{B}(Z_i)$ has size $2^{\mathcal{O}(|Z|)}$, so that Corollary 4.6 also runs in time $2^{\mathcal{O}(|Z|)}\text{poly}(n)$ for each of the $k+1 \in \mathcal{O}(n)$ choices of i. The run-time bound follows.

We prove correctness by induction over i. For the base case $i = 0$, we argue that $\mathcal{R}(\emptyset)$ satisfies the two conditions. The set $\mathcal{F}(Z_0) = \mathcal{F}(\emptyset)$ only contains subgraphs F of $G[Y]$ and for any of these we have $\pi_F(Z) = \{\{z_1\}, \{z_2\}, \ldots, \{z_k\}\}$ since $Y \cap Z = \emptyset$. Hence, the only $Q \in \Pi(Z)$ satisfying $\pi_F(Z) \sqcup Q = \{Z\}$ is $Q = \{\{Z\}\}$. Since for $F = \emptyset$ we have $\pi_F(Z) \sqcup Q = \{Z\}$, we find that $\mathcal{R}(\emptyset) = \{\emptyset\}$ is a representative set for $\mathcal{F}(\emptyset)$ on Z.

For the case $i > 0$, first note that any time we add a subgraph $F \cup F_T$ to $\mathcal{B}(Z_i)$ in Step 2, we have $F \in \mathcal{R}(Z_{i-1}) \subseteq \mathcal{F}(Z_{i-1}) \subseteq \mathcal{F}(Z_i)$, and $F_T \subseteq G[Z_i \cup Y]$. Hence their union is a subgraph of $G[Z_i \cup Y]$. Therefore, $\mathcal{R}_i(Z) \subseteq \mathcal{B}_i(Z) \subseteq \mathcal{F}(Z_i)$. Moreover, we have $|\mathcal{R}(Z_i)| \leq 2^{|Z|-1}$ because we constructed it using Corollary 4.6. All that remains to show is that $\mathcal{R}(Z_i)$ represents $\mathcal{F}(Z_i)$ on Z, i.e., for all $Q \in \Pi(Z)$ we have the following.

$$\min\{\text{cost}(F) \colon F \in \mathcal{F}(Z_i) \wedge \pi_F(Z) \sqcup Q = \{Z\}\} = \min\{\text{cost}(F) \colon F \in \mathcal{R}(Z_i) \wedge \pi_F(Z) \sqcup Q = \{Z\}\}$$

As $\mathcal{R}(Z_i) \subseteq \mathcal{F}(Z_i)$ we directly see that the left-hand side is at most the right-hand side. For the other direction, fix $Q \in \Pi(Z)$ and choose $F \in \mathcal{F}(Z_i)$ that minimizes the left-hand side. Let C be the connected component of F containing z_i (possibly $C = \emptyset$). Let $H = F \setminus C$. Note that $H \in \mathcal{F}(Z_{i-1})$ as now $H \subseteq G[Z_{i-1} \cup Y]$. Take $Q' = \pi_C(Z) \sqcup Q$. We find that

$$\pi_H(Z) \sqcup Q' = \pi_H(Z) \sqcup \pi_C(Z) \sqcup Q = \pi_F(Z) \sqcup Q = \{Z\}.$$

Using induction, we find that there exists $H' \in \mathcal{R}(Z_{i-1})$ such that $\pi_{H'}(Z) \sqcup Q' = \{Z\}$ and $\text{cost}(H') = \text{cost}(H) = \text{cost}(F) - \text{cost}(C)$. The algorithm at one point considered H' and $T = V(C) \cap Z$ in Step 2, and added $H' \cup F_T$ to $\mathcal{B}(Z_i)$. Note that

$$\begin{aligned}
\pi_{H' \cup F_T}(Z) \sqcup Q &= \pi_{H' \cup F_T}(Z) \sqcup Q \sqcup \pi_C(Z) && \text{as } F_T \text{ connects } V(C) \cap Z \\
&= \pi_{H' \cup F_T}(Z) \sqcup Q' && \text{by definition of } Q' \\
&= \{Z\}. && \text{as } \pi_{H'}(Z) \sqcup Q' = \{Z\}
\end{aligned}$$

Using Proposition 4.3, we find that there exists a subgraph $F' \in \mathcal{R}(Z_i)$ such that $F' \sqcup Q = \{Z\}$. To bound its cost, note that $\text{cost}(F_T) \leq \text{cost}(C)$ as F_T is a minimum-cost Steiner tree in $G[Z_i \cup Y]$ for terminal set T and C is a candidate solution. Then we derive

$$\begin{aligned}
\text{cost}(F') &= \text{cost}(H' \cup F_T) \leq \text{cost}(H') + \text{cost}(F_T) = \text{cost}(F) - \text{cost}(C) + \text{cost}(F_T) \\
&\leq \text{cost}(F) - \text{cost}(C) + \text{cost}(C) = \text{cost}(F).
\end{aligned}$$

Hence, F' shows that the left-hand side of the equation is greater than or equal to the right-hand side. This concludes the proof of Lemma 4.7. ◀

We can use Lemma 4.7 to solve Steiner Tree in $2^{\mathcal{O}(\mathsf{tw}_K(G,K))}\text{poly}(n)$ time. In the full version, we give a self-contained presentation of the existing [4] rank-based dynamic-programming algorithm for Steiner Tree parameterized by treewidth – thereby correcting some small issues from the literature – and show how Lemma 4.7 allows it to be extended for K-free treewidth. It leads to a proof of Theorem 1.2.

5 Conclusion

In this paper, we show that it is possible to design FPT algorithms for Steiner Tree when the considered parameter is structurally smaller than the number of terminals. For this purpose, we have extended the definitions of $\mathcal{H}$-treewidth and $\mathcal{H}$-elimination distance to terminal-free variants of these parameters.

Our polynomial-space algorithm for Steiner Tree parameterized by multiway cut is slightly superexponential, due to the number of different S-connecting systems. Is it possible to design an algorithm for Steiner Tree parameterized by $|S|$ that uses both polynomial space *and* single-exponential FPT time?

Another direction for future research is to apply such terminal-aware parameterizations to other problems that rely on terminals. An obvious example is Multiway Cut itself. Another problem to consider is Shortest K-Cycle, where one is given an undirected graph and a set of terminals K and asked to find a simple cycle that goes through all terminals. Björklund, Husfeldt, and Taslaman [3] give a $2^{|K|}\text{poly}(n)$ time algorithm for this problem. Is it fixed-parameter tractable parameterized by K-free treewidth? The same question can be asked for the Subset Traveling Salesperson Problem (Subset TSP), which asks for a minimum-weight tour visiting a given subset K of terminal vertices. Many techniques developed for Steiner Tree parameterized by treewidth also apply to Hamiltonian Cycle and other variants of TSP [4, 8]; we expect that our techniques for terminal-aware parameterizations can be extended for Subset TSP.

References

1 Akanksha Agrawal, Lawqueen Kanesh, Daniel Lokshtanov, Fahad Panolan, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. Deleting, eliminating and decomposing to hereditary classes are all FPT-equivalent. In *Proceedings of SODA 2022*, pages 1976–2004, 2022. doi:10.1137/1.9781611977073.79.

2 Akanksha Agrawal and M. S. Ramanujan. Distance from triviality 2.0: Hybrid parameterizations. In *Proc. 33rd IWOCA*, volume 13270 of *Lecture Notes in Computer Science*, pages 3–20. Springer, 2022. doi:10.1007/978-3-031-06678-8_1.

3 Andreas Björklund, Thore Husfeldt, and Nina Taslaman. Shortest cycle through specified elements. In *Proc. 23rd SODA*, pages 1747–1753. SIAM, 2012. doi:10.1137/1.9781611973099.139.

4 Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Information and Computation*, 243:86–111, 2015. doi:10.1016/j.ic.2014.12.008.

5 Jannis Bulian and Anuj Dawar. Graph isomorphism parameterized by elimination distance to bounded degree. *Algorithmica*, 75(2):363–382, 2016. doi:10.1007/S00453-015-0045-3.

6 Jianer Chen, Yang Liu, and Songjian Lu. An improved parameterized algorithm for the minimum node multiway cut problem. *Algorithmica*, 55(1):1–13, 2009. doi:10.1007/S00453-007-9130-6.

7 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

8 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michał Pilipczuk, Johan M.M. Van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. *ACM Transactions on Algorithms (TALG)*, 18(2):1–31, 2022. doi:10.1145/3506707.

9 11th DIMACS Implementation Challenge, 2014. Accessed 21 Nov 2023. URL: https://dimacs11.zib.de/.

10 Stuart E. Dreyfus and Robert A. Wagner. The Steiner problem in graphs. *Networks*, 1(3):195–207, 1971. doi:10.1002/net.3230010302.

11 Eduard Eiben, Robert Ganian, Thekla Hamm, and O joung Kwon. Measuring what matters: A hybrid approach to dynamic programming with treewidth. *Journal of Computer and System Sciences*, 121:57–75, 2021. doi:10.1016/j.jcss.2021.04.005.

12 Fedor V Fomin, Petteri Kaski, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Parameterized single-exponential time polynomial space algorithm for Steiner tree. *SIAM Journal on Discrete Mathematics*, 33(1):327–345, 2019. doi:10.1137/17M1140030.

13 Bernhard Fuchs, Walter Kern, D Molle, Stefan Richter, Peter Rossmanith, and Xinhui Wang. Dynamic programming for minimum Steiner trees. *Theory of Computing Systems*, 41:493–500, 2007. doi:10.1007/S00224-007-1324-4.

14 Jiong Guo, Falk Hüffner, and Rolf Niedermeier. A structural view on parameterizing problems: Distance from triviality. In *Proc. 1st IWPEC*, pages 162–173. Springer, 2004. doi:10.1007/978-3-540-28639-4_15.

15 Trey Ideker, Owen Ozier, Benno Schwikowski, and Andrew F Siegel. Discovering regulatory and signalling circuits in molecular interaction networks. *Bioinformatics*, 18:S233–S240, 2002. doi:10.1093/bioinformatics/18.suppl_1.S233.

16 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/JCSS.2000.1727.

17 Bart M. P. Jansen, Jari J. H. de Kroon, and Michał Włodarczyk. Vertex deletion parameterized by elimination distance and even less. In *Proc. 53rd STOC*, pages 1757–1769. ACM, 2021. doi:10.1145/3406325.3451068.

18 Bart M. P. Jansen, Jari J. H. de Kroon, and Michal Wlodarczyk. 5-approximation for $\mathcal{H}$-treewidth essentially as fast as $\mathcal{H}$-deletion parameterized by solution size. In *Proc. 31st ESA*, volume 274 of *LIPIcs*, pages 66:1–66:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ESA.2023.66.

19 Richard M. Karp. Reducibility among combinatorial problems. In *Proceedings of a symposium on the Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9.

20 Ton Kloks. *Treewidth: computations and approximations*, volume 842. Springer Science & Business Media, 1994. doi:10.1007/BFb0045375.

21 Thomas Lengauer. *Combinatorial algorithms for integrated circuit layout.* XApplicable Theory in Computer Science. Springer Science & Business Media, 2012. doi:10.1007/978-3-322-92106-2.

22 Ivana Ljubić. Solving Steiner trees: Recent advances, challenges, and perspectives. *Networks*, 77(2):177–204, 2021. doi:10.1002/net.22005.

23 Daniel Lokshtanov and Jesper Nederlof. Saving space by algebraization. In *Proc. 42nd STOC*, pages 321–330. ACM, 2010. doi:10.1145/1806689.1806735.

24 Jesper Nederlof. Fast polynomial-space algorithms using inclusion-exclusion. *Algorithmica*, 65(4):868–884, 2013. doi:10.1007/S00453-012-9630-X.

25 Jaroslav Nešetřil and Patrice Ossona de Mendez. Chapter 6: Bounded height trees and tree-depth. In *Sparsity: Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and Combinatorics*, pages 115–144. Springer, 2012. doi:10.1007/978-3-642-27875-4.

26 PACE 2018, 2018. Accessed 21 Nov 2023. URL: `https://pacechallenge.org/2018/`.

27 Mauricio G.C. Resende and Panos M. Pardalos. *Handbook of optimization in telecommunications*. Springer Science & Business Media, 2008. `doi:10.1007/978-0-387-30165-5`.

28 Olga Russakovsky and Andrew Y. Ng. A Steiner tree approach to efficient object detection. In *Proc. 23rd CVPR*, pages 1070–1077. IEEE Computer Society, 2010. `doi:10.1109/CVPR.2010.5540097`.

29 Tom van Roozendaal. Fixed-parameter tractable algorithms for refined parameterizations of graph problems. Master's thesis, Eindhoven University of Technology, 2023. URL: `https://research.tue.nl/en/studentTheses/fixed-parameter-tractable-algorithms-for-refined-parameterization`.