Proceedings Paper:
Feldmann, A.E. orcid.org/0000-0001-6229-5332 and Lampis, M. orcid.org/0000-0002-5791-0887 (2024) Parameterized algorithms for Steiner forest in bounded width graphs. In: Bringmann, K., Grohe, M., Puppis, G. and Svensson, O., (eds.) 51st International Colloquium on Automata, Languages, and Programming (ICALP 2024). International Colloquium on Automata, Languages, and Programming (ICALP), 08-12 Jul 2024, Tallinn, Estonia. Leibniz International Proceedings in Informatics, LIPIcs, 297 . , 61:1-61:20. ISBN 978-3-95977-322-5

https://doi.org/10.4230/LIPIcs.ICALP.2024.61

Parameterized Algorithms for Steiner Forest in Bounded Width Graphs

Andreas Emil Feldmann
Department of Computer Science, University of Sheffield, UK

Michael Lampis
Université Paris-Dauphine, PSL University, CNRS UMR7243, LAMSADE, Paris, France

Abstract

In this paper we reassess the parameterized complexity and approximability of the well-studied Steiner Forest problem in several graph classes of bounded width. The problem takes an edge-weighted graph and pairs of vertices as input, and the aim is to find a minimum cost subgraph in which each given vertex pair lies in the same connected component. It is known that this problem is APX-hard in general, and NP-hard on graphs of treewidth 3, treedepth 4, and feedback vertex set size 2. However, Bateni, Hajiaghayi and Marx [JACM, 2011] gave an approximation scheme with a runtime of $n^{O(k^2/\varepsilon)}$ on graphs of treewidth k. Our main result is a much faster *efficient parameterized approximation scheme (EPAS)* with a runtime of $2^{O(\frac{k^2}{\varepsilon}\log\frac{k}{\varepsilon})} \cdot n^{O(1)}$. If k instead is the vertex cover number of the input graph, we show how to compute the optimum solution in $2^{O(k \log k)} \cdot n^{O(1)}$ time, and we also prove that this runtime dependence on k is asymptotically best possible, under ETH. Furthermore, if k is the size of a feedback edge set, then we obtain a faster $2^{O(k)} \cdot n^{O(1)}$ time algorithm, which again cannot be improved under ETH.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Approximation algorithms analysis

Keywords and phrases Steiner Forest, Approximation Algorithms, FPT algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.61

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2402.09835`

Funding *Michael Lampis*: Supported by ANR project ANR-21-CE48-0022 (S-EX-AP-PE-AL).

1 Introduction

The Steiner Forest problem is one of the most well-studied problems in network design [16, 23, 24, 28]. In this problem the input consists of a graph $G = (V, E)$ with positive edge weights, a set of *terminals* $R \subseteq V$, and a set of *demands* $D \subseteq \binom{R}{2}$. The objective is to select a subgraph $F \subseteq G$, minimizing the total cost of selected edges, while ensuring that for every demand pair $\{s, t\} \in D$, s and t are in the same connected component of F. Since edge weights are positive, it is easy to see that the optimal solution is always a forest. The Steiner Forest problem finds many applications (see surveys [11, 28, 32, 33]), for example in telecommunication networks (cf. [33]).

Our goal in this paper is to reassess the complexity of this fundamental problem from the point of view of parameterized complexity and approximation algorithms.[1] In order to recall the context, it is helpful to compare Steiner Forest to the even more well-studied Steiner

[1] We assume the reader is familiar with the basics of parameterized complexity and approximation algorithms, such as the classes FPT and APX and the definition of treewidth, as given in standard textbooks [14, 19, 34]. We give full definitions of all parameters in Section 2.

Tree problem, which is the special case of Steiner Forest where all terminals are required to be connected, i.e., $D = \binom{R}{2}$, and an optimal solution is a tree. Steiner Tree was already included in Karp's seminal list [25] of NP-hard problems from the 1970s. From the approximation point of view, Steiner Tree (and therefore Steiner Forest) is known to be APX-hard [13], but both problems admit constant factor approximations in polynomial time for general input graphs, where the best approximation factors known are $\ln(4)+\varepsilon < 1.39$ [10] and 2 [1, 31], respectively. Despite this similarity, when considering graph width parameters the problems exhibit wildly divergent behaviors from the parameterized complexity point of view: whereas Steiner Tree is FPT parameterized by standard structural parameters such as treewidth and can in fact even be solved in single exponential $2^{O(k)}n^{O(1)}$ time [6] when k is the treewidth, Steiner Forest is NP-hard on graphs of treewidth 3, as shown independently by Gassner [21] and Bateni, Hajiaghayi, and Marx [4].

Steiner Forest is therefore a problem that presents a dramatic jump in complexity in this context, compared to Steiner Tree, as the hardness result on graphs of treewidth 3 rules out even an XP algorithm for parameter treewidth. One of the main positive contributions of Bateni, Hajiaghayi, and Marx [4] was an algorithm attempting to bridge this gap using approximation. In particular, they showed that Steiner Forest admits an approximation scheme for graphs of treewidth k, which computes a $(1+\varepsilon)$-approximation in $n^{O(k^2/\varepsilon)}$ time for any $\varepsilon > 0$. Hence, if we allow slightly sub-optimal solutions, we can at least place the problem in XP parameterized by treewidth. In their paper, Bateni, Hajiaghayi, and Marx [4] remark that because the exponent of the polynomial of this runtime depends on k and ε, "it remains an interesting question for future research whether this dependence can be removed", that is, whether a $(1+\varepsilon)$-approximation can be obtained in FPT time.

The main result of our paper is a positive resolution of the question of [4]: we show that Steiner Forest admits an *efficient parameterized approximation scheme (EPAS)* for treewidth, that is, a $(1+\varepsilon)$-approximation algorithm with a runtime of the form $f(k,\varepsilon)n^{O(1)}$. In other words, we show that their algorithm can be improved in a way that makes the running time FPT not only in the treewidth, but also in $1/\varepsilon$. More precisely, we show the following:

▶ **Theorem 1.** *The Steiner Forest problem admits an EPAS parameterized by the treewidth k with a runtime of $2^{O(\frac{k^2}{\varepsilon}\log\frac{k}{\varepsilon})} \cdot n^{O(1)}$.*

Moving on from treewidth, we ask what the most general parameter is for which we may hope to obtain an FPT *exact* algorithm for Steiner Forest. We observe that the NP-hardness result of [4, 21] for Steiner Forest on graphs of treewidth 3 actually has some further implications for some even more restricted parameters: the graphs constructed in their reductions also have constant *treedepth* and *feedback vertex set* size, implying that the problem remains hard for both of these parameters (which are incomparable in general). More precisely, known reductions imply the following:

▶ **Theorem 2** ([4, 21])**.** *The Steiner Forest problem is NP-hard on graphs of treewidth* 3*, treedepth* 4*, and feedback vertex set of size* 2*.*

This leads us to consider even more restricted parameters, such as the size of a *vertex cover* and *feedback edge set*, which are not bounded in this reduction. Indeed, not only do we prove that Steiner Forest is FPT for both of these parameters, but we are also able to determine the correct parameter dependence, under the Exponential Time Hypothesis (ETH). For feedback edge set the optimal dependence is single exponential:

▶ **Theorem 3.** *The* Steiner Forest *problem is FPT parameterized by the size k of a feedback edge set and can be solved in $2^{O(k)}n^{O(1)}$ time. Furthermore, no $2^{o(k)}n^{O(1)}$ time algorithm exists, under ETH.*

For the parameterization by the vertex cover size, we obtain a slower runtime for our FPT algorithm. Interestingly, we are also able to prove that this is best possible, under ETH. Our lower bound for Steiner Forest is in contrast to the Steiner Tree problem, for which a faster $2^{O(k)}n^{O(1)}$ time algorithm exists, even if k is the treewidth [6].

▶ **Theorem 4.** *The* Steiner Forest *problem is FPT parameterized by the size k of a vertex cover and can be solved in $2^{O(k \log k)}n^{O(1)}$ time. Furthermore, no $2^{o(k \log k)}n^{O(1)}$ time algorithm exists, under ETH.*

We remark that Bodlaender et al. [8] recently independently showed that Steiner Forest admits a $2^{O(k \log k)}n^{O(1)}$ time algorithm for the size k of a vertex cover (improving an algorithm for the unweighted version of the problem given in [22]). While they develop their own dynamic program to solve this problem, we rely on an existing algorithm by [4] (see Theorem 5). Accordingly our description of the algorithm is very short compared to [8]. The more interesting part of Theorem 4 however is the proof of the lower bound.

1.1 Overview of Techniques

Let us briefly sketch the high level ideas of our results given by Theorems 1, 3, and 4.

EPAS for treewidth. Our algorithm extends the work of [4], so let us briefly recall some key ideas. Given a rooted tree decomposition, a terminal t is called *active* for a bag B if there is a demand $\{s, t\} \in D$ such that t lies in the sub-tree rooted at B while s does not (see Section 2 for formal definitions). It is a standard property of tree decompositions that every bag is a separator. Hence the component of any feasible solution that contains an active terminal must intersect B. The hardness of the problem now inherently stems from the fact that we have to decide for all active terminals of a bag, how the corresponding component intersects the bag, and therefore how the active terminals (whose number is unbounded by k) are partitioned into connected components. Suppose, however, that someone supplied us with this information, that is, suppose that for each bag B we are given a set of partitions Π_B of its active terminals and we are promised that the optimal solution *conforms* to all Π_B. By this we mean that if we look at how the optimal solution partitions the active terminals of B into connected components and call this partition π, then $\pi \in \Pi_B$, that is, the optimal partition is always one of the supplied options. In this case, using this extra information, the problem does become tractable, as shown in [4]:

▶ **Theorem 5** ([4])**.** *For an input graph G on n vertices, let a rooted nice tree decomposition of width k be given, such that all terminals lie in bags of leaf nodes of the decomposition. Also, let a set Π_B of partitions of the active terminals of each bag B of the decomposition be given. If $p = \sum_B |\Pi_B|$ is the total number of partitions, then a minimum cost* Steiner Forest *solution conforming to all Π_B can be computed in $2^{O(k \log k)} \cdot (pn)^{O(1)}$ time.*

The above theorem does not seem immediately helpful, since one would still need to find a small collection of partitions Π_B in order to obtain an efficient algorithm. Note however, that the partition sets may conform to an approximate solution as well, which would let the algorithm compute a solution that is at least as good. The strategy of [4] therefore is to construct a collection of partitions that has size polynomial in n (when k, ε are fixed

constants) by stipulating that when two active terminals are "close" to each other, they should belong in the same set of the partition of some near-optimal solution. In order to bound the resulting approximation ratio, they need to provide a charging scheme: starting from an optimal solution, they merge components which are "close", to obtain a solution that conforms to the Π_B used by the algorithm. They then show that the resulting solution is still near-optimal by charging the extra cost incurred by a merging operation to one of the two merged components.

A blocking point in the above is that we need to make sure we do not "overcharge" any component. This is accomplished in [4] via a partial ordering of the components: we order the components according to the highest bag of the rooted tree decomposition they intersect, and whenever two components are merged we charge this to the *lower* component. As shown in [4], this ensures that no component is charged for more than k merges. Unfortunately, this also implies that the merging procedure is not symmetric, which severely diminishes the contexts in which we can apply it.

Let us now describe how our approach improves upon this algorithm. A key ingredient will be a more sophisticated charging scheme, which will allow us to obtain a better (smaller) collection of partitions Π_B, without sacrificing solution quality. Counter-intuitively, we will achieve this by introducing a *second* parameter: the height h of the tree decomposition. Informally, we will now construct a near-optimal solution by merging two components whenever the connection cost is low compared to the cost of (a part of) *either* component (as opposed to the lower component). As in [4], this runs the risk of charging many merging operations to a higher component, but by performing an accounting by tree decomposition level and using the fact that the decomposition only has h levels, we are able to show that our solution is still near-optimal even though we merge components much more aggressively than [4]. In this way, for each bag we construct one partition of its active terminals into a number of sets that is polynomial in $k + h + \frac{1}{\varepsilon} + \log n$, in a way that guarantees that this partition is a *refinement* of a near-optimal solution. That is, whenever we decide to place two terminals together in our partition, the near-optimal solution does the same. However, this solution does not necessarily conform to the resulting partitions, as two terminals of the same component might end up in different sets of the partition for a bag.

At this point an astute reader may be wondering that since we consider both the width k and the height h of the decomposition as parameters, we are effectively parameterizing by treedepth, rather than treewidth. This is correct, but we then go on to invoke a result of [7] which states that any tree decomposition can be rebalanced to have height $O(\log n)$ without severely increasing its width. Hence, the family of partitions we now have has size polynomial in $k + \frac{1}{\varepsilon} + \log n$. However, we are not done yet, since at this point we can only guarantee that our partitions are refinements of a near-optimal solution. To complete the algorithm, we work from this family of partitions to obtain a collection of partitions *conforming* to our near-optimal solutions using δ-nets (this is similar to the approach of [4]). This leads to a running time of the form $(\log n)^{O(\frac{k^2}{\varepsilon})} n^{O(1)}$, which by standard arguments of parameterized complexity is in fact FPT and can be upper-bounded by a function of the form $2^{O(\frac{k^2}{\varepsilon} \log \frac{k}{\varepsilon})} \cdot n^{O(1)}$. To summarize, our high-level strategy is to show that the approach of [4] can be significantly improved when the input decomposition has small width and height, but then we observe that our new scheme is efficient enough in the height that even if we replace h by a bound that can be obtained for *any* graph, we still have an algorithm with an FPT running time, that is, significantly faster than that of [4].

Vertex Cover. For the parameterization by the vertex cover size, as mentioned we obtain an FPT exact algorithm with dependence $2^{O(k \log k)}$. A similar algorithm was recently independently obtained by [8] via dynamic programming. However, our algorithm is significantly simpler, because our strategy is to show how to construct a tree decomposition and a collection of partitions Π_B such that we only need *one* partition of the active terminals for each bag. As a consequence, $p = O(n)$ and Theorem 5 implies the algorithm of Theorem 4, without the need to formulate a new dynamic program.

Our main result for this parameter is that under ETH the runtime dependence is asymptotically optimal. Note that this also implies that the runtime of the dynamic program given by Theorem 5 cannot be improved with regards to the dependence on the treewidth. To show this, we present a reduction from 3-SAT, where the goal is to compress an n-variable formula into a Steiner Forest instance such that the graph has vertex cover size $O(n/\log n)$. The intuition on why it is possible to achieve such a compression is the following: suppose we have an instance with vertex cover of size k and a demand between two vertices of the independent set. Then the simplest way to satisfy such a demand is to connect both vertices to a common neighbor in the vertex cover. This encodes a choice among k vertices, and hence it is sufficient to encode the assignment for $\log k$ binary variables. The strategy of our reduction is to set up some choice gadgets which allow us to encode the assignments to the original formula taking advantage of the fact that each choice can represent a logarithmic number of variables. Hence we can obtain a construction of slightly sub-linear ($O(n/\log n)$) size. We then of course need to add some verification gadgets, representing the clauses, to check that the formula is indeed satisfied. But even though the number of such gadgets is linear in n, we make sure that they form an independent set, and hence the total vertex cover size remains sufficiently small to obtain our lower bound. We note that this compression strategy is similar to techniques recently used to obtain slightly super-exponential lower bounds for vertex cover for other problems [26, 27], but the constructions we use are new and tailored to Steiner Forest.

Feedback Edge Set. For the parameterization by the size k of a feedback edge set, instead of relying on the dynamic program given by Theorem 5 we go an entirely different route in order to obtain the faster $2^{O(k)}n^{O(1)}$ time FPT algorithm of Theorem 3. First off, it is not hard to reduce the Steiner Forest problem to an instance in which all vertices have degree at least 2. We then consider paths with internal vertices of degree 2, with endpoints that are vertices incident to the feedback edge set or vertices of degree at least 3. We call these paths *topo-edges* and argue that there are only $O(k)$ of these. We then guess for which topo-edges the two endpoints lie in different components of the optimal Steiner Forest solution, which can be done in $2^{O(k)}$ time. If a topo-edge has both its endpoints in the same component of the optimum, we show that it can be easily handled. For the remaining topo-edges, we can decide which edges along the path do not belong to the optimal solution by a reduction to the polynomial-time solvable Min Cut problem.

1.2 Related work

Bateni, Hajiaghayi, and Marx [4] show that one of the consequences of their XP approximation scheme is a PTAS for Steiner Forest on planar graphs, by using the common technique pioneered by Baker [2] of reducing this problem to graphs for which the treewidth is bounded as a function of ε. Because their algorithm is not FPT, their PTAS has a running time of the form $n^{f(\varepsilon)}$. By using our algorithm from Theorem 1 we can improve this runtime to $f(\varepsilon)n^{O(1)}$, i.e., we obtain on EPTAS for planar graphs. However, [18] already showed that

a $(1+\varepsilon)$-approximation algorithm with a runtime of $O(f(\varepsilon) \cdot n \log^3 n)$ exists for Steiner Forest on planar graphs. While they build on the work of [4], and in particular also reduce to graphs of treewidth bounded as a function of ε, interestingly they do not obtain an EPAS parameterized by treewidth. Instead they use a different route and show that given a graph H of treewidth k, in $O(f(k,\varepsilon) \cdot n \log^2 n)$ time it is possible to compute a Steiner Forest solution in H whose cost is at most $\text{cost}(F^\star) + \varepsilon \,\text{cost}(H)$, i.e., there is an additive error that depends on the cost of H compared to the optimum solution $F^\star$. If the input graph G is planar, then a result by [9] implies that from G a so-called *banyan* [3, 30] can be computed, which is a subgraph of G with cost bounded by $O(g(\varepsilon)\,\text{cost}(F^\star))$, and which contains a near-optimal approximation of every Steiner forest (cf. [18, Lemma 2.1]). By applying the framework of [4] on the banyan instead of the input graph, it is then possible to obtain a graph H of treewidth bounded by a function of ε, for which the algorithm of [18] computes a $(1+O(\varepsilon))$-approximation for the input.

If it would be possible to compute a banyan for bounded treewidth graphs, then the algorithm of [18] would also imply an EPAS for treewidth. However, to the best of our knowledge, and as explicitly stated by [3], banyans are only known for planar graphs [9, 18], Euclidean metrics [30], and doubling metrics [3] (in fact, the latter are so-called *forest banyans*, which have weaker properties). Thus it is unclear how to obtain an EPAS for Steiner Forest parameterized by the treewidth via the algorithm of [18]. We leave open whether a banyan exists for bounded treewidth graphs, which could give an alternative algorithm to the one given in Theorem 1. However, a further remark is that the cost of the banyan for planar graphs obtained by [9] has exponential dependence on $1/\varepsilon$, which implies a double exponential runtime dependence on $1/\varepsilon$ for the EPTAS for planar graphs. If a banyan can be obtained for bounded treewidth graphs by generalizing the techniques of [9] to minor-free graphs, then the resulting EPAS parameterized by treewidth would also have double exponential runtime in $1/\varepsilon$. In this case however, our EPAS given by Theorem 1 would be exponentially faster.

A different parameter that is often studied in the context of Steiner problems is the number $p = |R|$ of terminals. The classic result of [15] presents an FPT algorithm for Steiner Tree with a runtime of $3^p n^{O(1)}$. For unweighted graphs, this was improved [5, 29] to $2^p n^{O(1)}$, while the fastest known algorithm for weighted graphs can compute the optimum in $(2+\varepsilon)^p n^{O(\sqrt{\frac{1}{\varepsilon}\log\frac{1}{\varepsilon}})}$ time [20] for any $\varepsilon > 0$. The algorithm of [15] can be generalized to solve Steiner Forest in $2^{O(p)} n^{O(1)}$ time (cf. [12]). A somewhat dual parameter to the number of terminals is the number q of non-terminals (so-called *Steiner vertices*) in the optimum solution. For this parameter, a folklore result states that Steiner Tree (and thus also Steiner Forest) is W[2]-hard (cf. [14, 17]). However, an EPAS with a runtime of $2^{O(q^2/\varepsilon^4)} n^{O(1)}$ was shown to exist for Steiner Tree [17]. For Steiner Forest it is not hard to see that such an EPAS parameterized by q cannot exist unless P=NP (cf. [17]), but if c denotes the number of components of the optimum solution, there is an EPAS with a runtime of $(2c)^{O((q+c)^2/\varepsilon^4)} n^{O(1)}$ [17]. Similar results have been found for related Steiner problems in directed graphs [12]. For further results in the area of parameterized approximations, we refer to the survey in [19].

2 Preliminaries

As mentioned, we assume the reader is familiar with the basics of parameterized complexity, such as the class FPT [14], and approximation algorithms such as a PTAS [34]. A *parameterized approximation scheme (PAS)* is an algorithm that computes a $(1+\varepsilon)$-approximation

for a problem in $f(k,\varepsilon)n^{g(\varepsilon)}$ time for some functions f and g, while an an *efficient parameterized approximation scheme (EPAS)* is a $(1+\varepsilon)$-approximation algorithm running in time $f(k,\varepsilon)n^{O(1)}$ (that is, the running time is FPT in $k+\frac{1}{\varepsilon}$). The distinction between a PAS and an EPAS is similar to the one between a PTAS and an EPTAS.

By $w : E \to \mathbb{R}^+$ we denote an edge-weight function, so that the cost of a solution F to the Steiner Forest problem is $\text{cost}(F) = \sum_{e\in E(F)} w(e)$. We will use $F^\star$ to denote an optimal solution, and for $\alpha \geq 1$ we will say that a solution F is α-approximate if $\text{cost}(F) \leq \alpha\,\text{cost}(F^\star)$. For $u, v \in V$ we use $\text{dist}(u,v)$ to denote the shortest-path distance from u to v in G according to the weight function w.

▶ **Definition 6.** *Given a graph $G = (V,E)$, a* tree decomposition *is a pair $(T, \{B_i\}_{i\in V(T)})$, where T is a tree and each node $i \in V(T)$ of the tree is associated with a* bag *$B_i \subseteq V$, with the following properties:*

1. *$\bigcup_{i\in V(T)} B_i = V$, i.e., all vertices of G are covered by the bags,*
2. *for every edge $uv \in E$ of G there exists a node $i \in V(T)$ of the tree for which $u, v \in B_i$, and*
3. *for every vertex $v \in V$ of G the nodes $\{i \in V(T) \mid v \in B_i\}$ of the tree for which the bags contain v induce a (connected) subtree of T.*

The width *of the tree decomposition is $\max_{i\in V(T)}\{|B_i| - 1\}$ and the* treewidth *of G is the minimum width over all its tree decompositions.*

A rooted tree decomposition is nice *if for every $i \in V(T)$ we have one of the following:*

1. *i has no children*[2] *(i is a* leaf node*),*
2. *i has exactly two children i_1 and i_2 such that $B_i = B_{i_1} = B_{i_2}$ (i is a* join node*),*
3. *i has a single child i' where $B_i = B_{i'} \cup \{v\}$ for some $v \in V$ (i is an* introduce node*), or*
4. *i has a single child i' where $B_i = B_{i'} \setminus \{v\}$ for some $v \in V$ (i is a* forget node*).*

Given a rooted tree decomposition T of a graph G, for a node u of T let B be the bag associated with it. Then V_B is the set of vertices of all bags in the subtree rooted at u. The set $A_B \subseteq R$ denotes the *active terminals* of the bag B: for any demand pair $\{s,t\} \in D$, if $s \in V_B$ and $t \notin V_B$ then $s \in A_B$. For any Steiner Forest solution F, if a connected component C of F contains an active terminal, then we say that C is an *active component* for B. For a fixed solution F, we denote the set of all active components for B by $\mathcal{C}_B$. Note that every active component must intersect the bag B.

If for every bag B a set of partitions Π_B of A_B is given, a Steiner Forest solution F is *conforming* to all Π_B, if for each bag B there exists a partition $\pi \in \Pi_B$ such that any two active terminals in A_B are in the same set $S \in \pi$ if and only if they are part of the same active component C of F, i.e., $S \subseteq V(C)$ and $S' \cap V(C) = \emptyset$ for any $S' \in \pi$ with $S' \neq S$ (note that this implies $|\pi| \leq |B|$). One technicality of Theorem 5 is that the algorithm needs a nice tree decomposition as input, for which the terminals only appear in bags that are leaf nodes of the decomposition. Given any tree decomposition, these conditions are not hard to meet (cf. [4, Lemma 6]). However, for our algorithms we are going to rely on tree decompositions with certain additional properties. Hence we will need to revisit the conditions needed for the algorithm of Theorem 5 when using it for our purposes.

We will also consider the following parameters: The *treedepth* of a graph G can be defined recursively as follows: (i) the treedepth of K_1 is 1 (ii) the treedepth of a disconnected graph is the maximum of the treedepth of any of its components (iii) the treedepth of a connected graph G is $1 + \min_{v\in V(G)} \text{td}(G - v)$. A *feedback vertex set* is a set of vertices whose removal

[2] here we do not demand the leaf nodes to be empty, as is often assumed for this definition.

leaves a forest. A *vertex cover* is a set of vertices such that its removal leaves an edge-less graph. A *feedback edge set* is a set of edges whose removal leaves a forest. In a connected graph with n vertices and m edges the minimum feedback edge set always has size $m - n + 1$.

As part of our approximation algorithm we will use the notion of δ-nets, defined as follows. A well-known fact is that a δ-net exists for any metric and any $\delta \geq 0$, and it can be constructed greedily in polynomial time.

▶ **Definition 7.** *Given a metric* (X, dist)*, a* δ-net *is a subset* $N \subseteq X$ *of points, such that*
1. *any two net points* $u, v \in N$ *are far from one another, i.e.,* $\mathrm{dist}(u, v) > \delta$*, and*
2. *for any node* $u \in X$ *there is some net point* $v \in N$ *close by, i.e.,* $\mathrm{dist}(u, v) \leq \delta$*.*

3 An efficient parameterized approximation scheme for treewidth

In this section we describe the main result of this paper which is an EPAS for Steiner Forest parameterized by treewidth. We begin by giving two preliminary tools (Lemma 8 and Lemma 9) which facilitate the algorithm by ensuring that the given tree decomposition has logarithmic height and that the instance has aspect ratio (ratio of the weights of the heaviest over the lightest edge) bounded by a polynomial in n.

We then go on to Subsection 3.1 where we introduce a second parameter, the height h of the decomposition. Our goal is to fix an almost-optimal solution F_ε and describe an algorithm that produces a partition ζ_B of the active terminals for each bag B of the decomposition, where ζ_B is *a refinement* of the partition implied by F_ε (Lemma 10). In other words, we seek a partition ζ_B of A_B such that if two terminals t_1, t_2 are in the same set of ζ_B, then they are also in the same component of F_ε. Of course, it is trivial to achieve this by giving a ζ_B where each active terminal is in its own set, so the interesting part here is how we group terminals together in a way that in the end allows us to bound $|\zeta_B|$ by a polynomial of $k + h + \frac{1}{\varepsilon} + \log n$, while still ensuring that F_ε is almost optimal.

The partition ζ_B of Lemma 10 is not yet conforming, because two terminals which are in distinct sets of ζ_B may still be in the same component of F_ε, and thus we cannot apply Theorem 5 at this point. Therefore in Subsection 3.2, given ζ_B we focus on how to obtain every possible partition of the set of active terminals, which could be conforming with an almost-optimal solution. By an appropriate use of δ-nets, similar to [4], we are able to "guess" (that is, brute-force) a choice of a small number of net points per active component. Since the number of choices for each point is at most $|\zeta_B|$ and we choose roughly $O(k^2/\epsilon)$ points in total, the total number of produced partitions (and hence the running time given by Theorem 5) is of the form $(\log n + k + \frac{1}{\epsilon})^{O(k^2/\epsilon)} n^{O(1)}$, which is FPT.

Let us now recall a result of Bodlaender and Hagerup [7] which states that a tree decomposition of logarithmic height can always be obtained.

▶ **Lemma 8** ([7])**.** *Given a tree decomposition of width* k *of a graph* G *on* n *vertices, there is a polynomial time algorithm computing a nice tree decomposition of* G *of width* $O(k)$ *and height* $O(k \log n)$*.*

We also need to reduce the aspect ratio of the given graph to a polynomial. This can be done using a standard technique, where however we need to make sure that the treewidth of the given graph remains bounded. Note that the aspect ratio of the resulting graph G' in the following lemma is polynomially bounded in the size of the original graph, but not necessarily in the size of G' (because G' may have significantly fewer vertices).

▶ **Lemma 9.** *Given* $\varepsilon > 0$*, an instance of* Steiner Forest *on a graph* G *with* n *vertices, and a (nice) tree decomposition* T *of width* k *and height* h *for* G*, in polynomial time we can compute an instance on a graph* G' *with at most* n *vertices and a (nice) tree decomposition* T'

of width at most k and height h for G', such that the ratio of the longest to the shortest edge in G' is at most $2n/\varepsilon$, and any α-approximation for G' can be converted into an $(\alpha+\varepsilon)$-approximation for G.

For simplicity, in the following we will scale the edge lengths of any given graph so that the shortest edge has length 1. In particular, after applying Lemma 9, the longest edge has length at most $2n/\varepsilon$.

3.1 Tree decompositions with bounded height

In this section we informally assume that the height h of the given tree decomposition is bounded as well as the width k. Our aim is to prove the following statement, where we restrict ourselves to input graphs of polynomial aspect ratio, which we may do according to Lemma 9 (keeping in mind that n is the number of vertices of the original input graph).

▶ **Lemma 10.** *Let an instance of* Steiner Forest *on a graph G with at most n vertices be given together with a tree decomposition T of width k and height h for G. For any $\varepsilon > 0$, if the ratio between the longest and shortest edge of G is at most $2n/\varepsilon$, then there exists a $(1+\varepsilon)$-approximation F_ε with the following properties. There exists a polynomial time algorithm, which for every bag B of T outputs a partition ζ_B of the active terminals A_B, such that each set of ζ_B belongs to the same component of F_ε and $|\zeta_B| = O(\frac{k^4h^2}{\varepsilon^2}\log\frac{n}{\varepsilon})$.*

To prove Lemma 10 we first identify the solution F_ε, after which we will show how to compute the partitions ζ_B.

3.1.1 A near-optimal solution

The high-level idea to obtain a $(1+\varepsilon)$-approximate solution F_ε is to connect components of the optimum solution $F^\star$ that lie very close to each other. In particular, if the distance between two components C and C' of $F^\star$ is of the form $f(k,h,\varepsilon)\,\mathrm{cost}(C)$ for some small enough function f, then we may hope to add a shortest path between C and C' and charge this additional cost to C, in order to obtain a $(1+\varepsilon)$-approximation. Unfortunately, this approach is not viable, since the number of components that are very close to C may be very large, meaning that the function f in the distance bound would have to linearly depend on the number of vertices in order to result in a $(1+\varepsilon)$-approximation. This in turn would mean that the size of the partition ζ_B would depend polynomially on the number of vertices, making it unsuitable for an FPT time algorithm. This issue lies at the heart of the problem and is the reason for why it is non-trivial to obtain an approximation scheme parameterized by the treewidth. To get around this issue we will measure the distance between components using a modified cost function, which we define next.

Given a bag B of the rooted tree decomposition T, we denote by T_B the subtree of T rooted at the node associated with B, and by $G_B = G[V_B]$ the graph induced by the vertices V_B lying in bags of T_B. We also define the graph $G_B^\downarrow \subseteq G_B$ as the graph spanned by all edges of G_B, except those induced by B, i.e., the edge set of $G_B^\downarrow$ is

$$E(G_B^\downarrow) = \{uv \in E(G_B) \mid u \notin B \vee v \notin B\}.$$

The cost of a component C of some Steiner Forest solution restricted to $G_B^\downarrow$ only counts the edge weights of C in $G_B^\downarrow$, and is denoted by

$$\mathrm{cost}_B^\downarrow(C) = \sum_{e \in E(C) \cap E(G_B^\downarrow)} w(e).$$

Based on these definitions, we fix an optimal solution $F^\star$ and construct a solution F_ε by initially setting $F_\varepsilon = F^\star$, and then connecting components by exhaustively applying the following rule, where we say that two components C and C' *share* a bag B if $V(C) \cap B \neq \emptyset$ and $V(C') \cap B \neq \emptyset$:

▶ **Rule 1.** *If C, C' are components of $F^\star$ sharing a bag B with $\operatorname{dist}(C, C') \leq \frac{\varepsilon}{kh} \cdot \operatorname{cost}^\downarrow_B(C)$ but C and C' are in different components of F_ε, then add a shortest path of length $\operatorname{dist}(C, C')$ between C and C' to the solution F_ε.*

▶ **Lemma 11.** *The cost of the solution F_ε obtained by Rule 1 from $F^\star$ is at most $(1+\varepsilon)\operatorname{cost}(F^\star)$.*

Proof. It suffices to prove that the cost of all paths added to $F^\star$ in order to obtain F_ε according to Rule 1 is at most $\varepsilon \cdot \operatorname{cost}(F^\star)$. For this we use a charging scheme that charges new paths to components of $F^\star$. In particular, we charge a path of length $\operatorname{dist}(C, C') \leq \frac{\varepsilon}{kh} \cdot \operatorname{cost}^\downarrow_B(C)$ to component C.

Fix a component C of $F^\star$ and a bag B with $V(C) \cap B \neq \emptyset$. We define $\operatorname{charge}(C, B)$ to be the cost we charge to C for operations involving other components of $F^\star$ that share B. It is not hard to see that $\operatorname{charge}(C, B) \leq \frac{\varepsilon}{h} \cdot \operatorname{cost}^\downarrow_B(C)$, because there are at most k other components of $F^\star$ that share B.

For $\ell \in \{0, \ldots, h-1\}$, let $\mathcal{B}_\ell$ be the set of bags of the tree decomposition that appear at distance exactly ℓ from the root, i.e., they lie on level ℓ of the tree. We now observe that

$$\sum_{B \in \mathcal{B}_\ell} \operatorname{charge}(C, B) \leq \sum_{B \in \mathcal{B}_\ell} \frac{\varepsilon}{h} \cdot \operatorname{cost}^\downarrow_B(C) \leq \frac{\varepsilon}{h} \operatorname{cost}(C),$$

where the last inequality follows because if we have two bags $B, B' \in \mathcal{B}_\ell$, then $E(G^\downarrow_B) \cap E(G^\downarrow_{B'}) = \emptyset$: note that every edge of $E(G^\downarrow_B)$ must be incident on a vertex v that appears in a descendant of B, but not in B. By the properties of tree decompositions, notably by the fact that B is a separator of G, v cannot appear in B' or any of its descendants. Therefore none of its incident edges are contained in $E(G^\downarrow_{B'})$. Because $\sum_{B \in \mathcal{B}_\ell} \operatorname{cost}^\downarrow_B(C)$ is the sum of costs of C over disjoint sets of edges, the sum is a lower bound on the total cost of C.

To conclude, we observe that the total charge of C is

$$\operatorname{charge}(C) \leq \sum_{\ell=0}^{h-1} \sum_{B \in \mathcal{B}_\ell} \operatorname{charge}(C, B) \leq \varepsilon \operatorname{cost}(C).$$

Therefore, summing over all components of $F^\star$, the total cost of the edges we have added according to Rule 1 is at most $\varepsilon \cdot \operatorname{cost}(F^\star)$. ◀

3.1.2 Partitioning active terminals

We are now ready to prove Lemma 10 for the near-optimal solution F_ε constructed above, for which we will compute the partitions ζ_B for all bags B. We will use the following two claims for the active terminals A_B of the given bag B.

▷ Claim 12. If there exist $t_1, t_2 \in A_B$ such that $\operatorname{dist}(t_1, t_2) \leq \frac{\varepsilon}{kh} \operatorname{dist}(t_1, B)$, then t_1, t_2 are in the same component of F_ε.

▷ Claim 13. Let $A \subseteq A_B$ and $d \geq 0$ be such that (i) there exists $b \in B$ such that for all $t \in A$ we have $\operatorname{dist}(t, B) = \operatorname{dist}(t, b)$ and $d \leq \operatorname{dist}(t, B) \leq 2d$ (ii) for all distinct $t, t' \in A$ we have $\operatorname{dist}(t, t') > \frac{\varepsilon}{kh} d$ (iii) $|A| \geq \frac{8k^2(k+1)h^2}{\varepsilon^2}$. Then, there exists a component of F_ε that contains all terminals of A.

Intuitively, Claim 12 allows us to place terminals of A which are very close to each other into the same set of the partition ζ_B, as placing one terminal in a component forces the placement of the other. Thanks to this claim we can work with an appropriate net. If we find a large collection of such net points which also are roughly the same distance from the bag and closest to the same vertex of the bag, Claim 13 allows us to group them all together in the partition ζ_B. Armed with these tools, we can now prove the main lemma.

Proof of Lemma 10. To compute the partition ζ_B in polynomial time, we first partition the active terminals $A_B \cap B$ contained in the bag B. For this we simply add a set $\{t\}$ for each $t \in A_B \cap B$ to ζ_B, which adds at most $|B| \leq k+1$ sets to ζ_B. Let now $A = A_B \setminus B$ be the remaining active terminals.

To partition A, let $d = \min_{t \in A_B \setminus B} \operatorname{dist}(t, B)$ and $D = \max_{t \in A_B \setminus B} \operatorname{dist}(t, B)$ be the minimum and maximum distances of these active terminals from the bag B. Then partition $A_B \setminus B$ into $|B| \leq k+1$ sets $A_1, A_2, \ldots, A_{|B|}$, depending on the vertex of B that is closest to each $t \in A$ (breaking ties arbitrarily). That is, for each A_i there exists $b \in B$ such that for all $t \in A_i$ we have $\operatorname{dist}(t, B) = \operatorname{dist}(t, b)$. Consider now a set A_i and further partition it into $r = \lceil \log_2 \frac{D}{d} \rceil$ sets $A_{i,0}, A_{i,1}, \ldots, A_{i,r-1}$, where $A_{i,j}$ contains all $t \in A_i$ such that $\operatorname{dist}(t, B) \in [2^j d, 2^{j+1} d)$. Now (greedily) compute an $(\frac{\varepsilon}{kh} 2^j d)$-net $N_{i,j}$ of $A_{i,j}$. We observe that $N_{i,j}$ satisfies the first two conditions of Claim 13 for $2^j d$, so if $|N_{i,j}| \geq \frac{8k^2(k+1)h^2}{\varepsilon^2}$, then we add $A_{i,j}$ as a set of our partition ζ_B, remove the terminals of $N_{i,j}$ from A and continue the algorithm for the remaining terminals. Repeat the previous step for all i, j for which $N_{i,j}$ is sufficiently large. This contributes at most $(k+1)\lceil \log \frac{D}{d} \rceil$ sets to ζ_B. To partition A, let $d = \min_{t \in A_B \setminus B} \operatorname{dist}(t, B)$ and $D = \max_{t \in A_B \setminus B} \operatorname{dist}(t, B)$ be the minimum and maximum distances of these active terminals from the bag B. Then partition $A_B \setminus B$ into $|B| \leq k+1$ sets $A_1, A_2, \ldots, A_{|B|}$, depending on the vertex of B that is closest to each $t \in A$ (breaking ties arbitrarily). That is, for each A_i there exists $b \in B$ such that for all $t \in A_i$ we have $\operatorname{dist}(t, B) = \operatorname{dist}(t, b)$. Consider now a set A_i and further partition it into $r = \lceil \log_2 \frac{D}{d} \rceil$ sets $A_{i,0}, A_{i,1}, \ldots, A_{i,r-1}$, where $A_{i,j}$ contains all $t \in A_i$ such that $\operatorname{dist}(t, B) \in [2^j d, 2^{j+1} d)$. Now (greedily) compute an $(\frac{\varepsilon}{kh} 2^j d)$-net $N_{i,j}$ of $A_{i,j}$. We observe that $N_{i,j}$ satisfies the first two conditions of Claim 13 for $2^j d$, so if $|N_{i,j}| \geq \frac{8k^2(k+1)h^2}{\varepsilon^2}$, then we add $A_{i,j}$ as a set of our partition ζ_B, remove the terminals of $A_{i,j}$ from A and continue the algorithm for the remaining terminals. Repeat the previous step for all i, j for which $N_{i,j}$ is sufficiently large. This contributes at most $(k+1)\lceil \log \frac{D}{d} \rceil$ sets to ζ_B.

Suppose now that we are left with a set of terminals A such that the procedure above fails to construct a sufficiently large net $N_{i,j}$ to apply Claim 13. For every index pair i, j, each remaining terminal $t \in A_{i,j}$ is close enough to some net point $t' \in N_{i,j}$ such that we can apply Claim 12. We therefore create a set in the partition ζ_B for each $t' \in N_{i,j}$, placing into such a set those terminals of $A_{i,j}$ that are closest to t' (breaking ties arbitrarily). Since we cannot apply Claim 13 to the remaining sets $A_{i,j}$, each of the at most $(k+1)\lceil \log \frac{D}{d} \rceil$ nets $N_{i,j}$ has size less than $\frac{8k^2(k+1)h^2}{\varepsilon^2}$, which implies $|\zeta_B| \leq O(\frac{k^4 h^2}{\varepsilon^2} \log \frac{D}{d})$.

Clearly the above procedure can be implemented in polynomial time, and the fact that every set of ζ_B is contained in the same component of F_ε follows from Claim 12 and Claim 13. Finally, any path in a graph with at most n vertices has less than n edges, so that $\frac{D}{d} < 2n^2/\varepsilon$, given that the ratio of the longest to the shortest edge is $2n/\varepsilon$ (note that $d > 0$ by definition). Hence the claimed bound of $|\zeta_B| \leq O(\frac{k^4 h^2}{\varepsilon^2} \log \frac{n}{\varepsilon})$ follows. ◀

3.2 Tree decompositions with logarithmic height

Given a tree decomposition T of logarithmic height, using Lemma 10 we are ready to compute a set of partitions Π_B of FPT size for each bag B, such that a near-optimal solution conforms to Π_B. In particular, by Lemma 8 we may assume that the height of T is $h = O(k \log n)$, which means that the bound on ζ_B in Lemma 10 translates to $O(\frac{k^6}{\varepsilon^2} \log^3 \frac{n}{\varepsilon})$. As in the previous section, we need to apply Lemma 9 in order to bound the aspect ratio of the graph, so that n denotes the number of vertices of the original input graph, while now the graph G has at most n vertices but the ratio between the longest and shortest edge is at most $2n/\varepsilon$. We begin by describing how to obtain the near-optimal solution, after which we will identify the partition sets Π_B.

3.2.1 A near-optimal solution

Bateni, Hajiaghayi, and Marx [4] construct a near-optimal solution by modifying the optimum. We will use similar techniques to obtain our near-optimal solution, but we construct it by instead modifying the $(1+\varepsilon)$-approximate solution F_ε given by Lemma 10. In particular, we construct a near-optimal $(1+\varepsilon)^2$-approximation $\widetilde{F}_\varepsilon$ from F_ε. The main idea to obtain $\widetilde{F}_\varepsilon$ is to connect components of F_ε if they are very close to one another. As before however, doing this naively would incur too much cost for the additional connections.

To make sure that the cost incurred by connecting components of F_ε is not too large, [4] introduced a partial order on the components based on the structure of a given rooted tree decomposition T. Let C_1, C_2 be two components of F_ε that share a bag B of T, i.e., $V(C_1) \cap B \neq \emptyset$ and $V(C_2) \cap B \neq \emptyset$. Since C_1 and C_2 are connected subgraphs of the input graph, a basic property of tree decompositions implies that there are (connected) subtrees T_1 and T_2 of T induced by the respective bags containing vertices of C_1 and C_2. Because these components both contain vertices of B, the node associated with B is part of both T_1 and T_2, and therefore the roots of both subtrees lie on the path from this node to the root of T. This defines an order on C_1 and C_2, and we write $C_1 \leq C_2$ if the root of T_1 is farther from the root of T than the root of T_2 is. This order is defined for any two components of F_ε that share a bag, and thus we obtain a partial order on the components of F_ε, where any components that do not share a bag are incomparable.

Using the defined order, [4] connect components of the optimum solution that are very close to each other. In order to obtain smaller partition sets, we modify the distance bound used in this procedure compared to [4]. In particular, for any value $x > 0$, let $\lfloor x \rfloor_2 = 2^{\lfloor \log_2 x \rfloor}$ denote the largest power of 2 that is at most x. Now, starting with $\widetilde{F}_\varepsilon = F_\varepsilon$ we connect components by exhaustively applying the following rule:

▶ **Rule 2.** *If C, C' are components of F_ε with $C \leq C'$ and $\operatorname{dist}(C, C') \leq \frac{\varepsilon}{k} \lfloor \operatorname{cost}(C) \rfloor_2$ but C and C' lie in different components of $\widetilde{F}_\varepsilon$, then add a shortest path of length $\operatorname{dist}(C, C')$ between C and C' to the solution $\widetilde{F}_\varepsilon$.*

A crucial but subtle observation is that for a component C of F_ε there can be many components $C' \leq C$ at distance at most $\frac{\varepsilon}{k} \lfloor \operatorname{cost}(C) \rfloor_2$ to C, which however are not connected to C in the resulting solution $\widetilde{F}_\varepsilon$ according to Rule 2. This makes it non-trivial to find small partition sets Π_B. Contrary to this however, an important property of the order on the components is that for any component C of F_ε, there are at most k other components C' for which $C \leq C'$, as we will argue for the following lemma to bound the cost of $\widetilde{F}_\varepsilon$. In particular, the lemma implies that $\widetilde{F}_\varepsilon$ is a near-optimal $(1+\varepsilon)^2$-approximation, given that F_ε is a $(1+\varepsilon)$-approximation.

▶ **Lemma 14.** *The cost of the solution $\widetilde{F}_\varepsilon$ obtained by Rule 2 from F_ε is at most $(1+\varepsilon) \operatorname{cost}(F_\varepsilon)$.*

3.2.2 Partitioning active terminals

Given the construction of the $(1+\varepsilon)^2$-approximate solution $\widetilde{F}_\varepsilon$ above, the next step is to find a set of partitions Π_B of the active terminals A_B for each bag B, such that $\widetilde{F}_\varepsilon$ conforms with all sets Π_B. In the following, fix a bag B of the given tree decomposition T. The technique used by [4] is to guess a small net for each active component of bag B,[3] so that every terminal of A_B close to a net point must be part of the same component in the approximate solution, after taking the order on the active components as defined previously into account. Next we choose a net on the terminals of each active component and bound its size.

▶ **Lemma 15.** *Let $N \subseteq A_B \cap C$ be an $\frac{\varepsilon}{k}\lfloor \mathrm{cost}(C)\rfloor_2$-net of the metric induced by the active terminals of some active component C. The size of the net can be bounded by $|N| \leq \lfloor 4k/\varepsilon \rfloor$.*[4]

Following the algorithm of [4], the next step would be to guess such an $\frac{\varepsilon}{k}\lfloor \mathrm{cost}(C)\rfloor_2$-net for each of the at most $k+1$ active components C of the bag B. By Lemma 15, the total number of net points for these at most $k+1$ nets is at most $\lfloor 4k/\varepsilon \rfloor (k+1) = O(k^2/\varepsilon)$. Since however there may be up to n active terminals, guessing these nets for all active components can result in $n^{O(k^2/\varepsilon)}$ many possible choices, which leads to an XP time algorithm. To circumvent this, we instead consider the partition ζ_B of the active terminals as given by Lemma 10, and guess which of the sets of ζ_B contains a net point. We will argue that since the size of ζ_B is $O(\frac{k^6}{\varepsilon^2}\log^3\frac{n}{\varepsilon})$ there are only $(\frac{k}{\varepsilon}\log\frac{n}{\varepsilon})^{O(k^2/\varepsilon)}$ possibilities, leading to a faster algorithm.

More concretely, to compute a set of partitions Π_B that $\widetilde{F}_\varepsilon$ conforms to, our algorithm considers every sequence $((S_1,\delta_1),(S_2,\delta_2),\ldots,(S_\ell,\delta_\ell),\rho)$ of at most $k+1$ pairs (S_j,δ_j) and partitions ρ of the index set $\{1,\ldots,\ell\}$, where each S_j is a subset of the parts of ζ_B such that $|S_j| \leq \lfloor 4k/\varepsilon \rfloor$, and $\delta_j \in \{2^q \mid q \in \mathbb{N}_0 \wedge 0 \leq q \leq \log_2(2n^2/\varepsilon)\}$ is an integer power of 2 between 1 and $2n^2/\varepsilon$, where n is the number of vertices of the original input graph in accordance with Lemma 9. From every such sequence, the algorithm attempts to construct a partition of the active terminals, and if it succeeds adds it to the set Π_B. As we will show, in this process the algorithm will successfully construct one partition π of A_B that $\widetilde{F}_\varepsilon$ conforms to.

Before describing how a partition of the active terminals arises from such a sequence, we bound the number of these sequences, which determines the running time. By Lemma 10, $|\zeta_B| = O(\frac{k^6}{\varepsilon^2}\log^3\frac{n}{\varepsilon})$ if the tree decomposition T has logarithmic height, so that there are at most $\binom{|\zeta_B|}{\lfloor 4k/\varepsilon \rfloor} = (\frac{k}{\varepsilon}\log\frac{n}{\varepsilon})^{O(k/\varepsilon)}$ possible choices for each S_j. Clearly there are $O(\log\frac{n}{\varepsilon})$ choices for each δ_j, and $\ell^\ell = k^{O(k)}$ possible partitions ρ, given that $\ell \leq k+1$. Since a sequence contains ℓ sets S_j, the total number of sequences is bounded by $(\frac{k}{\varepsilon}\log\frac{n}{\varepsilon})^{O(k^2/\varepsilon)}$.

Each sequence may give rise to a partition $\pi \in \Pi_B$ of the active terminals as follows. First, let $\pi = \{Y_1,\ldots,Y_{|\rho|}\}$, i.e., π has the same number of sets as the partition ρ. Let $U_j = \bigcup_{U\in S_j} U$ denote the set of active terminals in S_j, and let $\rho(j)$ be the part of ρ containing j. We distinguish between active terminals $t \in A_B$ that lie in some set U_j and those that do not:

- if $t \in U_j$ for some $j \in [\ell]$ then $t \in Y_{\rho(j)}$ (i.e., $U_j \subseteq Y_{\rho(j)}$), and
- otherwise, if $p_t \in \{1,\ldots,\ell\}$ denotes the smallest index for which $\mathrm{dist}(t,U_{p_t}) \leq \frac{\varepsilon}{k}\delta_{p_t}$, then $t \in Y_{\rho(p_t)}$.

If this π is a partition of A_B we add π to Π_B, and otherwise we dismiss the current sequence. Clearly π can be constructed in polynomial time, given a sequence.

[3] [4] refers to these nets as *groups*.
[4] A slightly worse bound follows from [4, Lemma 19].

▶ **Lemma 16.** *The $(1+\varepsilon)^2$-approximate solution $\widetilde{F}_\varepsilon$ conforms to the set Π_B of partitions constructed above.*

Proof. Consider the $(1+\varepsilon)$-approximate solution F_ε of Lemma 10 from which $\widetilde{F}_\varepsilon$ is constructed according to Rule 2, and the partition ζ_B of A_B as given by Lemma 10. Let the active components of F_ε be $C_1, \ldots, C_\ell$ indexed according to their order, i.e., $C_j \leq C_{j'}$ if and only if $j \leq j'$. For each active component C_j we fix an $\frac{\varepsilon}{k}\lfloor \mathrm{cost}(C_j)\rfloor_2$-net N_j of size at most $\lfloor 4k/\varepsilon \rfloor$ according to Lemma 15. Now, consider the sequence $((S_1,\delta_1),(S_2,\delta_2),\ldots,(S_\ell,\delta_\ell),\rho)$, where

- S_j contains exactly those sets of ζ_B that contain at least one net point of N_j,
- $\delta_j = \lfloor \mathrm{cost}(C_j)\rfloor_2$, and
- ρ is the partition of the index set corresponding to the components of $\widetilde{F}_\varepsilon$, i.e., $\rho(j) = \rho(j')$ if and only if C_j and $C_{j'}$ lie in the same component in $\widetilde{F}_\varepsilon$.

Recall that after applying Lemma 9 to the input, the ratio between the shortest and longest edge is at most $2n/\varepsilon$, where n is the number of vertices of the original input graph. Since we assume that the length of the shortest edge is 1, the cost of any component lies between 1 and $2n^2/\varepsilon$, given that a component is a tree with less than n edges. Therefore $\lfloor \mathrm{cost}(C_j)\rfloor_2 \in \{2^q \mid q \in \mathbb{N}_0 \wedge 0 \leq q \leq \log_2(2n^2/\varepsilon)\}$, which means that the algorithm will consider the above sequence in some iteration.

We now turn to $\pi = \{Y_1, \ldots, Y_{|\rho|}\}$ constructed for this sequence, and show that it is a partition of A_B and that $\widetilde{F}_\varepsilon$ conforms to it. For this, note that no set of ζ_B contains net points of several active components of F_ε, since by Lemma 10 all active terminals in the same set of ζ_B also belong to the same component of F_ε. Thus the sets S_j as defined above (and also the corresponding sets U_j) are pairwise disjoint. This means that, due to the definition of ρ, any two terminals $t \in U_j$ and $t' \in U_{j'}$ end up in the same set of π if and only if t and t' belong to the same component of $\widetilde{F}_\varepsilon$ (as $U_j \subseteq Y_{\rho(j)}$).

Now consider a terminal $t \in A_B$, which does not lie in any U_j, and let q be the index of the active component C_q of F_ε containing t. As $\delta_q = \lfloor \mathrm{cost}(C_q)\rfloor_2$, N_q is an $\frac{\varepsilon}{k}\delta_q$-net of $C_q \cap A_B$. Also, we chose S_q so that $N_q \subseteq U_q$. Hence we get $\mathrm{dist}(t, U_q) \leq \mathrm{dist}(t, N_q) \leq \frac{\varepsilon}{k}\delta_q$, and the definition of p_t implies $p_t \leq q$. Now C_{p_t} is either equal to C_q, or C_q is connected to the component C_{p_t} in the approximate solution $\widetilde{F}_\varepsilon$ according to Rule 2: on one hand we have $C_{p_t} \leq C_q$ due to the order of the indices, and at the same time by Lemma 10 we have $U_{p_t} \subseteq V(C_{p_t}) \cap A_B$, which implies

$$\mathrm{dist}(C_q, C_{p_t}) \leq \mathrm{dist}(t, C_{p_t}) \leq \mathrm{dist}(t, U_{p_t}) \leq \frac{\varepsilon}{k}\delta_{p_t} = \frac{\varepsilon}{k}\lfloor \mathrm{cost}(C_{p_t})\rfloor_2.$$

Hence we can conclude that t lies in the same component as C_{p_t} in $\widetilde{F}_\varepsilon$.

In conclusion, adding U_j to $Y_{\rho(j)}$ and t to $Y_{\rho(p_t)}$ for each terminal t not lying in any U_j, partitions the terminals according to the components of $\widetilde{F}_\varepsilon$. Hence π is a partition of the active terminals A_B that is added to Π_B, and $\widetilde{F}_\varepsilon$ conforms to it. ◀

Using all of the above, we can finally prove our main theorem, stating that there is an EPAS for Steiner Forest parameterized by the treewidth.

Proof of Theorem 1. The first steps of our algorithm are to preprocess the given tree decomposition using Lemma 8 so that it is nice and its height is $O(k \log n)$, and the input graph using Lemma 9 so that the aspect ratio is bounded (which means that n denotes the number of vertices in the original input graph). We then compute the partition sets Π_B for all bags B using the above procedure, resulting in partition sets of size $(\frac{k}{\varepsilon}\log\frac{n}{\varepsilon})^{O(k^2/\varepsilon)} = 2^{O(\frac{k^2}{\varepsilon}\log\frac{k}{\varepsilon})} \cdot n^{o(1)}$. Here, we are using a well-known Win/Win argument: if $k^2/\varepsilon < \sqrt{\log n}$, then $(\log n)^{k^2/\varepsilon} = n^{o(1)}$; otherwise, $\log n \leq k^4/\varepsilon^2$, therefore $(\frac{k}{\varepsilon}\log\frac{n}{\varepsilon})^{O(k^2/\varepsilon)} = (\frac{k}{\varepsilon})^{O(\frac{k^2}{\varepsilon})}$.

Since each partition of a set Π_B can be computed in polynomial time, and the number of bags of the nice tree decomposition is $O(kn)$, this takes $2^{O(\frac{k^2}{\varepsilon}\log\frac{k}{\varepsilon})} \cdot n^{O(1)}$ time. Next we apply Theorem 5 to compute a solution that is at least as good as $\widetilde{F}_\varepsilon$ conforming to all Π_B, in $2^{O(\frac{k^2}{\varepsilon}\log\frac{k}{\varepsilon})} \cdot n^{O(1)}$ time. Hence we obtain a $(1+\varepsilon)^2$-approximation F. According to Lemma 9, F can be converted into a $((1+\varepsilon)^2+\varepsilon)$-approximation to the original input graph. Since for any $\varepsilon' > 0$ we may choose $\varepsilon = \Theta(\varepsilon')$ so that $((1+\varepsilon)^2+\varepsilon) \leq 1+\varepsilon'$, we obtain an EPAS as claimed. ◀

4 Vertex cover

In this section we consider the parameterization by the size of a *vertex cover*, which is a set $S \subseteq V$ of vertices such that every edge is incident on at least one of the vertices of S. In the full version of this paper we present an easy FPT algorithm based on Theorem 5.

Our goal here is to present a reduction showing that the algorithm we have given for Steiner Forest parameterized by vertex cover is essentially optimal, assuming the ETH. Recall that the ETH is the hypothesis that 3-SAT on instances with n variables cannot be solved in time $2^{o(n)}$. We will give a reduction that given a 3-SAT instance ϕ, produces an equivalent Steiner Forest instance with vertex cover at most $O(n/\log n)$. We stress that our reduction works even for unweighted instances.

▶ **Theorem 17.** *If there exists an algorithm which, given an unweighted Steiner Forest instance on n vertices with vertex cover k, finds an optimal solution in time $2^{o(k\log k)}n^{O(1)}$, then the ETH is false.*

Proof. We present a reduction from 3-SAT. Before we proceed, we would like to add to our formula the requirement that the variable set comes partitioned into three sets in a way that each clause contains at most one variable from each set. It is not hard to show that this does not affect the complexity of the instance much, as we demonstrate in the following claim.

▷ Claim 18. Suppose that there exists an algorithm that takes as input a 3-SAT instance ϕ on n variables and a partition of the variables into three sets of equal size, such that each clause contains at most one variable from each set, and decides if ϕ is satisfiable in time $2^{o(n)}$. Then, the ETH is false.

In the remainder we will then assume that we are given a formula ϕ on $3n$ variables which are partitioned into three sets of size n as specified by the previous claim. Without loss of generality, suppose that n is a power of 4 (this can be achieved by adding dummy variables). Note that this ensures that $\frac{\log n}{2}$ and $\sqrt{n}$ are both integers.

We construct an equivalent instance of Steiner Forest as follows. Let $L = \lceil \frac{n}{\log^2 n} \rceil$. We begin by constructing i choice gadgets, i.e., for $i \in \{1, \ldots, 3\log n\}$ we make:

- $2L$ left vertices, labeled ℓ^i_j, for $j \in \{0, \ldots, 2L-1\}$.
- $2L$ right vertices, labeled r^i_j, for $j \in \{0, \ldots, 2L-1\}$.
- $\sqrt{n}$ middle vertices, labeled m^i_j, for $j \in \{0, \ldots, \sqrt{n}-1\}$.
- We connect all middle vertices to all left and right vertices, that is, for all $j \in \{0, \ldots, 2L-1\}$ and $j' \in \{0, \sqrt{n}-1\}$ we connect ℓ^i_j and r^i_j to $m^i_{j'}$.
- For each $j \in \{0, \ldots, 2L-1\}$ we add a demand from ℓ^i_j to r^i_j.

Notice that the graph we have constructed so far contains $3\log n$ choice gadgets, each of which has $4L + \sqrt{n} = O(n/\log^2 n)$ vertices, so the graph at the moment contains $O(n/\log n)$ vertices in total.

Before we proceed, let $X = X_a \cup X_b \cup X_c$ be the set of $3n$ variables of ϕ that was given to us partitioned into three sets of size n. We partition X into $3\log n$ groups $X_1, \ldots, X_{3\log n}$ in a way that (i) $|X_i| \leq \lceil n/\log n \rceil$ for all $i \in \{1, \ldots, \log n\}$ and (ii) for all $i \in \{1, \ldots, \log n\}$ we have X_i is contained in one of X_a, X_b, X_c. This can be done by taking the n variables of X_a and partitioning them arbitrarily into groups $X_1, \ldots, X_{\log n}$ of size as equal as possible (therefore at most $\lceil n/\log n \rceil$), and we proceed similarly for X_b, X_c. Rename the variables of ϕ so that for each i we have that $X_i = \{x_{(i,0)}, \ldots, x_{(i,\lceil n/\log n \rceil - 1)}\}$.

To give some intuition, we will now say that, for $i \in \{1, \ldots, 3\log n\}$, the choice gadget i represents the variables of the set X_i. In particular, for each $j \in \{0, \ldots 2L-1\}$, we will say that the way that the demand $\ell^i_j \to r^i_j$ was satisfied encodes the assignment to the $\frac{\log n}{2}$ variables $\{x_{(i,\frac{j\log n}{2})}, \ldots, x_{(i,\frac{(j+1)\log n}{2}-1)}\}$. More precisely, in our intended solution the demand $\ell^i_j \to r^i_j$ is satisfied by connecting both terminals to a common middle vertex $m^i_{j'}$. We can infer the assignment to the $\frac{\log n}{2}$ variables this represents simply by writing down the binary representation of j', which is a number between 0 and $\sqrt{n}-1$, hence a number with $\frac{\log n}{2}$ bits. Note that this way we represent $2L \cdot \frac{\log n}{2} \geq \lceil \frac{n}{\log n} \rceil$ variables, that is, we can represent the assignment to all the variables of the group.

Armed with this intuition, we can now complete our construction. For each clause c we construct two new vertices, c_1, c_2 and add a demand from c_1 to c_2. For each literal contained in c, suppose that the literal involves the variable $x_{(i,\frac{j\log n}{2}+\alpha)}$ for $i \in \{1, \ldots, 3\log n\}$, $j \in \{0, \ldots, 2L-1\}$, $\alpha \in \{0, \ldots, \frac{\log n}{2} - 1\}$. We then connect c_1 to ℓ^i_j. Furthermore, if $x_{(i,\frac{j\log n}{2}+\alpha)}$ appears positive in c, we connect c_2 to all $m^i_{j'}$ such that the binary representation of j' has a 1 in position α. If on the other hand $x_{(i,\frac{j\log n}{2}+\alpha)}$ appears negative in c, we connect c_2 to all $m^i_{j'}$ such that the binary representation of j' has a 0 in position α. In other words, we connect c_2 to all the middle vertices to which ℓ^i_j could be connected and are consistent with an assignment that satisfies c using the current literal. After repeating the above for all literals of each clause the construction is complete. We set the target cost to be $B = 2m + 12L\log n$.

Before we argue about the correctness of the reduction, let us observe that if the reduction preserves the satisfiability of ϕ, then we obtain the theorem, because the instance we constructed has vertex cover $k = O(n/\log n)$ and size polynomial in the size of ϕ. Indeed, as we argued the choice gadgets have $O(n/\log n)$ vertices in total, and all further edges we added have an endpoint in a choice gadget. If there was an algorithm solving the new instance in time $k^{o(k)}n^{O(1)}$, this would give a $2^{o(n)}$ algorithm to decide ϕ.

Regarding correctness, let us first observe that if ϕ is satisfiable, we can obtain a valid solution using the intuitive translation from assignments to choice gadget solutions we gave above. In particular, for each $i \in \{1, \ldots, 3\log n\}$ and $j \in \{0, \ldots, 2L-1\}$, we consider the assignment to variables $\{x_{(i,\frac{j\log n}{2})}, \ldots, x_{(i,\frac{(j+1)\log n}{2}-1)}\}$ as a binary number, which must have a value j' between 0 and $\sqrt{n}-1$. We then connect both ℓ^i_j, r^i_j to $m^i_{j'}$. Repeating this satisfies all demands internal to choice gadgets and uses $3\log n \cdot 4L = 12L\log n$ edges. Consider now a clause c and the demand from c_1 to c_2. Since we started with a satisfying assignment, c must contain a true literal, say involving the variable $x_{(i,\frac{j\log n}{2}+\alpha)}$. We select the edge from c_1 to ℓ^i_j. Furthermore, we observe that c_2 must be a neighbor of all vertices $m^i_{j'}$ such that the bit in position α of the binary representation of j' agrees with the value of $x_{(i,\frac{j\log n}{2}+\alpha)}$. Since ℓ^i_j is already connected to such a $m^i_{j'}$, we select the edge from that vertex to c_2 to satisfy the demand for this clause. We have therefore spent $2m$ further edges for the clause demands and have used a budget of exactly B.

For the converse direction, suppose we have a solution of cost B. We first observe that each vertex r^i_j must be connected to a middle vertex $m^i_{j'}$, since all right vertices are terminals, but such vertices only have edges connecting them to middle vertices. Recall that, for each i, j, the left vertex ℓ^i_j must be in the same component of the solution as r^i_j, since there is a demand between these two vertices. Hence, each ℓ^i_j is in the same component of the solution as some $m^i_{j'}$. We now slightly edit the solution as follows: suppose there exists a vertex ℓ^i_j which is not directly connected in the solution to any middle vertex $m^i_{j'}$. Since this vertex is in the same component as one such vertex $m^i_{j'}$, we add to the solution the edge connecting them, and since this creates a cycle, remove from the solution another edge incident on ℓ^i_j. Doing this repeatedly ensures that each ℓ^i_j is connected to a middle vertex $m^i_{j'}$ in the solution without increasing the total cost.

We now observe that since each ℓ^i_j and each r^i_j is connected to at least one middle vertex $m^i_{j'}$ in the solution, this already uses a cost of $3 \log n \cdot 4L = 12L \log n$. Furthermore, for each clause we have constructed two terminals, each of which must use at least one of its incident edges, giving an extra cost of $2m$. Since our budget is exactly $2m + 12L \log n$, we conclude that each terminal constructed for a clause is incident on exactly one edge, and each ℓ^i_j and each r^i_j is connected to exactly one middle vertex. Crucially, these observations imply the following fact: if for some i, j, j' we have that ℓ^i_j and $m^i_{j'}$ are in the same component of the solution, then the edge connecting ℓ^i_j and $m^i_{j'}$ is part of the solution. To see this, observe that any path connecting ℓ^i_j and $m^i_{j'}$ that is not a direct edge would need to have length at least 3. However, no clause terminal can be an internal vertex of such a path, since clause terminals have degree 1 in the solution. Furthermore, if we remove clause terminals from the graph, left and right vertices also have degree 1 in the remaining solution, so such vertices also cannot be internal in the path. Finally, middle vertices are an independent set, so it is impossible for all internal vertices of a path of length at least 3 to be middle vertices.

Armed with the observation that ℓ^i_j and $m^i_{j'}$ are in the same connected component of the solution if and only if they are directly connected, we are ready to extract a satisfying assignment from the Steiner forest. For each i, j, if ℓ^i_j is connected to $m^i_{j'}$ we write j' in binary and assign to variable $x_{(i, \frac{j \log n}{2} + \alpha)}$, for $\alpha \in \{0, \ldots, \frac{logn}{2} - 1\}$ the value in position α of the binary representation of j'. We claim that this assignment must be satisfying. Indeed, consider the clause c, and the terminals c_1, c_2 which represent it. Since these terminals have a demand, they must be in the same component. Because c_1 has at most three neighbors which are in different choice gadgets (as each clause contains variables from distinct groups), we can see that c_1 must be connected to some ℓ^i_j and c_2 to some $m^i_{j'}$ in the solution, such that ℓ^i_j and $m^i_{j'}$ are in the same component, and are therefore directly connected. But if ℓ^i_j is directly connected to $m^i_{j'}$ this means that the assignment we extracted from ℓ^i_j gives a value to a variable $x_{(i, \frac{j \log n}{2} + \alpha)}$ which satisfies the clause c, hence we have a satisfying assignment. ◀

5 Feedback Edge Set

A *feedback edge set* of a graph is a set of edges that when removed renders the graph acyclic. It is well-known that if G is a connected undirected graph on n vertices and m edges, then all minimal feedback edge sets of G have size $k = m - n + 1$. Indeed, such a set can be constructed in polynomial time by repeatedly locating a cycle in the graph and selecting an arbitrary edge of the cycle to insert into the feedback edge set.

In this section we will consider Steiner Forest parameterized by the feedback edge set of the input graph, which we will denote by k. Unlike the vertex cover section, here our main result is positive: we show that Steiner Forest can be solved optimally in time

$2^{O(k)}n^{O(1)}$, that is, in time single-exponential in the parameter. Since we are able to achieve a single-exponential dependence, it is straightforward to see that this is optimal under the ETH.

▶ **Theorem 19.** *If there is an algorithm solving* Steiner Tree *in time* $2^{o(k)}n^{O(1)}$*, where* k *is the feedback edge set of the input, then the ETH is false.*

Let us now proceed to the detailed presentation of the algorithm. Suppose that we are given a budget b and we want to decide if there exists a Steiner Forest solution F such that $\text{cost}(F) \leq b$. We start by applying a simple reduction rule.

▶ **Rule 3.** *Suppose we have a* Steiner Forest *instance on graph* G *with weight function* w *and budget* b*, such that a vertex* $u \in V$ *has degree* 1*. If* $u \notin R$*, then delete* u*. If* $u \in R$*, let* v *be the unique neighbor of* u*. Then set* $b' := b - w(uv)$*, delete* u *from the graph and the demand* $\{u, v\}$ *from* D *if it exists, and replace, for each* $x \in V \setminus \{u, v\}$ *such that* $\{u, x\} \in D$ *the demand* $\{u, x\}$ *with the demand* $\{v, x\}$*.*

▶ **Lemma 20.** *Rule 3 is safe.*

Observe that if we apply Rule 3 exhaustively, then the minimum degree of the graph is 2. As we show next, relatively few vertices can have higher degree.

▶ **Lemma 21.** *Suppose we have a* Steiner Forest *instance with feedback edge set of size* k *and minimum degree at least* 2*. Then* G *contains at most* $2k$ *vertices of degree at least* 3*.*

In the remainder we will assume that we have a Steiner Forest instance $G = (V, E)$ with a feedback edge set $H \subseteq E$ of size k, to which 3 can no longer be applied. We will say that a vertex v is *special* if v is incident on an edge of H or v has degree at least 3. By Lemma 21 we know that G contains at most $4k$ special vertices.

We define a *topological edge* (topo-edge for short) as follows: a path P in G is a topological edge if the two endpoints of P are special vertices and all internal vertices of P are non-special. Note that by this definition, all edges of H form topo-edges, since the endpoints of such edges are special. We observe the following:

▶ **Lemma 22.** *Suppose we have a graph* G *with feedback edge set of size* k *and minimum degree at least* 2*. Then* G *contains at most* $5k$ *topological edges.*

We are now ready to state the main algorithmic result of this section.

▶ **Theorem 23.** *There is an algorithm that solves* Steiner Forest *on instances with* n *vertices and a feedback edge set of size* k *in* $2^{O(k)}n^{O(1)}$ *time.*

References

1 Ajit Agrawal, Philip Klein, and Ramamoorthi Ravi. When trees collide: An approximation algorithm for the generalized steiner problem on networks. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 134–144, 1991.

2 Brenda S Baker. Approximation algorithms for np-complete problems on planar graphs. *Journal of the ACM (JACM)*, 41(1):153–180, 1994.

3 Yair Bartal and Lee-Ad Gottlieb. Near-linear time approximation schemes for steiner tree and forest in low-dimensional spaces. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1028–1041, 2021.

4 MohammadHossein Bateni, MohammadTaghi Hajiaghayi, and Dániel Marx. Approximation schemes for steiner forest on planar graphs and graphs of bounded treewidth. *Journal of the ACM (JACM)*, 58(5):1–37, 2011.

5 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets möbius: fast subset convolution. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 67–74, 2007.

6 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.

7 Hans L. Bodlaender and Torben Hagerup. Parallel algorithms with optimal speedup for bounded treewidth. *SIAM J. Comput.*, 27(6):1725–1746, 1998. doi:10.1137/S0097539795289859.

8 Hans L Bodlaender, Matthew Johnson, Barnaby Martin, Jelle J Oostveen, Sukanya Pandey, Daniel Paulusma, Siani Smith, and Erik Jan van Leeuwen. Complexity framework for forbidden subgraphs iv: The steiner forest problem. *arXiv preprint*, 2023. arXiv:2305.01613.

9 Glencora Borradaile, Philip Klein, and Claire Mathieu. An o (n log n) approximation scheme for steiner tree in planar graphs. *ACM Transactions on Algorithms (TALG)*, 5(3):1–31, 2009.

10 Jarosław Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. Steiner tree approximation via iterative randomized rounding. *Journal of the ACM (JACM)*, 60(1):1–33, 2013.

11 Xiuzhen Cheng and Ding-Zhu Du. *Steiner trees in industry*, volume 11. Springer Science & Business Media, 2013.

12 Rajesh Chitnis, Andreas Emil Feldmann, and Pasin Manurangsi. Parameterized approximation algorithms for bidirected steiner network problems. *ACM Transactions on Algorithms (TALG)*, 17(2):1–68, 2021.

13 Miroslav Chlebík and Janka Chlebíková. The steiner tree problem on graphs: Inapproximability results. *Theoretical Computer Science*, 406(3):207–214, 2008.

14 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.

15 Stuart E Dreyfus and Robert A Wagner. The steiner problem in graphs. *Networks*, 1(3):195–207, 1971.

16 Ding-Zhu Du, JM Smith, and J Hyam Rubinstein. *Advances in Steiner trees*, volume 6. Springer Science & Business Media, 2013.

17 Pavel Dvorák, Andreas E Feldmann, Dusan Knop, Tomás Masarík, Tomas Toufar, and Pavel Vesely. Parameterized approximation schemes for steiner trees with small number of steiner vertices. *SIAM Journal on Discrete Mathematics*, 35(1):546–574, 2021.

18 David Eisenstat, Philip Klein, and Claire Mathieu. An efficient polynomial-time approximation scheme for steiner forest in planar graphs. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 626–638. SIAM, 2012.

19 Andreas Emil Feldmann, Euiwoong Lee, and Pasin Manurangsi. A survey on approximation in parameterized complexity: Hardness and algorithms. *Algorithms*, 13(6):146, 2020.

20 Bernhard Fuchs, Walter Kern, D Molle, Stefan Richter, Peter Rossmanith, and Xinhui Wang. Dynamic programming for minimum steiner trees. *Theory of Computing Systems*, 41(3):493–500, 2007.

21 Elisabeth Gassner. The steiner forest problem revisited. *Journal of Discrete Algorithms*, 8(2):154–163, 2010.

22 Tatsuya Gima, Tesshu Hanaka, Masashi Kiyomi, Yasuaki Kobayashi, and Yota Otachi. Exploring the gap between treedepth and vertex cover through vertex integrity. *Theor. Comput. Sci.*, 918:60–76, 2022. doi:10.1016/J.TCS.2022.03.021.

23 Anupam Gupta and Jochen Könemann. Approximation algorithms for network design: A survey. *Surveys in Operations Research and Management Science*, 16(1):3–20, 2011.

24 Frank K Hwang and Dana S Richards. Steiner tree problems. *Networks*, 22(1):55–89, 1992.

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

26 Michael Lampis, Nikolaos Melissinos, and Manolis Vasilakis. Parameterized max min feedback vertex set. In Jérôme Leroux, Sylvain Lombardy, and David Peleg, editors, *48th International Symposium on Mathematical Foundations of Computer Science, MFCS 2023, August 28 to September 1, 2023, Bordeaux, France*, volume 272 of *LIPIcs*, pages 62:1–62:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.MFCS.2023.62.

27 Michael Lampis and Manolis Vasilakis. Structural parameterizations for two bounded degree problems revisited. In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman, editors, *31st Annual European Symposium on Algorithms, ESA 2023, September 4-6, 2023, Amsterdam, The Netherlands*, volume 274 of *LIPIcs*, pages 77:1–77:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ESA.2023.77.

28 Ivana Ljubic. Solving steiner trees: Recent advances, challenges, and perspectives. *Networks*, 77(2):177–204, 2021.

29 Jesper Nederlof. Fast polynomial-space algorithms using möbius inversion: Improving on steiner tree and related problems. In *International Colloquium on Automata, Languages, and Programming*, pages 713–725. Springer, 2009.

30 Satish B Rao and Warren D Smith. Approximating geometrical graphs via "spanners" and "banyans". In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 540–550, 1998.

31 R Ravi. A primal-dual approximation algorithm for the steiner forest problem. *Information processing letters*, 50(4):185–189, 1994.

32 Hao Tang, Genggeng Liu, Xiaohua Chen, and Naixue Xiong. A survey on steiner tree construction and global routing for vlsi design. *IEEE Access*, 8:68593–68622, 2020.

33 Stefan Voß. Steiner tree problems in telecommunications. In *Handbook of optimization in telecommunications*, pages 459–492. Springer, 2006.

34 David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.