SIAM J. Discrete Math.
Vol. 33, No. 1, pp. 327–345

PARAMETERIZED SINGLE-EXPONENTIAL TIME POLYNOMIAL SPACE ALGORITHM FOR STEINER TREE[*]

FEDOR V. FOMIN[†], PETTERI KASKI[‡], DANIEL LOKSHTANOV[§], FAHAD PANOLAN[†], AND SAKET SAURABH[¶]

Abstract. In the Steiner Tree problem, we are given as input a connected n-vertex graph with edge weights in $\{1, 2, \ldots, W\}$, and a set of k terminal vertices. Our task is to compute a minimum-weight tree that contains all of the terminals. The main result of the paper is an algorithm solving Steiner Tree in time $\mathcal{O}(7.97^k \cdot n^4 \cdot \log W)$ and using $\mathcal{O}(n^3 \cdot \log nW \cdot \log k)$ space. This is the first single-exponential time, polynomial space FPT algorithm for the weighted Steiner Tree problem. Whereas our main result seeks to optimize the polynomial dependency in n for both the running time and space usage, it is possible to trade between polynomial dependence in n and the single-exponential dependence in k to obtain faster running time as a function of k, but at the cost of increased running time and space usage as a function of n. In particular, we show that there exists a polynomial space algorithm for Steiner Tree running in $\mathcal{O}(6.751^k n^{\mathcal{O}(1)} \log W)$ time. Finally, by pushing such a trade-off between a polynomial in n and an exponential in k dependencies, we show that for any $\epsilon > 0$ there is an $n^{\mathcal{O}(f(\epsilon))} \log W$ space $4^{(1+\epsilon)k} n^{\mathcal{O}(f(\epsilon))} \log W$ time algorithm for Steiner Tree, where f is a computable function depending only on ϵ.

Key words. Steiner tree, FPT algorithm, recurrence relation

AMS subject classifications. 05C85, 68Q25, 68W05, 68W40

DOI. 10.1137/17M1140030

1. Introduction.

Steiner Tree problem, or minimum Steiner Tree problem, is a term for a class of problems in combinatorial optimization. Steiner Tree problem is formulated in various subfields of computer science like graph algorithms, computational geometry, etc. In this work we study the Steiner Tree problem in graphs. In the Steiner Tree problem, we are given as input a connected n-vertex graph, a nonnegative weight function $w : E(G) \to \{1, 2, \ldots, W\}$, and a set of terminal vertices $T \subseteq V(G)$. The task is to find a minimum-weight connected subgraph ST of G containing all terminal nodes T. In this paper we use the parameter $k = |T|$. In this work we consider the word RAM model of computation. But we assume that the edge weights are *large* and hence the memory required for weight W is $\mathcal{O}(\log W)$ words. Moreover, operations like compare, add, or multiply two weights W_1 and W_2 take $\mathcal{O}(\log(W_1 \cdot W_2))$ time.

Steiner Tree is one of the central and best-studied problems in computer science with various applications. We refer to the book of Prömel and Steger [22] for an

[*]Received by the editors July 24, 2017; accepted for publication (in revised form) October 22, 2018; published electronically February 5, 2019. Preliminary versions of this paper appeared in the Proceedings of ICALP 2015.
http://www.siam.org/journals/sidma/33-1/M114003.html
Funding: The research of the authors was supported by the European Research Council under the European Unions Seventh Framework Programme (FP/2007-2013)/ERC grant Agreements 267959, 338077, and 306992, and NFR MULTIVAL project.

[†]Department of Informatics, University of Bergen, N-5020 Norway (fomin@ii.uib.no, fahad.panolan@ii.uib.no).
[‡]Department of Information and Computer Science, Aalto University, Espoo, FI-00076 Finland (petteri.kaski@aalto.fi).
[§]Department of Computer Science, University of California Santa Barbara, Santa Barbara, CA 93106 (daniello@ucsb.edu).
[¶]The Institute of Mathematical Sciences, HBNI, Chennai, 600 113 India, and Department of Informatics, University of Bergen, Bergen, N-5020 Norway (saket@imsc.res.in).

overview of the results and applications of the Steiner Tree problem. Steiner Tree is known to be APX-complete, even when the graph is complete and all edge weights are either 1 or 2 [2]. On the other hand, the problem admits a constant factor approximation algorithm, and the currently best such algorithm (after a long chain of improvements) is due to Byrka et al. and has approximation ratio $\ln 4 + \varepsilon < 1.39$ [6].

Steiner Tree is a fundamental problem in parameterized algorithms [7, 8, 10, 19]. The classic algorithm for Steiner Tree of Dreyfus and Wagner [9] from 1971 might well be the first parameterized algorithm for *any* problem. The study of parameterized algorithms for Steiner Tree has led to the design of important techniques, such as fast subset convolution [3] and the use of branching walks [18]. Research on the parameterized complexity of Steiner Tree is still ongoing, with very recent significant advances for the planar version of the problem [20, 21, 17].

Algorithms for Steiner Tree are frequently used as a subroutine in fixed-parameter tractable (FPT) algorithms for other problems; examples include vertex cover problems [15], near-perfect phylogenetic tree reconstruction [4], and connectivity augmentation problems [1].

Motivation and earlier work. For more than 30 years, the fastest FPT algorithm for Steiner Tree was the $3^k \cdot \log W \cdot n^{\mathcal{O}(1)}$-time dynamic programming algorithm by Dreyfus and Wagner [9]. Fuchs et al. [14] gave an improved algorithm with running time $\mathcal{O}((2+\varepsilon)^k n^{f(1/\varepsilon)} \log W)$. For the unweighted version of the problem, Björklund et al. [3] gave a $2^k n^{\mathcal{O}(1)}$-time algorithm. All of these algorithms are based on dynamic programming and use exponential space.

Algorithms with high space complexity are in practice more constrained because the amount of memory is not easily scaled beyond hardware constraints, whereas time complexity can be alleviated by allowing for more time for the algorithm to finish. Furthermore, algorithms with low space complexity are typically easier to parallelize and more cache-friendly. These considerations motivate a quest for algorithms whose memory requirements scale polynomially in the size of the input, even if such algorithms may be slower than their exponential space counterparts. The first polynomial space $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$-time algorithm for the unweighted Steiner Tree problem is due to Nederlof [18]. This algorithm runs in time $2^k n^{\mathcal{O}(1)}$, matching the running time of the best known exponential space algorithm. Nederlof's algorithm can be extended to the weighted case; unfortunately this comes at the cost of an $\mathcal{O}(W)$ factor both in the time and the space complexity. Lokshtanov and Nederlof [16] showed that the $\mathcal{O}(W)$ factor can be removed from the space bound, but with a factor $\mathcal{O}(W)$ in the running time. The algorithm of Lokshtanov and Nederlof [16] runs in $2^k \cdot n^{\mathcal{O}(1)} \cdot W$ time and uses $n^{\mathcal{O}(1)} \log W$ space. Note that both the algorithm of Nederlof [18] and the algorithm of Lokshtanov and Nederlof [16] have an $\mathcal{O}(W)$ factor in their running time. Thus the running time of these algorithms depends exponentially on the input size, and therefore these algorithms are not FPT algorithms for weighted Steiner Tree.

For weighted Steiner Tree, the only known polynomial space FPT algorithm has a $2^{\mathcal{O}(k \log k)}$ running time dependence on the parameter k. This algorithm follows from combining a $(27/4)^k \cdot n^{\mathcal{O}(\log k)} \cdot \log W$ time, polynomial space algorithm by Fomin et al. [11] with the Dreyfus–Wagner algorithm. Indeed, one runs the algorithm of Fomin et al. [11] if $n \le 2^k$, and the Dreyfus–Wagner algorithm if $n > 2^k$. If $n \le 2^k$, the running time of the algorithm of Fomin et al. is bounded from above by $2^{\mathcal{O}(k \log k)}$. When $n > 2^k$, the Dreyfus–Wagner algorithm becomes a polynomial time (and space) algorithm.

Prior to this work, the existence of a polynomial space algorithm with running time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)} \cdot \log W$, i.e., a single-exponential time polynomial space FPT algorithm, was an open problem asked explicitly in [11, 16].

Contributions and methodology. The starting point of our present algorithm is the $(27/4)^k \cdot n^{\mathcal{O}(\log k)} \cdot \log W$ time, polynomial-space algorithm by Fomin et al. [11]. This algorithm crucially exploits the possibility for *balanced separation* (cf. Lemma 2.1). Specifically, an optimal Steiner tree ST can be partitioned into two trees ST_1 and ST_2 containing the terminal sets T_1 and T_2, respectively, so that the following three properties are satisfied:

(1) The two trees share exactly one vertex v and no edges.
(2) Neither of the two trees ST_1 or ST_2 contains more than a 2/3 fraction of the terminal set T.
(3) The tree ST_1 is an optimal Steiner tree for the terminal set $T_1 \cup \{v\}$, and ST_2 is an optimal Steiner tree for the terminal set $T_2 \cup \{v\}$.

Conversely, to find the optimal tree ST for the terminal set T, it suffices to (1) guess the vertex v, (2) partition T into T_1 and T_2, and (3) recursively find optimal trees for the terminal sets $T_1 \cup \{v\}$ and $T_2 \cup \{v\}$. The most unbalanced partition (T_1, T_2) of T in (2) has one block containing 1/3 fraction of T and another block containing 2/3 fraction of T. Since there are n choices for v, and $\binom{k}{k/3}$ ways to partition T into two sets T_1 and T_2 such that $|T_1| = |T|/3$, the running time of the algorithm is essentially governed by the recurrence

$$T(n,k) \leq n \cdot \binom{k}{k/3} \cdot (T(n,k/3) + T(n,2k/3)). \tag{1}$$

Unraveling (1) gives the $(27/4)^k \cdot n^{\mathcal{O}(\log k)} \cdot \log W$ upper bound for the running time, and it is easy to see that the algorithm runs in polynomial space. However, this algorithm is not an FPT algorithm because of the $n^{\mathcal{O}(\log k)}$ factor in the running time.

The factor $n^{\mathcal{O}(\log k)}$ is incurred by the factor n in (1), which in turn originates from the need to iterate over all possible choices for the vertex v in each recursive call. In effect the recursion tracks an $\mathcal{O}(\log k)$-sized set S of *split vertices* (together with a subset T' of the terminal vertices T) when it traverses the recursion tree from the root to a leaf.

The key idea in our new algorithm is to redesign the recurrence for optimal Steiner trees so that we obtain control over the size of S using an alternation between

1. balanced separation steps (as described above) and
2. novel *resplitting* steps that maintain the size of S at no more than three vertices throughout the recurrence.

In essence, a resplit takes a set S of size 3 and splits that set into three sets of size 2 by combining each element in S with an arbitrary vertex v, while at the same time splitting the terminal set T' into three parts in all possible (not only balanced) ways. While the combinatorial intuition for resplitting is elementary (cf. Lemma 2.2 below), the implementation and analysis requires a somewhat careful combination of ingredients.

Namely, to run in polynomial space, it is not possible to use extensive amounts of memory to store intermediate results to avoid recomputation. Yet, if no memorization is used, the novel recurrence does not lead to an FPT algorithm, let alone to a single-exponential FPT algorithm. Thus neither a purely dynamic programming nor a purely recursive implementation will lead to the desired algorithm. *A combination of the two* *will, however, give a single-exponential time algorithm that uses polynomial space.*

Roughly, our approach is to employ recursive evaluation over subsets T' of the terminal set T, but each recursive call with T' will compute and return the optimal solutions for every possible set S of split vertices. Since by resplitting we have arranged that S always has size at most 3, this hybrid evaluation approach will use polynomial space. Since each recursive call on T' yields the optimum weights for every possible S, we can use dynamic programming to efficiently combine these weights so that single-exponential running time results.

In precise terms, our main result is as follows.

THEOREM 1. STEINER TREE *can be solved in time* $\mathcal{O}(7.97^k n^4 \log(nW))$ *using space* $\mathcal{O}(n^3 \log(nW) \log k)$.

Whereas our main result seeks to optimize the polynomial dependency in n for both the running time and space usage, it is possible to trade between polynomial dependency in n and the single-exponential dependency in k to obtain faster running time as a function of k, but at the cost of increased running time and space usage as a function of n. In particular, we can use larger (but still constant-size) sets S to avoid recomputation and to arrive at a somewhat faster algorithm.

THEOREM 2. *There exists a polynomial-space algorithm for* STEINER TREE *running in* $\mathcal{O}(6.751^k n^{\mathcal{O}(1)} \log W)$ *time.*

Finally, by using more ideas, like introducing *even* balanced separators of larger cardinality in trees and generalizing our STEINER TREE algorithm on SUBSET STEINER FOREST, we obtain the following theorem.

THEOREM 3. *For any* $\epsilon > 0$ *there is an* $n^{\mathcal{O}(f(\epsilon))} \log W$ *space* $4^{(1+\epsilon)k} n^{\mathcal{O}(f(\epsilon))} \log W$ *time algorithm for* STEINER TREE*, where* f *is a computable function depending only on* ϵ.

Notice that to get polynomial space parameterized single-exponential time algorithms for the unweighted version of STEINER TREE, it is enough to substitute uniform weight to all of the edges (for example, the weight of each edge is 2) in the above theorems.

2. Preliminaries.

Given a graph G, we write $V(G)$ and $E(G)$ for the set of vertices and edges of G, respectively. For subgraphs G_1, G_2 of G, we write $G_1 + G_2$ for the subgraph of G with vertex set $V(G_1) \cup V(G_2)$ and edge set $E(G_1) \cup E(G_2)$. For a graph G, $S \subseteq V(G)$ and $v \in V(G)$, we use $G - S$ and $G - v$ to denote the induced subgraphs $G[V(G) \setminus S]$ and $G[V(G) \setminus \{v\}]$, respectively. The minimum weight of a Steiner tree of G on terminals T is denoted by $st_G(T)$. When graph G is clear from the context, we will simply write $st(T)$. For a set U and a nonnegative integer i, we use $\binom{U}{i}$ and $\binom{U}{\leq i}$ to denote the set of all subsets of U of size exactly i and the set of all subsets of U of size at most i, respectively.

Partitions. A partition of a nonempty set U is a collection of nonempty, pairwise disjoint subsets of U whose union is U. The subsets are called *blocks*. For a set U, we write $U_1 \uplus U_2 \uplus \cdots \uplus U_\ell = U$ if $U_1, U_2, \ldots, U_\ell$ is a partition of U. For a set U and a positive integer i, we use $\mathbf{P}(U)$ and $\mathbf{P}_i(U)$ to denote the set of all partitions of U and the set of all partitions of U into i blocks, respectively. For convenience we use $\mathbf{P}(\emptyset) = \{\emptyset\}$. For a set U and a constant $\epsilon \geq 0$, we use $\mathcal{B}_\epsilon(U)$ to denote the set of all partitions (U_1, U_2) of U into two blocks such that $|U_1|, |U_2| \leq \left(\frac{1}{2} + \epsilon\right)|U|$. For a set U, a subset U' of U, and a partition $P \in \mathbf{P}(U)$, we use $P[U']$ to denote the restriction of partition P on the set U', i.e., the blocks in $P[U']$ are obtained by deleting $U \setminus U'$

from the blocks of P. For a set U and partitions $P_1, P_2 \in \mathbf{P}(U)$, we say partition P_1 *refines* partition P_2, denoted by $P_1 \preceq P_2$, if every block of P_1 is contained in some block of P_2. We also use $P_1 \preceq P_2$ if P_1 is a restriction of a partition in $\mathbf{P}(U)$ which *refines* partition P_2. That is, for a set U, a subset U' of U, and partitions $P_1 \in \mathbf{P}(U')$ and $P_2 \in \mathbf{P}(U)$, we write $P_1 \preceq P_2$ if each block of P_1 is contained in some block of P_2. For two partitions P_1 and P_2 in $\mathbf{P}(U)$, the *join* of P_1 and P_2, denoted by $P_1 \sqcup P_2$, is the smallest (with respect to $\preceq$) partition P refined by both P_1 and P_2. For a graph G, we use P_G to denote the partition $\{V(C) \mid C \text{ is a connected component of } G\}$ of $V(G)$.

Separation and resplitting. A set of nodes S is called an *α-separator* of a graph G, $0 < \alpha \leq 1$, if the vertex set $V(G) \setminus S$ can be partitioned into sets V_L and V_R of size at most αn each, such that no vertex of V_L is adjacent to any vertex of V_R. We next define a similar notion, which turns out to be useful for Steiner trees. Given a Steiner tree ST on terminals T, an α-Steiner separator S of ST is a subset of nodes which partitions $ST - S$ into two forests $\mathcal{R}_1$ and $\mathcal{R}_2$, each one containing at most αk terminals from T.

LEMMA 2.1 (separation). (*See* [5, 11].) *Every Steiner tree ST on terminal set T, $|T| \geq 3$, has a $2/3$-Steiner separator $S = \{s\}$ of size one.*

The following easy lemma enables us to control the size of the split set at no more than three vertices.

LEMMA 2.2 (resplitting). *Let F be a tree and $S \in \binom{V(F)}{3}$. Then there is a vertex $v \in V(F)$ such that each connected component in $F - v$ contains at most one vertex of S.*

Proof. Let $S = \{s_1, s_2, s_3\}$. Let P_1 be the unique path between s_1 and s_3 in the tree F. Let P_2 be the unique path between s_3 and s_2 in the tree F. If P_1 and P_2 are edge-disjoint, then $V(P_1) \cap V(P_2) = \{s_3\}$ and P_1P_2 is the unique path between s_1 and s_2. Thus any connected component in $G - s_3$ will not contain both s_1 and s_2. In this case s_3 is the required vertex. Suppose $V(P_1) \cap V(P_2) \neq \{s_3\}$. For a path $P = u_1u_2 \ldots u_\ell$ in a graph G, let $\overleftarrow{P}$ denote the reverse path $u_\ell u_{\ell-1} \ldots u_1$. Consider the unique path $\overleftarrow{P_1}$ between s_3 and s_1, which is the reverse of the path P_1. Since F is a tree, these paths $\overleftarrow{P_1}$ and P_2 will be of the form $P_1 = Q\overleftarrow{P_1}'$ and $P_2 = QP_2'$. Note that Q is a path starting at s_3. Let w be the last vertex in the path Q. Since F is a tree, $V(\overleftarrow{P_1}') \cap V(P_2') = \{w\}$. Now consider the graph $G - w$. Any connected component in $G - w$ will not contain more than one vertex from $\{s_1, s_2, s_3\}$, because the unique path between any pair of vertices in $\{s_1, s_2, s_3\}$ passes through w. ∎

3. Algorithm.

In this section, we design an algorithm for STEINER TREE which runs in $\mathcal{O}(7.97^k n^4 \log(nW))$ time using $\mathcal{O}(n^3 \log(nW) \log k)$ space. Most algorithms for STEINER TREE, including ours, are based on recurrence relations that reduce finding an optimal Steiner tree to finding optimal Steiner trees in the same graph, but with a smaller terminal set. We first define two functions f_i for $i \in \{2, 3\}$. Each function $f_i, i \in \{2, 3\}$ takes as input a vertex set S of size at most i (the split set defined in the introduction) and a subset T' of T. The function $f_i(S, T')$ returns a real number. We will define the functions using recurrence relations, and then prove that $f_i(S, T')$ is exactly $st_G(T' \cup S)$. These two functions are alternatively invoked by our algorithm. Later we define two more functions f_i for $i \in \{0, 1\}$, and they are invoked only in the first two recursive steps of our algorithm.

For $T' \subseteq T$, $i \in \{2, 3\}$, and $S \in \binom{V(G)}{\leq i}$, we define $f_i(S, T')$ as follows. When

$|T'| \leq 2$, we set $f_i(S,T') = st_G(T' \cup S)$. For $|T'| \geq 3$, we define $f_i(S,T')$ using the following recurrences.

Separation.

$$f_2(S,T') = \min_{(T_1,T_2)\in\mathcal{B}_{\frac{1}{6}}(T')} \min_{\substack{v\in V(G)\\ S_1\uplus S_2=S}} f_3\bigl(S_1\cup\{v\},T_1\bigr) + f_3\bigl(S_2\cup\{v\},T_2\bigr). \tag{2}$$

Resplitting.

$$f_3(S,T') = \min_{(T_1,T_2,T_3)\in\mathbf{P}_3(T')} \min_{\substack{S_1\uplus S_2\uplus S_3=S\\ |S_1|,|S_2|,|S_3|\leq 1\\ v\in V(G)}} \sum_{r=1}^{3} f_2\bigl(S_r\cup\{v\},T_r\bigr). \tag{3}$$

The recurrences (2) and (3) are recurrence relations for Steiner Tree as proved in the following lemma.

Lemma 3.1. *For all $T' \subseteq T$, $i \in \{2,3\}$, and $S \in \binom{V(G)}{\leq i}$ it holds that $f_i(S,T') = st_G(T' \cup S)$.*

Proof. We prove the lemma using induction on $|T'|$. For the base case $|T'| \leq 2$, the lemma holds by the definition of f_i. For the inductive step, let us assume that the lemma holds for all T'' of size less than j. We proceed to show that $f_i(S,T') = st_G(T' \cup S)$ for all $T' \subseteq T$ with $|T'| = j$. We split into cases based on i and in each case establish the inequalities $f_i(S,T') \leq st_G(T' \cup S)$ and $f_i(S,T') \geq st_G(T' \cup S)$ to conclude equality.

Case 1: $i = 2$. By (2), we know that there are a vertex $v \in V(G)$, a partition $S_1 \uplus S_2 = S$, and another partition $(T_1,T_2) \in \mathcal{B}_{1/6}(T')$ such that $f_i(S,T') = f_{i+1}(S_1 \cup \{v\},T_1) + f_{i+1}(S_2 \cup \{v\},T_2)$. Since $(T_1,T_2) \in \mathcal{B}_{1/6}(T')$ and $|T'| \geq 3$, we have that $|T_1|,|T_2| < |T'|$. Then by induction hypothesis,

$$f_{i+1}(S_1 \cup \{v\},T_1) = st_G(T_1 \cup S_1 \cup \{v\})$$

and

$$f_{i+1}(S_2 \cup \{v\},T_2) = st_G(T_2 \cup S_2 \cup \{v\}).$$

So we have that

$$f_i(S,T') = st_G(T_1 \cup S_1 \cup \{v\}) + st_G(T_2 \cup S_2 \cup \{v\}).$$

Let ST_1 be an optimum Steiner tree for the set of terminals $T_1 \cup S_1 \cup \{v\}$ and let ST_2 be an optimum Steiner tree for the set of terminals $T_2 \cup S_2 \cup \{v\}$. Note that $ST_1 + ST_2$ is a connected subgraph containing $T_1 \cup T_2 \cup S$ and

$$w(E(ST_1 + ST_2)) \leq st_G(T_1 \cup S_1 \cup \{v\}) + st_G(T_2 \cup S_2 \cup \{v\}).$$

This implies that

$$st_G(T' \cup S) \leq w(E(ST_1 + ST_2)) \leq st_G(T_1 \cup S_1 \cup \{v\}) + st_G(T_2 \cup S_2 \cup \{v\}) = f_i(S,T').$$

Hence $f_i(S,T') \geq st_G(T' \cup S)$.

Conversely, let ST be an optimum Steiner tree for the set of terminals $T' \cup S$. Thus ST is also a Steiner tree for the set of terminals T'. Hence by Lemma 2.1, we know that there is a 2/3-Steiner separator $\{v\}$ of size one for the tree ST on the terminal set T'. Let F_1 and F_2 be two forests created by the separator $\{v\}$, such that $V(F_r) \cap T' \leq 2|T'|/3$ for each $1 \leq r \leq 2$. Let $T_r = V(F_r) \cap T'$, $1 \leq r \leq 2$. If $v \in T'$ and $|T_1| \leq |T_2|$, then we replace T_1 with $T_1 \cup \{v\}$. If $v \in T'$ and $|T_1| > |T_2|$, then we replace T_2 with $T_2 \cup \{v\}$. Note that (T_1, T_2) is a partition of T'. Since $\{v\}$ is a 2/3-Steiner separator and $|T'| \geq 3$, we have that

$$|T_1|, |T_2| \leq 2|T'|/3 \leq \left(\frac{1}{2} + \frac{1}{6}\right)|T'| < |T'|.$$

Hence $(T_1, T_2) \in \mathcal{B}_{1/6}(T')$. For any $r \in \{1, 2\}$, let $ST_r = ST[V(F_r) \cup \{v\}]$ and let $S_r = V(ST_r) \cap S$. Thus

$$f_{i+1}(S_1 \cup \{v\}, T_1) + f_{i+1}(S_2 \cup \{v\}, T_2) \geq f_i(S, T').$$

Note that $ST_1 = ST[V(F_1) \cup \{v\}]$ and $ST_2 = ST[V(F_2) \cup \{v\}]$ are subtrees of ST. By the induction hypothesis, we have that $f_{i+1}(S_1 \cup \{v\}, T_1) = st_G(T_1 \cup S_1 \cup \{v\})$ and $f_{i+1}(S_2 \cup \{v\}, T_2) = st_G(T_2 \cup S_2 \cup \{v\})$. Since ST_1 and ST_2 are trees containing $T_1 \cup S_1 \cup \{v\}$ and $T_2 \cup S_2 \cup \{v\}$, respectively, we have

$$\begin{aligned} w(E(ST_1)) + w(E(ST_2)) &\geq st_G(T_1 \cup S_1 \cup \{v\}) + st_G(T_2 \cup S_2 \cup \{v\}) \\ &= f_{i+1}(S_1 \cup \{v\}, T_1) + f_{i+1}(S_2 \cup \{v\}, T_2) \geq f_i(S, T'). \end{aligned}$$

Since $V(ST_1) \cap V(ST_2) = \{v\}$ and $T' \cup S \subseteq V(ST_1) \cup V(ST_2)$, we have that $st_G(T' \cup S) = w(E(ST_1)) + w(E(ST_2))$. Thus $f_i(S, T') \leq st_G(T' \cup S)$.

Case 2: $i = 3$. By (3), there are $v \in V(G)$, $S_1, S_2, S_3 \in \binom{S}{\leq 1}$, $S_1 \uplus S_2 \uplus S_3 = S$, and a partition $(T_1, T_2, T_3) \in \mathbf{P}_3(T')$ such that

$$f_3(S, T') = \sum_{r=1}^{3} f_2(S_r \cup \{v\}, T_r).$$

We have shown (in Case 1) that $f_2(S_r \cup \{v\}, T_r) = st_G(T_r \cup S_r \cup \{v\})$ for all $1 \leq r \leq 3$. Therefore, $f_3(S, T') = \sum_{r=1}^{3} st_G(T_r \cup S_r \cup \{v\})$. Let ST_r be an optimum Steiner tree for the set of terminals $T_r \cup S_r \cup \{v\}$ for all r. Note that $ST_1 + ST_2 + ST_3$ is a connected subgraph containing $T_1 \cup T_2 \cup T_3 \cup S$ and

$$w(E(ST_1 + ST_2 + ST_3)) \leq \sum_{r=1}^{3} st_G(T_r \cup S_r \cup \{v\}).$$

Thus

$$st_G(T' \cup S) \leq w(E(ST_1 + ST_2 + ST_3)) \leq \sum_{r=1}^{3} st_G(T_r \cup S_r \cup \{v\}) = f_3(S, T').$$

Hence $f_3(S, T') \geq st_G(T' \cup S)$.

Conversely, let ST be an optimum Steiner tree for the set of terminals $T' \cup S$. By Lemma 2.2, there is a vertex $v \in V(ST)$ such that each connected component C in $ST - v$ contains at most one vertex from S. Let $\mathcal{C}_1, \mathcal{C}_2$, and $\mathcal{C}_3$ be a partition of connected components of $ST - v$ such that $|V(\mathcal{C}_r) \cap S| \leq 1$ for all $1 \leq r \leq 3$. For each r, let $T_r = T' \cap V(\mathcal{C}_r)$ and $ST_r = ST[V(\mathcal{C}_r) \cup \{v\}]$. If $v \in T'$, then we replace T_1 with

$T_1 \cup \{v\}$. Note that (T_1, T_2, T_3) is a partition of T'. Hence $(T_1, T_2, T_3) \in \mathbf{P}_3(T')$. For each r, let $S_r = (S \setminus \{v\}) \cap V(ST_r)$. Since each $\mathcal{C}_r$ contains at most one vertex from S, $|S_r| \leq 1$. This implies

$$\sum_{r=1}^{3} f_2(S_r \cup \{v\}, T_r) \geq f_3(S, T').$$

Note that $ST_r = ST[V(\mathcal{C}_r) \cup \{v\}]$ is a tree for each r. Since $V(\mathcal{C}_1) \cup V(\mathcal{C}_2) \cup V(\mathcal{C}_3) \cup \{v\} = V(ST)$ and for all $1 \leq r_1 \neq r_2 \leq 3$ it holds that $V(\mathcal{C}_{r_1}) \cap V(\mathcal{C}_{r_2}) = \emptyset$, we have

$$\begin{aligned} st_G(T' \cup S) &= w(E(ST)) \\ &= \sum_{r=1}^{3} w(E(ST_r)) \geq \sum_{r=1}^{3} f_2(S_r \cup \{v\}, T_r) \geq f_3(S, T'). \end{aligned}$$

Thus $f_3(S, T') \leq st_G(T' \cup S)$. ◻

We would like to mention that 3 is the smallest size of S for which we can do resplitting and reduce the size of the separator (i.e., first argument in the function f_i). Our algorithm uses (2) and (3) repeatedly to compute the weight of an optimum Steiner tree. Notice that $f_2(\emptyset, T)$ is equal to the weight of an optimum Steiner tree. Since f_3 runs over all partitions of size 3, unlike the balanced partitions in the case of f_2, f_3 is costlier for the running time of the algorithm. Also notice that $f_2(\emptyset, T)$ invokes two instances of f_3 with the first argument being size one sets in each instance. In fact for those two instances of f_3, we do not have to make a *resplitting* step. Thus towards designing a faster algorithm we define two more functions f_0 and f_1 which are defined similarly to f_2. For $T' \subseteq T$, $i \in \{0, 1\}$, and $S \in \binom{V(G)}{\leq i}$, we define $f_i(S, T')$ as follows. When $|T'| \leq 2$, we set $f_i(S, T') = st_G(T' \cup S)$. For $|T'| \geq 3$, we define $f_i(S, T')$ using the following recurrence:

$$f_i(S, T') = \min_{(T_1, T_2) \in \mathcal{B}_{\frac{1}{6}}(T')} \min_{\substack{v \in V(G) \\ S_1 \uplus S_2 = S}} f_{i+1}\big(S_1 \cup \{v\}, T_1\big) + f_{i+1}\big(S_2 \cup \{v\}, T_2\big). \tag{4}$$

LEMMA 3.2. *For all $T' \subseteq T$, $i \in \{0, 1\}$, and $S \in \binom{V(G)}{\leq i}$ it holds that $f_i(S, T') = st_G(T' \cup S)$.*

Proof. By substituting $i = 1$ in Case 1 of the proof of Lemma 3.1 and the fact that $f_2(S_1 \cup \{v\}, T_1) = st_G(T_1 \cup S_1 \cup \{v\})$ and $f_2(S_2 \cup \{v\}, T_2) = st_G(T_2 \cup S_2 \cup \{v\})$ (by Lemma 3.1), we conclude that $f_1(S, T') = st_G(T' \cup S)$. Again, by arguments similar to that of the proof for the case $i = 1$, we conclude that $f_0(S, T') = st_G(T' \cup S)$. ◻

A naïve way of turning the recurrences into an algorithm would be to simply make one recursive procedure for each f_i, and apply (4), (2), and (3) directly. However, this would result in a factor $n^{\mathcal{O}(\log k)}$ in the running time, which we seek to avoid. The reason to get a factor $n^{\mathcal{O}(\log k)}$ in the running time of a naïve implementation is the number of branches of each node in the recurrence tree is at least $n^{\mathcal{O}(1)}$ (because the number of choices of separators for each recurrence is $n^{\mathcal{O}(1)}$) and the depth of the recurrence tree is $\mathcal{O}(\log k)$ (because of the balanced partition in (2)). Like the naïve approach, our algorithm has one recursive procedure F_i for each function f_i. The procedure F_i takes as input a subset T' of the terminal set, and returns an array that, for every $S \in \binom{V(G)}{\leq i}$, contains $f_i(S, T')$.

The key observation is that if we seek to compute $f_i(S, T')$ for a fixed T' and *all* choices of $S \in \binom{V(G)}{\leq i}$ using recurrence (4) or (2) or (3), we should not just iterate over

Algorithm 1: Implementation of procedure F_i for $i \in \{0, 1, 2\}$.

```
Input: $T' \subseteq T$
Output: $st_G(T' \cup S)$ for all $S \in \binom{V(G)}{\leq i}$
if $|T'| \leq 2$ then
    for $S \in \binom{V(G)}{\leq i}$ do
        $A[S] \leftarrow st_G(T' \cup S)$ (compute using the Dreyfus–Wagner algorithm)
    end
    return $A$
end
for $S \in \binom{V(G)}{\leq i}$ do
    $A[S] \leftarrow \infty$
end
for $T_1, T_2 \in \mathcal{B}_{1/6}(T')$ do
    $A_1 \leftarrow F_{i+1}(T_1)$
    $A_2 \leftarrow F_{i+1}(T_2)$
    for $S_1 \uplus S_2 \in \binom{V(G)}{\leq i}$ such that $|S_2| \leq |S_1|$ and $v \in V(G)$ do
        if $A[S_1 \uplus S_2] > A_1[S_1 \cup \{v\}] + A_2[S_2 \cup \{v\}]$ then
            $A[S_1 \uplus S_2] \leftarrow A_1[S_1 \cup \{v\}] + A_2[S_2 \cup \{v\}]$
        end
    end
end
return $A$.
```

every choice of S and then apply the recurrence to compute $f_i(S, T')$; it is much faster to compute all of the entries of the return array of F_i simultaneously, by iterating over every eligible partition of T', making the required calls to F_{i+1} (or F_{i-1} if we are using recurrence (3)), and updating the appropriate array entries to yield the return array of F_i. Next, we give pseudocode for the procedures F_0, F_1, F_2, F_3.

The procedure F_i for $0 \leq i \leq 2$ operates as follows. (See Algorithm 1.) Let $T' \subseteq T$ be the input to the procedure F_i. If $|T'| \leq 2$, then F_i computes $st_G(T' \cup S)$ for all $S \in \binom{V(G)}{\leq i}$ using the Dreyfus–Wagner algorithm and returns these values. The procedure F_i has an array A indexed by $S \in \binom{V(G)}{\leq i}$. At the end of the procedure F_i, $A[S]$ will contain the value $st_G(T' \cup S)$ for each $S \in \binom{V(G)}{\leq i}$. For each $(T_1, T_2) \in \mathcal{B}_{1/6}(T')$ (line 10), F_i calls $F_{i+1}(T_1)$ and $F_{i+1}(T_2)$ and it returns two sets of values $\{f_{i+1}(S, T_1) \mid S \in \binom{V(G)}{\leq i+1}\}$ and $\{f_i(S, T_2) \mid S \in \binom{V(G)}{\leq i}\}$, respectively. Let A_1 and A_2 be two arrays used to store the return values of $F_{i+1}(T_1)$ and $F_{i+1}(T_2)$, respectively. That is, $A_1[S] = f_{i+1}(S, T_1)$ for each $S \in \binom{V(G)}{\leq i+1}$ and $A_2[S'] = f_{i+1}(S', T_2)$ for each $S' \in \binom{V(G)}{\leq i+1}$. Now we update A as follows. For each $S_1 \uplus S_2 \in \binom{V(G)}{\leq i}$ and $v \in V(G)$ (line 13), if $A[S_1 \uplus S_2] > A_1[S_1 \cup \{v\}] + A_2[S_2 \cup \{v\}]$, then we update the entry $A[S_1 \uplus S_2]$, with the value $A_1[S_1 \cup \{v\}] + A_2[S_2 \cup \{v\}]$. So at the end of the inner **for** loop, $A[S]$ contains the value

$$\min_{\substack{v \in V(G) \\ S_1 \uplus S_2 = S}} f_{i+1}(S_1 \cup \{v\}, T_1) + f_{i+1}(S_2 \cup \{v\}, T_2).$$

Since we do have a outer **for** loop which runs over $(T_1, T_2) \in \mathcal{B}_{1/6}(T')$, we have

Algorithm 2: Implementation of procedure F_3.

```
Input: $T' \subseteq T$
Output: $st_G(T' \cup S)$ for all $S \in \binom{V(G)}{\leq 3}$
if $|T'| \leq 2$ then
    for $S \in \binom{V(G)}{\leq 3}$ do
        $A[S] \leftarrow st_G(T' \cup S)$ (compute using the Dreyfus–Wagner algorithm)
    end
    return $A$
end
for $S \in \binom{V(G)}{\leq 3}$ do
    $A[S] \leftarrow \infty$
end
for $T_1, T_2, T_3 \in \mathbf{P}_3(T')$ do
    $A_1 \leftarrow F_2(T_1)$
    $A_2 \leftarrow F_2(T_2)$
    $A_3 \leftarrow F_2(T_3)$
    for $S_1, S_2, S_3 \in \binom{V(G)}{\leq 1}$ and $v \in V(G)$ do
        if $A[S_1 \cup S_2 \cup S_3] > A_1[S_1 \cup \{v\}] + A_2[S_2 \cup \{v\}] + A_3[S_3 \cup \{v\}]$ then
            $A[S_1 \cup S_2 \cup S_3] \leftarrow A_1[S_1 \cup \{v\}] + A_2[S_2 \cup \{v\}] + A_3[S_3 \cup \{v\}]$
        end
    end
end
return $A$.
```

updated $A[S]$ with

$$\min_{(T_1,T_2)\in\mathcal{B}_{1/6}(T')} \min_{\substack{v\in V(G)\\ S_1 \uplus S_2 = S}} f_{i+1}(S_1 \cup \{v\}, T_1) + f_{i+1}(S_2 \cup \{v\}, T_2)$$

at the end of the procedure. Then F_i will return A.

The procedure F_3 works as follows. (See Algorithm 2.) Let $T' \subseteq T$ be the input to the procedure F_3. If $|T'| \leq 2$, then F_3 computes $st_G(T' \cup S)$ for all $S \in \binom{V(G)}{\leq 3}$ using the Dreyfus–Wagner algorithm and returns these values. The procedure F_3 has an array A indexed by $S \in \binom{V(G)}{\leq 3}$. At the end of the procedure F_3, $A[S]$ will contain the value $st_G(T' \cup S)$ for all $S \in \binom{V(G)}{\leq 3}$. For each $(T_1, T_2, T_3) \in \mathbf{P}_3(T')$ (line 10), F_3 calls $F_2(T_1)$, $F_2(T_2)$, and $F_2(T_3)$, and it returns three sets of values $\{f_2(S, T_1) \mid S \in \binom{V(G)}{\leq 2}\}$, $\{f_2(S, T_2) \mid S \in \binom{V(G)}{\leq 2}\}$ and $\{f_2(S, T_3) \mid S \in \binom{V(G)}{\leq 2}\}$, respectively. Let A_1, A_2, and A_3 be three arrays used to store the outputs of $F_2(T_1)$, $F_2(T_2)$, and $F_2(T_3)$, respectively. That is, $A_r[S] = f_2(S, T_r)$ for $r \in \{1, 2, 3\}$. Now we update A as follows. For each $S_1, S_2, S_3 \in \binom{V(G)}{\leq 1}$ and $v \in V(G)$ (line 14), if $A[S_1 \cup S_2 \cup S_3] > A_1[S_1 \cup \{v\}] + A_2[S_2 \cup \{v\}] + A_3[S_3 \cup \{v\}]$, then we update the entry $A[S_1 \cup S_2 \cup S_3]$, with the value $A_1[S_1 \cup \{v\}] + A_2[S_2 \cup \{v\}] + A_3[S_3 \cup \{v\}]$. So at the end of the inner **for** loop, $A[S]$ contains the value

$$\min_{\substack{S_1\cup S_2\cup S_3=S\\ |S_1|,|S_2|,|S_3|\leq 1\\ v\in V(G)}} \sum_{r=1}^{3} f_2(S_r \cup \{v\}, T_r).$$

Since we do have a outer **for** loop which runs over $(T_1, T_2, T_3) \in \mathbf{P}_3(T')$, we have updated $A[S]$ with

$$\min_{(T_1,T_2,T_3)\in\mathbf{P}_3(T')} \min_{\substack{S_1\cup S_2\cup S_3=S\\ |S_1|,|S_2|,|S_3|\le 1\\ v\in V(G)}} \sum_{r=1}^{3} f_2(S_r \cup \{v\}, T_r)$$

at the end of the procedure. Then F_3 will return A as the output.

In what follows, we prove the correctness and analyze the running time and memory usage of the call to the procedure $F_0(T)$.

LEMMA 3.3. *For every $i \le 3$, and every set $T' \subseteq T$, the procedure $F_i(T')$ outputs an array that contains $f_i(S, T')$ for every $S \in \binom{V(G)}{\le i}$.*

Proof. Correctness of the lemma follows directly by an induction on $|T|$. Indeed, assuming that the lemma statement holds for the recursive calls made by the procedure F_i, it is easy to see that each entry of the output table is exactly equal to the right-hand side of recurrence (4) or (2) (recurrence (3) in the case of F_3). □

OBSERVATION 3.1. *The recursion tree of the procedure $F_0(T)$ has depth $\mathcal{O}(\log k)$.*

Proof. For every $i \le 2$ the procedure $F_i(T')$ only makes recursive calls to $F_{i+1}(T'')$ where $|T''| \le 2|T'|/3$. The procedure $F_3(T')$ makes recursive calls to $F_2(T'')$ where $|T''| \le |T'|$. Therefore, on any root-leaf path in the recursion tree, the size of the considered terminal set T' drops by a constant factor every second step. When the terminal set reaches size at most 2, no further recursive calls are made. Thus any root-leaf path has length at most $\mathcal{O}(\log k)$. □

LEMMA 3.4. *The procedure $F_0(T)$ uses $\mathcal{O}(n^3 \log(nW) \log k)$ space.*

Proof. As each F_i, $i \in \{0, 1, 2, 3\}$, is a recursive procedure once its recursive call is returned, the memory used by the recursive procedure can be reused. Thus to upper bound the space used by the procedure $F_0(T)$, it is sufficient to upper bound the memory usage of every individual recursive call, not taking into account the memory used by its recursive calls, and then multiply this upper bound by the depth of the recursion tree.

Each individual recursive call will at any point of time keep a constant number of tables, each containing at most $\mathcal{O}(n^3)$ entries. Each entry is a number less than or equal to nW; therefore, each entry can be represented using at most $\mathcal{O}(\log(nW))$ bits. Thus each individual recursive call uses at most $\mathcal{O}(n^3 \log(nW))$ bits. Combining this with Observation 3.1 proves the lemma. □

Next, we analyze the running time of the algorithm. On an n-vertex graph, let $\tau_i(k)$ be the total number of arithmetic operations of the procedure $F_i(T')$ for all $i \le 3$, where $k = |T'|$. It follows directly from the structure of the procedures F_i for $i \le 2$, that there exists a constant C such that the following recurrence holds for τ_i, $i \le 2$:

$$\begin{aligned} \tau_i(k) &\le \sum_{\frac{k}{3}\le j\le \frac{2k}{3}} \binom{k}{j} (\tau_{i+1}(j) + \tau_{i+1}(k-j) + Cn^3) \\ &\le 2 \sum_{\frac{k}{3}\le j\le \frac{2k}{3}} \binom{k}{j} (\tau_{i+1}(j) + Cn^3) \le 2k \max_{\frac{k}{3}\le j\le \frac{2k}{3}} \binom{k}{j} (\tau_{i+1}(j) + Cn^3). \end{aligned} \tag{5}$$

In the above expression, the term Cn^3 is the time required to execute the inner **for** loop of F_i. Let $\binom{k}{i_1,i_2,i_3}$ be the number of partitions of k distinct elements into sets of sizes i_1, i_2, and i_3 such that $i_1+i_2+i_3=k$. It follows directly from the structure of the procedure F_3 that there exists a constant C such that the following recurrence holds for τ_3:

$$\begin{aligned}
\tau_3(k) &= \sum_{i_1+i_2+i_3=k} \binom{k}{i_1,i_2,i_3}(\tau_2(i_1)+\tau_2(i_2)+\tau_2(i_3)+Cn^4)\\
&\le \sum_{\substack{i_1\ge i_2,i_3\\ i_1+i_2+i_3=k}} \binom{k}{i_1,i_2,i_3} 3\cdot(\tau_2(i_1)+Cn^4)\\
&\le 3\sum_{i_1\ge \frac{k}{3}} \binom{k}{i_1} 2^{k-i_1}\cdot(\tau_2(i_1)+Cn^4)\\
&\le 3k\max_{i_1\ge\frac{k}{3}} \binom{k}{i_1} 2^{k-i_1}\cdot(\tau_2(i_1)+Cn^4). \qquad (6)
\end{aligned}$$

In the above expression, the term Cn^4 is the time required to execute the inner **for** loop of F_3. Now we will bound $\tau_3(k)$ from above using (5) and (6). The following facts are required for the proof.

FACT 1 (see [13, Lemma 3.13]). *By Stirling's approximation, we have that* $\binom{k}{\alpha k} \le \left(\alpha^{-\alpha}(1-\alpha)^{(\alpha-1)}\right)^k$.

FACT 2. *For every fixed* $x \ge 4$, *the function* $f(y) = \frac{x^y}{y^y(1-y)^{1-y}}$ *is increasing on the interval* $(0, 2/3]$.

LEMMA 3.5. *There exists a constant* C *such that* $\tau_3(k) \le C\cdot 11.7899^k n^4$.

Proof. We prove by induction on k that $\tau_2(k) \le \hat{C}k^{(c\log k)}9.78977^k n^4$ and $\tau_3(k) \le \hat{C}k^{(c\log k)}11.7898^k n^4$, where $\hat{C}$ and c are constants. We will select the constant $\hat{C}$ to be larger than the constants of (5) and (6), and sufficiently large so that the base case of the induction holds. We prove the inductive step. By the induction hypothesis and (5), we have that

$$\begin{aligned}
\tau_2(k) &\le 2k\max_{\frac{1}{3}\le\alpha\le\frac{2}{3}} \binom{k}{\alpha k}\left(\hat{C}(\alpha k)^{(c\log\alpha k)}11.7898^{\alpha k}n^4+\hat{C}n^3\right)\\
&\le 2k\left(\frac{11.7898^{2/3}}{(2/3)^{2/3}(1/3)^{1/3}}\right)^k\cdot\left(\hat{C}\left(\frac{2k}{3}\right)^{(c\log 2k/3)}n^4+\hat{C}n^3\right) \quad \text{(Facts 1 and 2)}\\
&\le 2k\cdot(9.78977)^k\cdot\left(\hat{C}\left(\frac{2k}{3}\right)^{(c\log 2k/3)}n^4+\hat{C}n^3\right)\\
&\le 9.78977^k\cdot\hat{C}k^{(c\log k)}n^4.
\end{aligned}$$

The last inequality holds if c is a sufficiently large constant (independent of k). By the induction hypothesis and (6), we have that

$$\begin{aligned}\tau_3(k) &\le 3k \max_{1\ge\alpha\ge\frac{1}{3}} \binom{k}{\alpha k} 2^{(1-\alpha)k} \cdot \left(9.78977^{\alpha k} \cdot \hat{C}(\alpha k)^{(c\log \alpha k)} n^4 + \hat{C}n^4\right)\\ &\le 3k \max_{1\ge\alpha\ge\frac{1}{3}} \left(\alpha^{-\alpha}(1-\alpha)^{(\alpha-1)}2^{(1-\alpha)}9.78977^{\alpha}\right)^k \cdot \left(\hat{C}(\alpha k)^{(c\log\alpha k)} + \hat{C}n^4\right)\\ &\le 11.7898^k \cdot \hat{C}k^{(c\log k)}n^4.\end{aligned}$$

The last inequality holds for sufficiently large constants $\hat{C}$ and c. For a sufficiently large constant C it holds that

$$C \cdot 11.7899^k n^4 \ge 11.7898^k \cdot \hat{C}k^{(c\log k)}n^4,$$

completing the proof. ∎

To upper bound $\tau_0(k)$ from $\tau_3(k)$, we repeatedly use the following lemma.

LEMMA 3.6. *For every $i \le 2$ and constants C_{i+1} and $\beta_{i+1} \ge 4$ such that for every $k \ge 1$ we have $\tau_{i+1}(k) \le C_{i+1}\beta_{i+1}^k n^4$, there exists a constant C_i such that $\tau_i(k) \le C_i \cdot 1.8899^k \cdot \beta_{i+1}^{2k/3} \cdot n^4$.*

Proof. By (5) we have that

$$\begin{aligned}\tau_i(k) &\le 2k \max_{\frac{k}{3}\le j\le\frac{2k}{3}} \binom{k}{j}(\tau_{i+1}(j) + Cn^3)\\ &\le (2k + C) \max_{\frac{k}{3}\le j\le\frac{2k}{3}} \binom{k}{j}(C_{i+1}\beta_{i+1}^j n^4)\\ &\le C_{i+1} \cdot (2k + C) \cdot \left(\frac{3}{2^{2/3}}\right)^k \cdot \beta_{i+1}^{2k/3} \cdot n^4\\ &\le C_i \cdot 1.8899^k \cdot \beta_{i+1}^{2k/3} \cdot n^4.\end{aligned}$$

The last inequality holds for a sufficiently large C_i depending on C_{i+1} and β_{i+1} but not on k. ∎

LEMMA 3.7. *The procedure $F_0(T)$ uses $\mathcal{O}(7.97^k n^4 \log(nW))$ time.*

Proof. We show that $\tau_0(k) = \mathcal{O}(7.9631^k n^4)$. Since each arithmetic operation takes at most $\mathcal{O}(\log(nW))$ time, the lemma follows. Applying Lemma 3.6 on the upper bound for $\tau_3(k)$ from Lemma 3.5 proves that

$$\tau_2(k) = \mathcal{O}(1.8899^k \cdot 11.7899^{2k/3}n^4) = \mathcal{O}(9.790^k n^4).$$

Reapplying Lemma 3.6 on the above upper bound for $\tau_2(k)$ yields

$$\tau_1(k) = \mathcal{O}(1.8899^k \cdot 9.790^{2k/3}n^4) = \mathcal{O}(8.6489^k n^4).$$

Reapplying Lemma 3.6 on the above upper bound for $\tau_1(k)$ yields

$$\tau_0(k) = \mathcal{O}(1.8899^k \cdot 8.6489^{2k/3}n^4) = \mathcal{O}(7.9631^k n^4).$$

This completes the proof. ∎

We are now in position to prove our main theorem.

Proof of Theorem 1. The algorithm calls the procedure $F_0(T)$ and returns the value stored for $f_0(\emptyset, T)$. By Lemma 3.3, the procedure $F_0(T)$ correctly computes $f_0(\emptyset, T)$, and by Lemma 3.1, this is exactly equal to the weight of an optimal Steiner tree. By Lemma 3.4, the space used by the algorithm is at most $\mathcal{O}(n^3 \log(nW) \log k)$, and by Lemma 3.7, the time used is $\mathcal{O}(7.97^k n^4 \log(nW))$. ∎

3.1. Obtaining better parameter dependence. The algorithm from Theorem 1 is based on defining and computing the functions f_i, $0 \leq i \leq 3$. The functions f_i, $i \leq 2$, are defined using recurrence (2), while the function f_3 is defined using recurrence (3). For every constant $t \geq 4$, we could obtain an algorithm for Steiner Tree by defining functions f_i, $0 \leq i \leq t-1$, using (2) and f_t using (3). A proof identical to that of Lemma 3.1 shows that $f_i(S, T') = ST_G(S \cup T')$ for every $i \leq t$.

We can now compute $f_0(\emptyset, T)$ using an algorithm almost identical to the algorithm of 1, except that now we have $t+1$ procedures, namely a procedure F_i for each $i \leq t$. For each i and terminal set $T' \subseteq T$, a call to the procedure $F_i(T')$ computes an array containing $f_i(S, T')$ for every set S of size at most i.

For $i < t$, the procedure F_i is based on (2) and is essentially the same as Algorithm 1. Further, the procedure F_t is based on (3) and is essentially the same as Algorithm 2. The correctness of the algorithm and an $\mathcal{O}(n^t \log(nW))$ upper bound on the space usage follow from arguments identical to Lemmas 3.3 and 3.4, respectively.

For the running time bound, an argument identical to Lemma 3.5 shows that $\tau_t(k) = \mathcal{O}(11.7899^k n^{t+1})$. Furthermore, Lemma 3.6 now holds for $i \leq t-1$. In the proof of Lemma 3.7, the bound for $\tau_0(k)$ is obtained by starting with the $\mathcal{O}(11.7899^k n^4)$ bound for τ_3 and applying Lemma 3.6 three times. Here we can upper bound $\tau_0(k)$ by starting with the $\mathcal{O}(11.7899^k n^{t+1})$ bound for τ_t and applying Lemma 3.6 t times. This yields a $C_0 \cdot \beta_0^k$ upper bound for $\tau_0(k)$, where

$$\beta_0 = (11.7899^{(2/3)^t}) 1.8899^{\sum_{i=0}^{t-1} (2/3)^i}.$$

It is easy to see that when $t \geq 25$ the upper bound for β_0 is a number between 6.75 and 6.751. This proves Theorem 2.

4. Faster polynomial space algorithm.

In this section, for any $\epsilon > 0$, we design a $4^{(1+\epsilon)k} n^{\mathcal{O}(f(\epsilon))} \log W$ time, $n^{\mathcal{O}(f'(\epsilon))} \log W$ space algorithm for Steiner Tree, where f and f' are computable functions that depend only on ϵ. Towards that we need to explain Subset Steiner Forest problem and show that the algorithm in section 3 can be generalized to an algorithm for Subset Steiner Forest. In subsection 4.1 we explain Subset Steiner Forest. Then in subsection 4.2 we give a faster polynomial space algorithm for Steiner Tree.

4.1. Subset Steiner forest. In this subsection we generalize our parameterized single-exponential time polynomial space algorithm (the algorithm in section 3) to a general version of the Steiner Tree problem, named Subset Steiner Forest.

Definition 4.1. *Let G be a graph, $w : E(G) \to \{1, 2, \ldots, W\}$ be a nonnegative weight function, $\mathcal{S}$ be a partition of a subset of vertices, and $T \subseteq V(G)$ be a set of terminals. A subgraph G' of G is called a subset Steiner forest of G on the partition $\mathcal{S}$ and the terminal set T if the following conditions hold:*

- *$T \cup \left(\bigcup_{S \in \mathcal{S}} S\right) \subseteq V(G')$,*
- *for all $S \in \mathcal{S}$, there is a connected component C in G' such that $S \subseteq V(C)$, and*
- *for any $t \in T$, the connected component C in G', containing t, also contains some $S \in \mathcal{S}$.*

We use $sf_G(\mathcal{S},T)$ to denote the minimum weight of a subset Steiner forest of G on the partition $\mathcal{S}$ and the terminal set T.

In Subset Steiner Forest, the objective is to find a subset Steiner forest of minimum weight for an input graph G, a set of terminals T, and a partition $\mathcal{S}$ of a subset Y of vertices. For the application considered in this paper, we assume that $|Y|$ is a constant. Thus, the formal definition of a Subset Steiner Forest is as follows.

Subset Steiner Forest **Parameter:** $k = |T|$
Input: An undirected graph G, a nonnegative weight function $w : E(G) \to \{1, 2, \dots, W\}$, a partition $\mathcal{S}$ of a subset Y of vertices, and a set of terminals T, where $|Y|$ is a constant.
Question: A minimum weight subset Steiner forest of G on the partition $\mathcal{S}$ and the terminal set T.

The recurrence relations defined for Steiner Tree (see (4), (2) and (3)) can be generalized to get recurrence relations for Subset Steiner Forest. That is, we define four functions f_i for $i \in \{0, 1, 2, 3\}$. In what follows let $\mathcal{S} = \{S_1, \dots, S_r\}$ be a partition of a subset Y of the vertex set such that for all j, $|S_j| \le c$, where c and r are constants. Each function f_i takes as input a subset T' of T and a partition $\mathcal{S}$ of a vertex subset Y, of size r, such that for all $S \in \mathcal{S}$, $|S| \le c + i$. These functions f_i are recurrence relations for Subset Steiner Forest. That is $f_i(\mathcal{S}, T')$ is exactly $sf_G(\mathcal{S}, T')$. Now we define $f_i(\mathcal{S}, T')$ for all $i \in \{0, 1, 2, 3\}$. When $|T'| \le c + 3$, $f_i(\mathcal{S}, T') = sf_G(\mathcal{S}, T')$. For $|T'| > c + 3$, we define $f_i(S, T')$ using the following recurrences.

Separation. For $i \in \{0, 1, 2\}$, let us define

$$f_i(\mathcal{S}, T') = \min \sum_{\ell=1}^{2} f_{i+1}\big(\{S_1^{(\ell)} \cup \{v_1\}\}, \dots, \{S_r^{(\ell)} \cup \{v_r\}\}, T_\ell\big), \tag{7}$$

where the minimum is taken over partition $(T_1, T_2) \in \mathcal{B}_{\frac{1}{6}}(T')$, vertices $v_1, \dots, v_r \in V(G)$, and partition $S_j^{(1)} \uplus S_j^{(2)}$ of S_j for all $j \in [r]$.

Resplitting. For $i = 3$, let us define

$$f_i(\mathcal{S}, T') = \min \sum_{\ell=1}^{3} f_{i-1}\big(\{S_1^{(\ell)} \cup \{v_1\}\}, \dots, \{S_r^{(\ell)} \cup \{v_r\}\}, T_\ell\big), \tag{8}$$

where the minimum is taken over partition $(T_1, T_2, T_3) \in \mathbf{P}_3(T')$, vertices $v_1, \dots, v_r \in V(G)$, and for all $j \in [r]$, partition $S_j^{(1)} \uplus S_j^{(2)} \uplus S_j^{(3)}$ of S_j such that $|S_j^{(1)}|, |S_j^{(2)}|, |S_j^{(3)}| \le c - 2$.

The proof of correctness of the above recurrence relation is a generalization of the proof of correctness of the recurrence relations of Steiner Tree (see (4), (2), and (3)). As in section 3, a recursive algorithm using (7) and (8) can be designed, which leads to the following theorem.

Theorem 4. Subset Steiner Forest *can be solved in* $\mathcal{O}(7.97^k n^{r(c+4)} \log(nW))$ *time using* $n^{r(c+3)} \log(nW) \cdot \log k$ *space.*

4.2. Polynomial space $4^{(1+\epsilon)k} n^{\mathcal{O}(f(\epsilon))}$ time algorithm. In this subsection we design a faster polynomial space algorithm for Steiner Tree using Subset Steiner Forest. We design a new recurrence relation for subset Steiner forest using a notion of α-Steiner separator for subset Steiner forest. Given a subset Steiner forest SF on a set of terminals T and a partition $\mathcal{S}$ of a vertex subset, an α-Steiner separator S of SF is a subset of nodes which partitions $SF - S$ in two forests $\mathcal{R}_1$ and $\mathcal{R}_2$, each one containing at most $\alpha|T|$ terminals from T. The following lemma follows from Lemma 1 (sharp separation) of [12].

LEMMA 4.2 (c-separation). *For any constant $c \geq 0$, every subset Steiner forest SF on terminal set T and family $\mathcal{S}$ has a $\left(\frac{1}{2} + \frac{1}{2^{c/2}}\right)$-Steiner separator S of size at most c.*

We design a recurrence relation $g^{(c)}$ for subset Steiner forest for any constant $c \geq 2$. The function $g^{(c)}$ takes inputs $S \subseteq V(G)$, $P_S \in \mathbf{P}(S)$, and a set of terminals T. When $|T| \leq c$, the value of $g^{(c)}(S, P_S, T)$ is defined to be $sf_G(P_S, T)$. Otherwise, $g^{(c)}(S, P_S, T)$ is defined by the following recurrence relation. Let us fix $\alpha = \frac{1}{2^{(c/2)}}$,

$$g^{(c)}(S, P_S, T) = \min(g^{(c)}(S', P_1, T_1) + g^{(c)}(S', P_2, T_2)), \tag{9}$$

where the minimum is taken over partition $(T_1, T_2) \in \mathcal{B}_\alpha(T)$, a super set $S' \supseteq S$, and partitions $P_1, P_2 \in \mathbf{P}(S')$ such that $|S' \setminus S| \leq c$ and $P_S \preceq P_1 \sqcup P_2$.

We need to show that (9) is a recurrence relation for a SUBSET STEINER FOREST and we prove it in Lemma 4.4. The following lemma is useful for proving Lemma 4.4.

LEMMA 4.3. *Let G be a graph, $S', T \subseteq V(G)$, and $P_1, P_2 \in \mathbf{P}(S')$. Let F_1 and F_2 be subset Steiner forests for the pairs (P_1, T) and (P_2, T), respectively. Let $S \subseteq S'$ and $P_S \in \mathbf{P}(S)$ such that $P_S \preceq P_1 \sqcup P_2$. Then $F_1 + F_2$ is a subset Steiner forest for the pair (P_S, T).*

Proof. Let $F = F_1 + F_2$. Since F_1 is a subset Steiner forest for the pair (P_1, T), where $P_1 \in \mathbf{P}(S')$, we have that $S' \cup T \subseteq V(F_1)$. This implies that $S \cup T \subseteq V(F)$. Now we need to show that for any $Q \in P_S$, there is a component C in F such that $Q \subseteq V(C)$. Since F_1 and F_2 are SUBSET STEINER FORESTS for the pairs (P_1, T) and (P_2, T), respectively, the graph $F = F_1 + F_2$ is a subset Steiner forest for the pair $(P_1 \sqcup P_2, T)$. Since $P_S \preceq P_1 \sqcup P_2$ any $Q \in P_S$ is completely contained in a block Q' in $P_1 \sqcup P_2$. Since F is a subset Steiner forest for the pair $(P_1 \sqcup P_2, T)$, there is a component C in F such that $Q' \subseteq V(C)$. This implies that $Q \subseteq V(C)$. This completes the proof of the lemma. □

Now we show that the above recurrence (9) is indeed a recurrence relation for subset Steiner forest.

LEMMA 4.4. *For any $T \subseteq V(G)$, any partition P_S of vertex subset S of G, and any constant $c \geq 2$, it holds that $g^{(c)}(S, P_S, T) = sf_G(P_S, T)$.*

Proof. We prove the lemma using induction on $|T|$. For the base case, when $|T| \leq c$, the lemma holds by the definition of $g^{(c)}$. For the inductive step, let us assume that the lemma holds for all T' of size less than j and any $S' \subseteq V(G)$ and any partition $P_{S'} \in \mathbf{P}(S')$. We now show that $g^{(c)}(S, P_S, T) = sf_G(P_S, T)$ for all $T \in \binom{V(G)}{j}$, $S \subseteq V(G)$, and $P_S \in \mathbf{P}(S)$. Fix a set $T \in \binom{V(G)}{j}$, $S \subseteq V(G)$, and $P_S \in \mathbf{P}(S)$. Let $\alpha = \frac{1}{2^{(c/2)}}$.

First, we show that $g^{(c)}(S, P_S, T) \geq sf_G(P_S, T)$. By (9), we know there exist $(T_1, T_2) \in \mathcal{B}_\alpha(T)$, $S' \supseteq S$, and $P_1, P_2 \in \mathbf{P}(S')$ such that $|S' \setminus S| \leq c$, $P_S \preceq P_1 \sqcup P_2$, and

$$g^{(c)}(S, P_S, T) = g^{(c)}(S', P_1, T_1) + g^{(c)}(S', P_2, T_2).$$

Since $(T_1, T_2) \in \mathcal{B}_\alpha(T)$ and $|T| \geq 2$, we have that $|T_1|, |T_2| < |T|$. Then by induction hypothesis $g^{(c)}(S', P_1, T_1) = sf_G(P_1, T_1)$ and $g^{(c)}(S', P_2, T_2) = sf_G(P_2, T_2)$. So we have that $g^{(c)}(S, P_S, T) = sf_G(P_1, T_1) + sf_G(P_2, T_2)$. Let SF_1 and SF_2 be optimum subset Steiner forests for the pairs (P_1, T_1) and (P_2, T_2), respectively. Hence, by Lemma 4.3, $G' = SF_1 + SF_2$ is a subset Steiner forest for the pair (P_S, T). Thus we have shown that G' is a subset Steiner forest for the pair (P_S, T) of weight $g^{(c)}(S, P_S, T)$. This implies that $g^{(c)}(S, P_S, T) \geq sf_G(P_S, T)$.

Conversely, let SF be an optimum subset Steiner forest for the pair (P_S, T). By Lemma 4.2 we know that there exists an $(\frac{1}{2}+\alpha)$-Steiner separator S'' of SF such that $|S''| \leq c$. Let $S' = S'' \cup S$. Since $S' \supseteq S''$, S' is also an $(\frac{1}{2}+\alpha)$-Steiner separator of SF. Let $\mathcal{R}_1$ and $\mathcal{R}_2$ be the forests created by S' such that $|V(\mathcal{R}_1) \cap T| \leq (\frac{1}{2}+\alpha)|T|$ and $|V(\mathcal{R}_2) \cap T| \leq (\frac{1}{2}+\alpha)|T|$. Let $T_1 = V(\mathcal{R}_1) \cap T$ and $T_2 = V(\mathcal{R}_2) \cap T$. If $T_1 \cup T_2 \neq T$, then arbitrarily add each vertex in $T \setminus (T_1 \cup T_2)$ to either T_1 or T_2 such that $|T_r| \leq (\frac{1}{2}+\alpha)|T|$ for any $r \in \{0, 1\}$. Note that $(T_1, T_2) \in \mathcal{B}_\alpha(T)$. Since S' is a separator for $\mathcal{R}_1$ and $\mathcal{R}_2$ in SF, there is no edge in $E(SF)$ which is incident to both $\mathcal{R}_1$ and $\mathcal{R}_2$. Let E_1 be the set of edges in $E(SF)$ which are incident to $\mathcal{R}_1$, and let $E_2 = E(SF) \setminus E_1$. Consider the graphs $F_1 = (V(\mathcal{R}_1) \cup S', E_1)$ and $F_2 = (V(\mathcal{R}_2) \cup S', E_2)$. The graphs F_1 and F_2 are subset Steiner forests for the pairs $(P_{F_1}[S'], T_1)$ and $(P_{F_2}[S'], T_2)$, respectively (recall that for a graph G, P_G is the partition of $V(G)$, where each block is the vertex set of a component in G). Thus we have that

$$w(SF) = w(F_1) + w(F_2) \geq sf_G(P_{F_1}[S'], T_1) + sf_G(P_{F_2}[S'], T_2). \tag{10}$$

Since $F_1 + F_2 = SF$, we have that $P_S \preceq P_{F_1}[S'] \sqcup P_{F_2}[S']$. Thus, we have that $|S' \setminus S| \leq c$, $(T_1, T_2) \in \mathcal{B}_\alpha(T)$, and $P_{F_1}[S'], P_{F_2}[S'] \in \mathbf{P}(S')$ such that $P_S \preceq P_{F_1}[S'] \sqcup P_{F_2}[S']$. Hence by induction hypothesis and our recurrence relation (9), we have that $sf_G(P_{F_1}[S'], T_1) + sf_G(P_{F_2}[S'], T_2) \geq g^{(c)}(S, P_S, T)$. Combining this with (10), we get $g^{(c)}(S, P_S, T) \leq w(SF) = sf_G(P_S, T)$. ∎

Now for any $\epsilon > 0$, we explain an $n^{\mathcal{O}(f'(\epsilon))} \log W$ space $4^{(1+\epsilon)k} n^{\mathcal{O}(f(\epsilon))} \log W$ time algorithm for Steiner Tree. We fix (later) two constants $c \geq 4$ and d based on ϵ. Let $\alpha = \frac{1}{2^{(c/2)}}$ and $\beta = \left(\frac{1}{2}+\alpha\right)^d$. The algorithm is a recursive algorithm based on (9). Whenever the cardinality of the terminal set in a recursive call is bounded by βk, the algorithm uses 4 as a black box. Otherwise, it branches according to (9). The initial call to the recurrence is on the set of terminals T, $S = \emptyset$, and $P_S = \emptyset$. Since each application of (9) reduces the cardinality of the terminal set by a factor of $(\frac{1}{2}+\alpha)$, the depth of the recurrence tree is bounded by d. Hence the total number of vertices in the partition when our algorithm invokes Theorem 4 is bounded by $d \cdot c$. This implies that the space usage of our algorithm is bounded by $(n^{dc(dc+1)} + d) \log W$.

Now we bound the running time of the algorithm. Let $T(k)$ be the running time of the algorithm for an n-vertex graph.

Lemma 4.5. *There exists a constant C such that for any $k' \leq k$,*

$$T(k') = C \cdot (dc)^{(2dc)d'} n^{cd'} n^{cd(cd+4)} 2^{d'} \log nW \cdot (7.97)^{\beta k} 4^{k'} 2^{\frac{2k'}{1-2\alpha}},$$

where $d' = \log_{\frac{1}{2}+\alpha} \frac{\beta k}{k'}$.

Proof. Let C be a constant such that the algorithm in Theorem 4 runs in time $C \cdot 7.97^k n^{r(c+4)} \log(nW)$. We prove the lemma by induction on k'. Assume that the lemma holds for all values of $k' < k$. Now we need to bound $T(k)$. According to (9),

$$T(k) = n^c c^{2c} 2^k \cdot 2 \cdot T\left(\left(\frac{1}{2}+\alpha\right)k\right). \tag{11}$$

Let $\gamma = \left(\frac{1}{2}+\alpha\right)$. Now by induction hypothesis, we simplify (11) as

$$\begin{aligned} T(k) &= n^c c^{2c} 2^k \cdot 2 \cdot T(\gamma k) \\ &= n^c c^{2c} 2^{k+1} \cdot C(dc)^{(2dc)d'} n^{cd'} n^{cd(cd+4)} 2^{d'} \log(nW) \cdot (7.97)^{\beta k} 4^{\gamma k} 2^{\frac{2\gamma k}{1-2\alpha}}, \end{aligned} \tag{12}$$

where $d' = \log_\gamma \frac{\beta k}{\gamma k} \le d-1$.

Substituting $d' = d-1$ in (12),

$$T(k) = C \cdot (dc)^{(2dc)d} n^{cd} n^{cd(cd+4)} 2^d \log(nW) \cdot 2^k (7.97)^{\beta k} 4^{\gamma k} 2^{\frac{2\gamma k}{1-2\alpha}}. \tag{13}$$

CLAIM 4.6. $2^k 4^{\gamma k} 2^{\frac{2\gamma k}{1-2\alpha}} \le 4^k 2^{\frac{2k}{1-2\alpha}}$.

Proof. We prove that

$$\begin{aligned} 2^k 4^{\gamma k} 2^{\frac{2\gamma k}{1-2\alpha}} &\le 2^k \cdot 2^{(1+2\alpha)k} \cdot 2^{\frac{(1+2\alpha)k}{1-2\alpha}} \\ &\le 4^k \cdot 2^{\left(2\alpha + \frac{1+2\alpha}{1-2\alpha}\right)k}. \end{aligned} \tag{14}$$

Here $\alpha = \frac{1}{2^{(c/2)}}$. For $c \ge 4$, $\left(2\alpha + \frac{1+2\alpha}{1-2\alpha}\right) \le \frac{2}{1-2\alpha}$. This completes the proof of the claim. ∎

By applying Claim 4.6 in (13), we get

$$T(k) = C \cdot (dc)^{(2dc)d} n^{cd} n^{cd(cd+4)} 2^d \log(nW) \cdot (7.97)^{\beta k} 4^k 2^{\frac{2k}{1-2\alpha}}.$$

This completes the proof. ∎

Thus, for any $\epsilon > 0$, by choosing constants c and d such that $(7.97)^{\beta k} 2^{\frac{2k}{1-2\alpha}} \le 4^{\epsilon k}$, we derive the following theorem.

THEOREM 3. *For any $\epsilon > 0$ there is an $n^{\mathcal{O}(f'(\epsilon))} \log W$ space $4^{(1+\epsilon)k} n^{\mathcal{O}(f(\epsilon))} \log W$ time algorithm for* STEINER TREE*, where f and f' are computable functions depending only on ϵ.*

5. Conclusion. In this paper we designed the first polynomial space single-exponential FPT algorithm for STEINER TREE. Our algorithm runs in $\mathcal{O}(7.96^k n^4 \log(nW))$ time. We also designed an $\mathcal{O}(4^{(1+\epsilon)k} n^{f(\epsilon)})$ algorithm with $\mathcal{O}(n^{f'(\epsilon)})$ space complexity for any $\epsilon > 0$. Getting a faster algorithm without paying much on the polynomial factor and the space usage is an interesting open problem.

REFERENCES

[1] M. BASAVARAJU, F. V. FOMIN, P. A. GOLOVACH, P. MISRA, M. S. RAMANUJAN, AND S. SAURABH, *Parameterized algorithms to preserve connectivity*, in Proceedings of the 41st International Colloquium of Automata, Languages and Programming (ICALP), Lecture Notes in Comput. Sci. 8572, Springer, Heidelberg, 2014, pp. 800–811.

[2] M. W. BERN AND P. E. PLASSMANN, *The Steiner problem with edge lengths* 1 *and* 2, Inform. Process. Lett., 32 (1989), pp. 171–176.

[3] A. BJÖRKLUND, T. HUSFELDT, P. KASKI, AND M. KOIVISTO, *Fourier meets Möbius: Fast subset convolution*, in Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC), ACM, New York, 2007, pp. 67–74.

[4] G. E. BLELLOCH, K. DHAMDHERE, E. HALPERIN, R. RAVI, R. SCHWARTZ, AND S. SRIDHAR, *Fixed parameter tractability of binary near-perfect phylogenetic tree reconstruction*, in Proceedings of the 33rd International Colloquium of Automata, Languages and Programming (ICALP), Lecture Notes in Comput. Sci. 4051, Springer, Berlin, 2006, pp. 667–678.

[5] H. L. BODLAENDER, *A partial k-arboretum of graphs with bounded treewidth*, Theoret. Comput. Sci., 209 (1998), pp. 1–45.

[6] J. Byrka, F. Grandoni, T. Rothvoss, and L. Sanità, *Steiner tree approximation via iterative randomized rounding*, J. ACM, 60 (2013), 6.
[7] M. Cygan, F. V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh, *Parameterized Algorithms*, Springer, Cham, 2015.
[8] R. G. Downey and M. R. Fellows, *Fundamentals of Parameterized Complexity*, Texts Comput. Sci., Springer, London, 2013.
[9] S. E. Dreyfus and R. A. Wagner, *The Steiner problem in graphs*, Networks, 1 (1971), pp. 195–207.
[10] J. Flum and M. Grohe, *Parameterized Complexity Theory*, Texts Theoret. Comput. Sci. EATCS Ser., Springer-Verlag, Berlin, 2006.
[11] F. V. Fomin, F. Grandoni, D. Kratsch, D. Lokshtanov, and S. Saurabh, *Computing optimal Steiner trees in polynomial space*, Algorithmica, 65 (2013), pp. 584–604.
[12] F. V. Fomin, F. Grandoni, D. Lokshtanov, and S. Saurabh, *Sharp separation and applications to exact and parameterized algorithms*, Algorithmica, 63 (2012), pp. 692–706.
[13] F. V. Fomin and D. Kratsch, *Exact Exponential Algorithms*, Texts Theoret. Comput. Sci. EATCS Ser., Springer, Heidelberg, 2010.
[14] B. Fuchs, W. Kern, D. Mölle, S. Richter, P. Rossmanith, and X. Wang, *Dynamic programming for minimum Steiner trees*, Theory Comput. Syst., 41 (2007), pp. 493–500.
[15] J. Guo, R. Niedermeier, and S. Wernicke, *Parameterized complexity of generalized vertex cover problems*, in Proceedings of the 9th International Workshop Algorithms and Data Structures (WADS), Lecture Notes in Comput. Sci. 3608, Springer, Berlin, 2005, pp. 36–48.
[16] D. Lokshtanov and J. Nederlof, *Saving space by algebraization*, in Proceedings of the 42nd Annual ACM Symposium on Theory of Computing (STOC), ACM, New York, 2010, pp. 321–330.
[17] D. Marx, M. Pilipczuk, and M. Pilipczuk, *On subexponential parameterized algorithms for Steiner tree and directed subset TSP on planar graphs*, in Proceedings of the 59th IEEE Annual Symposium on Foundations of Computer Science, 2018, pp. 474–484, https://dblp.org/rec/bibtex/conf/focs/MarxPP18.
[18] J. Nederlof, *Fast polynomial-space algorithms using inclusion-exclusion*, Algorithmica, 65 (2013), pp. 868–884.
[19] R. Niedermeier, *Invitation to Fixed-Parameter Algorithms*, Oxford Lecture Ser. Math. Appl. 31, Oxford University Press, Oxford, UK, 2006.
[20] M. Pilipczuk, M. Pilipczuk, P. Sankowski, and E. J. van Leeuwen, *Subexponential-time parameterized algorithm for Steiner tree on planar graphs*, in Proceedings of the 30th International Symposium on Theoretical Aspects of Computer Science (STACS), Leibniz International Proceedings in Informatics (LIPIcs) 20, Dagstuhl, Germany, 2013, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Wadern, Germany, pp. 353–364.
[21] M. Pilipczuk, M. Pilipczuk, P. Sankowski, and E. J. van Leeuwen, *Network sparsification for Steiner problems on planar and bounded-genus graphs*, in Proceedings of the 55th Annual Symposium on Foundations of Computer Science (FOCS), IEEE, Los Alamitos, CA, 2014, pp. 276–285.
[22] H. J. Prömel and A. Steger, *The Steiner Tree Problem. A Tour through Graphs, Algorithms, and Complexity*, Adv. Lectures Math., Friedr. Vieweg & Sohn, Braunschweig, 2002.