# Parameterized single-exponential time polynomial space algorithm for Steiner Tree

Fedor V. Fomin[1], Petteri Kaski[2], Daniel Lokshtanov[1], Fahad Panolan[1,3], and Saket Saurabh[1,3]

[1] University of Bergen, Norway. {fomin|daniello}@uib.no
[2] Aalto University, Finland. petteri.kaski@aalto.fi
[3] Institute of Mathematical Sciences, India. {fahad|saket}@imsc.res.in

**Abstract.** In the Steiner tree problem, we are given as input a connected $n$-vertex graph with edge weights in $\{1, 2, \dots, W\}$, and a subset of $k$ terminal vertices. Our task is to compute a minimum-weight tree that contains all the terminals. We give an algorithm for this problem with running time $\mathcal{O}(7.97^k \cdot n^4 \cdot \log W)$ 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.

## 1 Introduction

In the Steiner Tree problem, we are given as input a connected $n$-vertex graph, a non-negative weight function $w : E(G) \to \{1, 2, \dots, W\}$, 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 $k = |T|$.

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 [16] for an 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 costs are either 1 or 2 [2]. On the other hand the problem admits a constant factor approximation algorithm, 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]. The classic algorithm for Steiner Tree of Dreyfus and Wagner [8] 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 [13]. Research on the parameterized complexity of Steiner Tree is still on-going, with very recent significant advances for the planar version of the problem [14, 15].

Algorithms for Steiner Tree are frequently used as a subroutine in fixed-parameter tractable (FPT) algorithms for other problems; examples include vertex cover problems [11], 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 [8]. Fuchs et al. [10] 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 [13]. 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 a $\mathcal{O}(W)$ factor both in the time and the space complexity. Lokshtanov and Nederlof [12] show that the $\mathcal{O}(W)$ factor can be removed from the space bound, their algorithm 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 [13] and the algorithm of Lokstanov and Nederlof [12] have a $\mathcal{O}(W)$ factor in their running time. Thus the running time of these algorithms depends exponentially on 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. [9] with the Dreyfus–Wagner algorithm. Indeed, one runs the algorithm of Fomin et al. [9] if $n \leq 2^k$, and the Dreyfus–Wagner algorithm if $n > 2^k$. If $n \leq 2^k$, 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 [9, 12].

**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. [9]. This algorithm crucially exploits the possibility for *balanced separation* (cf. Lemma 1 below). 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: (a) The two trees share exactly one vertex $v$ and no edges. (b) Neither of the two trees $ST_1$ or $ST_2$ contain more than a $2/3$ fraction of the terminal set $T$. (c) 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 $T_2 \cup \{v\}$.

Dually, to find the optimal tree $ST$ for the terminal set $T$ it suffices to (a) guess the vertex $v$, (b) the partition of $T$ into $T_1$ and $T_2$, and (c) recursively find optimal trees for the terminal sets $T_1 \cup \{v\}$ and $T_2 \cup \{v\}$. 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 3 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 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 memoization 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* $\mathcal{O}(7.97^k n^4 \log nW)$ *time using* $\mathcal{O}(n^3 \log nW \log k)$ *space.*

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 $k$, 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* $\mathcal{O}(6.751^k n^{O(1)} \log W)$ *time.*

## 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. For a path $P = u_1 u_2 \cdots u_\ell$ in a graph $G$, we use $\overleftarrow{P}$ to denote the reverse path $u_\ell u_{\ell-1} \cdots u_1$. 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 non negative 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. 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 $U$.

**Separation and resplitting.** A set of nodes $S$ is called an $\alpha$-*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 $\alpha n$ 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 $\alpha$-Steiner separator $S$ of $ST$ is a subset of nodes which partitions $ST - S$ in two forests $\mathcal{R}_1$ and $\mathcal{R}_2$, each one containing at most $\alpha k$ terminals from $T$.

**Lemma 1 (Separation).** [5,9] *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 $S$ set at no more than 3 vertices (see Section A in the appendix for a proof).

**Lemma 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* $S$.

## 3 Algorithm

In this section we design an algorithm for Steiner Tree which runs in time $\mathcal{O}(7.97^k n^4 \log nW)$ 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 the optimal Steiner tree to finding optimal Steiner trees in the same graph, but with a smaller terminal set. We will define four functions $f_i$ for $i \in \{0,1,2,3\}$. Each function $f_i$ takes as input a vertex set $S$ of size at most $i$ 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 $st_G(T' \cup S)$.

In the recurrences we will work with the following partitioning schemes for the current set of terminals $T'$. Let $\mathcal{P}(T')$ is the set of all possible partitions $(T_1,T_2,T_3)$ of $T'$ into three parts and let $\mathcal{B}(T')$ be the set of all possible partitions $(T_1,T_2)$ of $T'$ into two parts such that $|T_1|,|T_2| \leq 2k/3$.

For $T' \subseteq T$, $i \in \{0,1,2,3\}$, and $S \in \binom{V(G)}{\leq i}$, we define $f_i(S,T')$ as follows. When $|T'| \leq 2$, $f_i(S,T') = st_G(T' \cup S)$. For $|T'| \geq 3$, we define $f_i(S,T')$ using the following recurrences.

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

$$f_i(S,T') = \min_{(T_1,T_2)\in\mathcal{B}(T')} \min_{\substack{v\in V(G)\\ S_1 \uplus S_2 = S}} f_{i+1}\big(S_1 \cup \{v\}, T_1\big) + f_{i+1}\big(S_2 \cup \{v\}, T_2\big) \quad (2)$$

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

$$f_i(S,T') = \min_{(T_1,T_2,T_3)\in\mathcal{P}(T')} \min_{\substack{S_1 \uplus S_2 \uplus S_3 = S\\ |S_1|,|S_2|,|S_3| \leq i-2\\ v\in V(G)}} \sum_{r=1}^{3} f_{i-1}\big(S_r \cup \{v\}, T_r\big) \quad (3)$$

The recurrences (2) and (3) are recurrence relations for Steiner Tree (see Section B in the appendix for a proof):

**Lemma 3.** *For all $T' \subseteq T$, $0 \leq i \leq 3$, and $S \in \binom{V(G)}{\leq i}$ it holds that $f_i(S,T') = st_G(T' \cup S)$.*

Our algorithm uses (2) and (3) to compute $f_0(\emptyset, T)$, which is exactly the cost of an optimum Steiner tree. A naïve way of turning the recurrences into an algorithm would be to simply make one recursive procedure for each $f_i$, and apply (2) and (3) directly. However, this would result in a factor $n^{O(\log k)}$ in the running time, which we seek to avoid. As 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 (2) or (3), we should not just iterate

**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 $S \in \binom{V(G)}{\leq i}$
1  if $|T'| \leq 2$ then
2      for $S \in \binom{V(G)}{\leq 3}$ do
3          $A[S] \leftarrow st_G(T' \cup S)$ (compute using the Dreyfus–Wagner algorithm)
4      return $A$
5  for $S \in \binom{V(G)}{\leq i}$ do
6      $A[S] \leftarrow \infty$
7  for $T_1, T_2 \in \mathcal{B}(T')$ do
8      $A_1 \leftarrow F_{i+1}(T_1)$
9      $A_2 \leftarrow F_{i+1}(T_2)$
10     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
11         if $A[S_1 \uplus S_2] > A_1[S_1 \cup \{v\}] + A_2[S_2 \cup \{v\}]$ then
12             $A[S_1 \uplus S_2] \leftarrow A_1[S_1 \cup \{v\}] + A_2[S_2 \cup \{v\}]$
13 return $A$.
```

over every choice of $S$ and then apply the recurrence to compute $f_i(S, T')$ because it is much faster to compute all the entries of the return array of $F_i$ simultaneosly, 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 all $S \in \binom{V(G)}{\leq i}$. For each $(T_1, T_2) \in \mathcal{B}(T')$ (line 7), $F_i$ 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 all $S \in \binom{V(G)}{\leq i+1}$ and $A_2[S'] = f_i(S', T_2)$ for all $S' \in \binom{V(G)}{\leq i+1}$. 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 10), 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 the inner **for** loop, $A[S]$ contains the value

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

**Algorithm 2:** Implementation of procedure $F_3$

```
   Input: $T' \subseteq T$
   Output: $st_G(T' \cup S)$ for all $S \in \binom{V(G)}{\leq 3}$
1  if $|T'| \leq 2$ then
2      for $S \in \binom{V(G)}{\leq 3}$ do
3          $A[S] \leftarrow st_G(T' \cup S)$ (compute using the Dreyfus–Wagner algorithm)
4      return $A$
5  for $S \in \binom{V(G)}{\leq 3}$ do
6      $A[S] \leftarrow \infty$
7  for $T_1, T_2, T_3 \in \mathcal{P}(T')$ do
8      $A_1 \leftarrow F_2(T_1)$
9      $A_2 \leftarrow F_2(T_2)$
10     $A_3 \leftarrow F_2(T_3)$
11     for $S_1, S_2, S_3 \in \binom{V(G)}{\leq 1}$ and $v \in V(G)$ do
12         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
13             $A[S_1 \cup S_2 \cup S_3] \leftarrow A_1[S_1 \cup \{v\}] + A_2[S_2 \cup \{v\}] + A_3[S_3 \cup \{v\}]$
14 return $A$.
```

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

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

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 $(T_1, T_2, T_3) \in \mathcal{P}(T')$ (line 7), $F_3$ 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 $F_3(T_3)$ 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 11), 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 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 $(T_1, T_2, T_3) \in \mathcal{P}(T')$, we have updated $A[S]$ with

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

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

*Proof.* Correctness of Lemma 4 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 (2) (recurrence (3) in the case of $F_3$). □

**Observation 1** *The recursion tree of the procedure $F_0(T)$ has depth $\mathcal{O}(\log k)$.*

*Proof.* For every $i \leq 2$ the procedure $F_i(T')$ only makes recursive calls to $F_{i+1}(T'')$ where $|T''| \leq 2|T'|/3$. The procedure $F_3(T')$ makes recursive calls to $F_2(T'')$ where $|T''| \leq |T'|$. 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 5.** *The procedure $F_0(T)$ uses $\mathcal{O}(n^3 \log nW \log k)$ space.*

*Proof.* 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 recurisve call uses at most $\mathcal{O}(n^3 \log nW)$ bits. Combining this with Observation 1 proves the lemma. □

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

$$\begin{aligned}\tau_i(k) &\le \sum_{\frac{k}{3} \le j \le \frac{2k}{3}} \binom{k}{j}(\tau_{i+1}(j) + \tau_{i+1}(k-j) + Cn^3) \\ &\le 2 \sum_{\frac{k}{2} \le j \le \frac{2k}{3}} \binom{k}{j}(\tau_{i+1}(j) + Cn^3) \le 2k \max_{\frac{k}{2} \le i \le \frac{2k}{3}} \binom{k}{j}(\tau_{i+1}(j) + Cn^3) \quad (4)\end{aligned}$$

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 $i_3$. It follows directly from the structure of the procedure $F_3$, that there exists a constant $C$ such that the following recurrence holds for $\tau_3$:

$$\begin{aligned}\tau_3(k) &= \sum_{i_1+i_2+i_3=k} \binom{k}{i_1, i_2, i_3}(\tau_2(i_1) + \tau_2(i_2) + \tau_2(i_3) + Cn^4) \\ &\le \sum_{i_1 \ge i_2, i_3} \binom{k}{i_1, i_2, i_3} 3 \cdot (\tau_2(i_1) + Cn^4) \le 3 \sum_{i_1 \ge \frac{k}{3}} \binom{k}{i_1} 2^{k-i_1} \cdot (\tau_2(i_1) + Cn^4) \\ &\le 3k \max_{i_1 \ge \frac{k}{3}} \binom{k}{i_1} 2^{k-i_1} \cdot (\tau_2(i_1) + Cn^4) \quad (5)\end{aligned}$$

Now we will bound $\tau_3(k)$ from above using (4) and (5). The following facts are required for the proof.

**Fact 1** *By Stirling's approximation,* $\binom{k}{\alpha k} \le \left(\alpha^{-\alpha}(1-\alpha)^{(\alpha-1)}\right)^k$ [17].

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

**Lemma 6.** *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 $\tau_3(k) \le \hat{C}k^{(c \log k)} 11.7898^k n^4$. We will pick $\hat{C}$ to be a constant larger than the constants of (4) and (5), and sufficiently large so that the base case of the induction holds. We prove the inductive step. By the induction hypothesis and (4), we have that

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

The last inequality holds if $c$ is a sufficiently large constant (independent of $k$). By the induction hypothesis and (5), we have that

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

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

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

completing the proof. □

**Lemma 7.** *For every $i \leq 2$ and constants $C_{i+1}$ and $\beta_{i+1} \geq 4$ such that for every $k \geq 1$ we have $\tau_{i+1}(k) \leq C_{i+1}\beta_{i+1}^k n^4$, there exists a constant $C_i$ such that $\tau_i(k) \leq C_i \cdot 1.8899^k \cdot \beta_{i+1}^{2k/3} \cdot n^4$.*

*Proof.* By (4) we have that

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

The last inequality holds for a sufficiently large $C_i$ depending on $C_{i+1}$ and $\beta_{i+1}$ but not on $k$. □

**Lemma 8.** *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 7 on the upper bound for $\tau_3(k)$ from Lemma 6 proves that

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

Re-applying Lemma 7 on the above upper bound for $\tau_2(k)$ yields

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

Re-applying Lemma 7 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 4 the procedure $F_0(T)$ correctly computes $f_0(\emptyset, T)$, and by Lemma 3 this is exactly equal to the cost of the optimal Steiner tree. By Lemma 5 the space used by the algorithm is at most $\mathcal{O}(n^3 \log nW \log k)$, and by Lemma 8 the time used is $\mathcal{O}(7.97^k n^4 \log nW)$. $\square$

**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 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 Theorem 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. Furhter, 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 follows from arguments identical to Lemma 4 and Lemma 5 respectively.

For the running time bound, an argument identical to Lemma 6 shows that $\tau_t(k) = \mathcal{O}(11.7899^k n^{t+1})$. Furthermore, Lemma 7 now holds for $i \leq t-1$. In the proof of Lemma 8 the bound for $\tau_0(k)$ is obtained by starting with the $\mathcal{O}(11.7899^k n^4)$ bound for $\tau_3$ and applying Lemma 7 three times. Here we can upper bound $\tau_0(k)$ by starting with the $\mathcal{O}(11.7899^k n^{t+1})$ bound for $\tau_t$ and applying Lemma 7 $t$ 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 as $t$ tends to infinity, the upper bound for $\beta_0$ tends to a number between 6.75 and 6.751. This proves Theorem 2.

## 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)*, volume 8572 of *Lecture Notes in Comput. Sci.*, pages 800–811. Springer, 2014.
2. M. W. Bern and P. E. Plassmann. The Steiner problem with edge lengths 1 and 2. *Inf. Process. Lett.*, 32(4):171–176, 1989.

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)*, pages 67–74, New York, 2007. ACM.
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)*, volume 4051 of *Lecture Notes in Comput. Sci.*, pages 667–678. Springer, 2006.
5. H. L. Bodlaender. A partial $k$-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1-2):1–45, 1998.
6. J. Byrka, F. Grandoni, T. Rothvoß, and L. Sanità. Steiner tree approximation via iterative randomized rounding. *J. ACM*, 60(1):6, 2013.
7. R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity.* Texts in Computer Science. Springer, 2013.
8. S. E. Dreyfus and R. A. Wagner. The Steiner problem in graphs. *Networks*, 1(3):195–207, 1971.
9. F. V. Fomin, F. Grandoni, D. Kratsch, D. Lokshtanov, and S. Saurabh. Computing optimal Steiner trees in polynomial space. *Algorithmica*, 65(3):584–604, 2013.
10. B. Fuchs, W. Kern, D. Mölle, S. Richter, P. Rossmanith, and X. Wang. Dynamic programming for minimum Steiner trees. *Theory of Computing Systems*, 41(3):493–500, 2007.
11. 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 )*, volume 3608 of *Lecture Notes in Comput. Sci.*, pages 36–48. Springer, 2005.
12. D. Lokshtanov and J. Nederlof. Saving space by algebraization. In *Proceedings of the 42nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 321–330. ACM, 2010.
13. J. Nederlof. Fast polynomial-space algorithms using inclusion-exclusion. *Algorithmica*, 65(4):868–884, 2013.
14. 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)*, volume 20 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 353–364, Dagstuhl, Germany, 2013. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
15. 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)*, pages 276–285. IEEE, 2014.
16. H. J. Prömel and A. Steger. *The Steiner Tree Problem.* Advanced Lectures in Mathematics. Friedr. Vieweg & Sohn, Braunschweig, 2002.
17. H. Robbins. A remark on Stirling's formula. *Amer. Math. Monthly*, 62:26–29, 1955.

## A Proof of Lemma 2

**Lemma 9 (Lemma 2 restated).** *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 from $S$.*

*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 $V(P_1) \cap V(P_2) \neq \{s_3\}$. 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 that one 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 $w$. $\square$

## B Proof of Lemma 3

**Lemma 10 (Lemma 3 restated).** *For all $T' \subseteq T$, $0 \leq i \leq 3$, 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 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 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:** $0 \leq i \leq 2$**.** By (2), we know that there is a vertex $v \in V(G)$, $S_1 \uplus S_2 = S$ and a partition $(T_1, T_2) \in \mathcal{B}(T')$ 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 $(T_1, T_2) \in \mathcal{B}(T')$ 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 $ST_2$ 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 1, we know that there is a 2/3-Steiner separator $\{v\}$ of size one. 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

$1 \le r \le 2$. If $v \in T'$ and $|T_1| \le |T_2|$, 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 $|T'| \ge 3$, we have that $|T_1|, |T_2| \le 2|T'|/3 < |T'|$. Hence $(T_1, T_2) \in \mathcal{B}(T')$. Let $S_r = V(ST_r) \cap S$ and $T_r = V(F_r) \cap T'$, $1 \le r \le 2$. Thus $f_{i+1}(S_1 \cup \{v\}, T_1) + f_{i+1}(S_2 \cup \{v\}, T_2) \ge f_i(S, T')$ 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 $w(E(ST_1)) + w(E(ST_2)) \ge st_G(T_1 \cup S_1 \cup \{v\}) + st_G(T_2 \cup \{v\}) = f_{i+1}(S_1 \cup \{v\}, T_1) + f_{i+1}(S_2 \cup \{v\}, T_2) \ge f_i(S, T')$. 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 $f_i(S, T') \le st_G(T' \cup S)$.

**Case 2:** $i = 3$. By (3), there is $v \in V(G)$, $S_1, S_2, S_3 \in \binom{V(G)}{\le 1}$, $S_1 \uplus S_2 \uplus S_3 = S$, and a partition $(T_1, T_2, T_3) \in \mathcal{P}(T')$ 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 $1 \le r \le 3$. 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 $w(E(ST_1 + ST_2 + ST_3)) \le \sum_{r=1}^{3} st_G(T_r \cup S_r \cup \{v\})$. Thus $st_G(T' \cup S) \le w(E(ST_1 + ST_2 + ST_3)) \le \sum_{r=1}^{3} st_G(T_r \cup S_r \cup \{v\}) = f_3(S, T')$. Thus, $f_3(S, T') \ge st_G(T' \cup S)$.

Conversely, let $ST$ be an optimum Steiner tree for the set of terminals $T' \cup S$. By Lemma 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 be $\mathcal{C}_1, \mathcal{C}_2$ and $\mathcal{C}_3$ be a partition of connected components of $ST - v$ such that for each $|V(\mathcal{C}_r) \cap S| \le 1$ for all $1 \le r \le 3$. For each $r$, let $T_r = T' \cap V(\mathcal{C}_r)$. 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 $(T_1, T_2, T_3) \in \mathcal{P}(T')$. For each $r$, let $S_r = (S \setminus \{v\}) \cap V(ST_r)$. Since each $\mathcal{C}_r$ contains at most one vertex from $S$, $|S_r| \le 1$. This implies $\sum_{r=1}^{3} f_2(S_r \cup \{v\}, T_r) \ge f_3(S, T')$. 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 $1 \le r_1 \ne r_2 \le 3$ it holds that $V(\mathcal{C}_{r_1}) \cap V(\mathcal{C}_{r_2}) = \{v\}$, we have $st_G(T' \cup S) = w(E(ST)) = \sum_{r=1}^{3} w(E(ST_r)) \ge \sum_{r=1}^{3} f_2(S_r \cup \{v\}, T_r) \ge f_3(S, T')$. Thus $f_3(S, T') \le st_G(T' \cup S)$. $\square$