Faster Steiner Tree Computation in Polynomial-Space

Fedor V. Fomin[1⋆], Fabrizio Grandoni[2⋆⋆], and Dieter Kratsch[3]

[1] Department of Informatics, University of Bergen, N-5020 Bergen, Norway, fomin@ii.uib.no
[2] Dipartimento di Informatica, Sistemi e Produzione, Università di Roma "Tor Vergata", via del Politecnico 1, 00133, Roma, Italy, grandoni@disp.uniroma2.it
[3] LITA, Université Paul Verlaine-Metz, 57045 Metz Cedex 01, France, kratsch@univ-metz.fr

Abstract. Given an n-node graph and a subset of k terminal nodes, the NP-hard Steiner tree problem is to compute a minimum-size tree which spans the terminals. All the known algorithms for this problem which improve on trivial $O(1.62^n)$-time enumeration are based on dynamic programming, and require exponential space.
Motivated by the fact that exponential-space algorithms are typically impractical, in this paper we address the problem of designing faster polynomial-space algorithms. Our first contribution is a simple polynomial-space $O(6^k n^{O(\log k)})$-time algorithm, based on a variant of the classical tree-separator theorem. This improves on trivial $O(n^{k+O(1)})$ enumeration for, roughly, $k \leq n/4$.
Combining the algorithm above (for small k), with an improved branching strategy (for large k), we obtain an $O(1.60^n)$-time polynomial-space algorithm. The refined branching is based on a charging mechanism which shows that, for large values of k, convenient local configurations of terminals and non-terminals must exist. The analysis of the algorithm relies on the Measure & Conquer approach: the non-standard measure used here is a linear combination of the number of nodes and number of non-terminals.
As a byproduct of our work, we also improve the (exponential-space) time complexity of the problem from $O(1.42^n)$ to $O(1.36^n)$.

1 Introduction

The *Steiner tree problem* is one of the best-known optimization problems: Given a connected graph $G = (V, E)$ on $n = |V|$ nodes, edge costs $c : E \to \mathbb{R}^+$ and a set $T \subseteq V$ of $k = |T|$ *terminals*, the objective is to find a subtree S of G spanning T such that the cost of S (i.e. the total cost of its edges) is minimum. Steiner trees are important in various applications such as VLSI routings [22], phylogenetic tree reconstruction [21] and network routing [24]. We refer to the

⋆ Supported by the Norwegian Research Council
⋆⋆ Partially supported by MIUR under project MAINSTREAM.

book of Prömel and Steger [27] for an overview of the results and applications of the Steiner tree problem.

The Steiner tree problem is known to be NP-hard [18]. Furthermore, it is APX-complete, even when the graph is complete and all edge costs are either 1 or 2 [3]. Finding the best approximation algorithm for the Steiner tree problem has been a challenge and many papers have been written on this subject. The currently best polynomial-time approximation algorithm for the Steiner tree problem, due to Robins and Zelikovsky, has approximation ratio $1 + (\ln 3)/2 < 1.55$ [28]. Among other results, Robins and Zelikovsky establish an approximation ratio of 1.28 for complete graphs with edge costs 1 or 2. The Steiner tree problem remains NP-hard for Euclidean and rectilinear metrics [17]. On the positive side, Arora established polynomial-time approximation schemes for those two important variants of the Steiner tree problem [1].

The Steiner tree problem plays a crucial role also in parameterized algorithms [9,12,26]. The aim here is designing the fastest possible algorithm under the (realistic) assumption that $k \ll n$. For more than 30 years the fastest parameterized algorithm for the Steiner Tree problem was the classical $O^*(3^k)$ dynamic programming algorithm by Dreyfus and Wagner [10].[4] Dreyfus-Wagner's algorithm is still probably the most popular algorithm used for solving different variants of the Steiner tree problem in practice [8,16]. This algorithm and its variations are also used as a subroutine in many other algorithms. For example, recent applications of it can be found in fixed parameter tractable algorithms for certain vertex cover problems [19] and for near-perfect phylogenetic tree reconstruction [6]. Recent progress in parameterized complexity and exact algorithms led to new insights on the Steiner tree problem. Mölle, Richter, and Rossmanith [25] (see also [15]) improved the running time to $O^*((2+\epsilon)^k)$, for any constant $\epsilon > 0$. More recently, Björklund, Husfeldt, Kaski, and Koivisto [5] obtained an $O^*(2^k)$ time algorithm for the version of the problem where edges have bounded integer weights. All the mentioned algorithms are based on a dynamic programming approach: they store useful auxiliary information for every subset of the terminal set, and thus use exponential space $\Omega(2^k)$.

For arbitrary values of k, the fastest known $O^*(1.4143^n)$-time (exponential-space) algorithm for the Steiner tree problem is obtained by combining the algorithm by Mölle et al. [25] (for small k) with trivial enumeration (for large k).

Exponential-space versus polynomial-space. The situation with exact algorithms for the Steiner tree problem is quite typical for a number of other NP-hard problems: the best exponential time complexity is achieved by algorithms with exponential space complexity [29]. However, algorithms with very high space complexity are unlikely to be fast in practice, especially when external memory accesses are frequent. This kind of phenomena is not captured by the standard RAM model. Hence it makes sense to search for algorithms with

[4] Throughout this paper we use the O^* notation which suppresses polynomial factors: for any polynomial $p(n)$, $O(p(n)f(n))$ is $O^*(f(n))$.

low memory requirements, even if they are asymptotically slower than their exponential-space counterpart. Polynomial-space exact algorithms have been studied for various NP-hard problems, among them Hamiltonian Path [2,20,23] and Coloring [4].

For $k = \omega(\log n)$, the existing parameterized algorithms for the Steiner tree problem are not polynomial-space. Under that assumption, the fastest known polynomial-space algorithm is the (almost) trivial enumerative algorithm, based on the following observation. Since all the leaves of any optimal Steiner tree are terminals, the number of Steiner nodes T' of degree 3 or larger is at most k. Given T', the Steiner tree problem is equivalent to the minimum spanning tree problem on $G_M[T \cup T']$, where G_M is the metric closure of G. Such problem can be solved in polynomial time. Hence it is sufficient to list all the subsets $T' \subseteq N := V \setminus T$ of size at most k, and then apply the observation above. This takes time $O(\sum_{i=1}^{k} \binom{n-k}{i} n^{O(1)})$. For $k \ll n$ this running time is $O^*(n^k)$, while for arbitrary values of k it is $O^*(1.6181^n)$.

Our Results and Techniques. Motivated by the practical limitations of exponential-space algorithms and by the theoretical interest of the topic itself, in this paper we address the problem of designing faster polynomial-space exact algorithms for the Steiner tree problem. In particular, we present an exact algorithm for the cardinality version of the problem (where every edge is of weight one), of running time $O^*(1.5949^n)$. This result is achieved in three steps:

• We describe a new, easy-to-implement, Steiner tree algorithm, taking $O(5.96^k n^{O(\log k)})$ time and polynomial space. This means an improvement on known polynomial-space results for $\omega(\log n) = k \leq 0.269\, n$, which covers many real-world instances. Our result is based on a simple variant of the classical tree-separator theorem: shortly, there is a node in every Steiner tree which separates two *balanced* subsets of terminals. This can be exploited in a top-down recursive implementation of the classical algorithm by Dreyfus and Wagner, hence achieving running time $O((27/4)^k n^{O(\log k)})$ and polynomial-space. The running time can be refined to $O(5.96^k n^{O(\log k)})$ by exploiting the properties of the Steiner separators in combination with a more careful branching. This algorithm works also in the weighted case, and might be of independent, practical interest[5].

• We design an improved branching strategy, based on the following idea. When k is small, it is convenient to use the algorithm above. Otherwise, there must be clusters of terminals "close" to each other. This property can be used to guide the branching process. From the technical point of view, we use a simple charging mechanism to show that, for large k's, the graph must contain one of a small list of local configurations of terminals and non-terminals. On such configurations we are able to branch better than trivially.

• We analyze the algorithm above with the Measure & Conquer technique described in [13,14], and based on the quasiconvex analysis of multivariate recurrences by Eppstein [11]. The basic idea is designing a convenient (non-trivial)

[5] An experimental analysis of our algorithms is postponed to future work.

measure of the size of the problem. This measure is used to bound in a tighter way the progress made by the recursive algorithm considered at each branching step. The running time obtained with respect to the refined measure is eventually turned into the equivalent running time in terms of the standard measure considered (typically the number of nodes or edges for graph problems). As it will be clearer from the analysis, a convenient measure in our case is a linear combination of the number n of nodes and number $n_N = n - k$ of non-terminals in the graph.

Preliminaries. In the following $st_G(T)$ denotes the minimum number of edges of a Steiner tree of graph G over terminal set T. When the graph G is clear from the context, we will simply write $st(T)$. By *contracting* a subset of nodes V', we mean (i) removing V' from the graph, (ii) adding a new node v', and (iii) adding one edge between v' and each neighbor of V' not in V'. The following lemma is easy to verify.

Lemma 1. (Contraction Lemma) *Let (G, T) be an instance of the cardinality Steiner tree problem. Also let V' be a connected component of terminals, G' be the graph resulting from contracting V' in a unique node v', and $T' = T \cup \{v'\} \setminus V'$. Then*

$$st_G(T) = |V'| - 1 + st_{G'}(T').$$

The rest of this paper is organized as follows. In Section 2 we present our $O(5.96^k n^{O(\log k)})$ polynomial-space algorithm. The refined branching strategy based on the charging argument is described in Section 3, and analyzed in Section 4 with the Measure & Conquer technique.

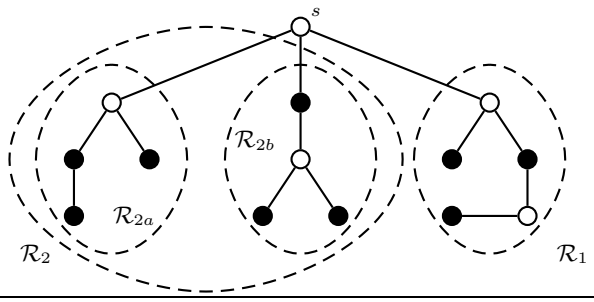
2 Steiner Tree via Steiner Separators

In this section we describe a simple polynomial-space algorithm for the Steiner tree problem of running time $O((27/4)^k n^{O(\log k)})$. We later show how to reduce the time complexity to $O(5.96^k n^{O(\log k)})$. We remark that, with minor modifications, this algorithm works also in the weighted case.

Our algorithm is inspired by the classical dynamic programming algorithm `D&W` by Dreyfus and Wagner [10], which takes $O^*(3^k)$ time and exponential space. Algorithm `D&W` is based on the following observation. Consider any Steiner tree S on the set of terminals T, $|T| \geq 3$. There must be an internal node $s \in S$, not necessarily a terminal, such that the subtrees of S rooted at s can be partitioned in two forests $\mathcal{R}_1$ and $\mathcal{R}_2$, each one containing at least one terminal. Let T_i be the terminals in $\mathcal{R}_i$, $i \in \{1, 2\}$. If we compute optimal Steiner trees on terminals $T_1 \cup \{s\}$ and $T_2 \cup \{s\}$, and we merge them, we obtain an optimal Steiner tree for the original problem. Of course we do not know s nor (T_1, T_2) a priori, but we can guess them by enumerating all the possible cases. Recall that $st_G(T) = st(T)$ is the minimum cost of a Steiner tree of G on terminals T. The following equation holds:

$$st(T) = \min_{s \in V} \min_{(T_1, T_2) \in \mathcal{P}(s,T)} \{st(T_1 \cup \{s\}) + st(T_2 \cup \{s\})\}, \tag{1}$$

Figure 1 Tight example for Lemma 2 (black nodes are terminals): a Steiner separator s, and the corresponding forests $\mathcal{R}_1$ and $\mathcal{R}_2$ with $|T_1| = k/3$ and $|T_2| = 2k/3$ terminals, respectively. Note that in $\mathcal{R}_2 \cup \{s\}$, node s separates two perfectly balanced forests $\mathcal{R}_{2a}$ and $\mathcal{R}_{2b}$.



where $\mathcal{P}(s,T)$ is the set of possible partitions (T_1,T_2) of $T \setminus \{s\}$ in two non-empty subsets. Algorithm `D&W` simply applies Equation (1) to any subset of T, in a bottom-up fashion, storing each partial solution computed for later computations. Storing the partial solutions takes $\Omega(2^k)$ space.

A simple-minded approach to obtain a polynomial-space variant of `D&W` is to apply Equation (1) recursively, in a top-down fashion, without storing any partial solution. When $|T| \leq 2$, the problem is solved trivially in polynomial time and space (*base case*). Unfortunately, this approach leads to a very high running time. The main reason is that, by applying Equation (1) as it is, one generates some subproblems with almost the same number of terminals as in the original problem.

This problem can be circumvented by exploiting a variant of the classical tree-separator theorem. It is well known that any n-node tree contains a node s (*separator*) whose removal divides the tree in two forests, each one containing at most $2n/3$ nodes. The same basic result holds if we put weights on the nodes [7]. In particular, the following lemma holds (see Figure 1 for a tight example).

Lemma 2. **[7]** *Consider any Steiner tree S on the set of terminals T, $|T| = k \geq 3$. Then there exists an internal node $s \in S$ (Steiner-separator), not necessarily a terminal, whose removal divides the tree in two forests, each one containing at most $2k/3$ terminals.*

As a consequence of Lemma 2, when applying Equation (1), we do not really need to consider all the partitions in $\mathcal{P}(s,T)$, but it is sufficient to consider only the subset $\mathcal{B}(s,T) \subseteq \mathcal{P}(s,T)$ of ("almost balanced") partitions (T_1,T_2) where $|T_1| \leq |T_2| \leq 2k/3$:

$$st(T) = \min_{s \in V} \min_{(T_1,T_2) \in \mathcal{B}(s,T)} \{st(T_1 \cup \{s\}) + st(T_2 \cup \{s\})\}. \quad (2)$$

Using Equation (2) instead of (1) makes no substantial difference with the dynamic programming approach by Dreyfus and Wagner: in fact, the most frequent

partitions (which determine the running time) contain a balanced number of terminals, and such partitions are contained both in $\mathcal{B}(s,T)$ and in $\mathcal{P}(s,T)$. The situation changes drastically in the top-down recursive implementation of the algorithm: here the running time is essentially determined by the most *unbalanced* partitions. Hence, replacing $\mathcal{P}(s,T)$ with $\mathcal{B}(s,T)$ has a tremendous impact on the performance of the algorithm.

The following Steiner tree algorithm summarizes the discussion above:

- **(base case)** If $T = \{v\}$, return v. If $T = \{v, w\}$, return the shortest path from v to w.
- **(recursive case)** For every $s \in V$ and for every partition (T_1, T_2) of $T \setminus \{s\}$, $|T_1| \leq |T_2| \leq 2k/3$, compute recursively optimal Steiner trees S_1 and S_2 over $T_1 \cup \{s\}$ and $T_2 \cup \{s\}$, respectively. Return the cheapest Steiner tree $S_1 \cup S_2$ obtained.

Theorem 1. *The Steiner tree algorithm above takes time* $O((27/4)^k n^{O(\log k)})$ *and polynomial space.*

Proof. The correctness of the algorithm follows from the discussion above, and its space complexity is trivially polynomial. Let $P(k)$ be the number of base instances generated by the algorithm to solve the problem. The time complexity of the algorithm is $O(P(k)n^{O(1)} \log k) = O(P(k)n^{O(1)})$, where we used the fact that each branching step takes polynomial time and the depth of the recursion is $O(\log k)$.

It remains to bound $P(k)$. We will show by induction that $P(k) \leq Cn^{c \ln k}\alpha^k$, for some constants $C > 0$, $c > 0$, and $\alpha \geq 4$. Clearly the condition is true for $k \leq 2$. Now assume it is satisfied for every $h \leq k-1$, and consider an instance with k terminals. For a given partition (T_1, T_2), the number of base instances generated is $P(|T_1|+1) + P(|T_2|+1)$. By construction, $k/2 \leq |T_2| \leq 2k/3$ and $|T_1| + |T_2| \leq k$. Hence, for sufficiently large constants C and c and for $\alpha = 8$, the following inequalities hold:

$$\begin{aligned} P(k) &\leq n \sum_{i=k/2}^{2k/3} \binom{k}{i} (P(i+1) + P(k-i+1)) \leq 2n \sum_{i=k/2}^{2k/3} \binom{k}{i} P(i+1) \\ &\leq 2n\, P(2k/3+1) \sum_{i=k/2}^{2k/3} \binom{k}{i} \leq 2n\, C n^{c \ln(2k/3+1)} \alpha^{2k/3+1}\, 2^k \\ &\leq C\, n^{c \ln k} (2\alpha^{2/3})^k \leq C\, n^{c \ln k} \alpha^k. \end{aligned}$$

Above we used the fact that $2\alpha^{2/3} = \alpha$ for $\alpha = 8$. In order to obtain a better value of α, we use the following observation.

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

From Stirling's formula and Fact 1, for $i \in [k/3, 2k/3]$,

$$\binom{k}{i} P(i+1) \leq \frac{C n^{c \ln(i+1)} \alpha^{i+1}}{((i/k)^{i/k}(1-i/k)^{1-i/k})^k} \leq \frac{(\alpha^{i/k})^k \alpha\, C\, n^{c \ln(2k/3+1)}}{((i/k)^{i/k}(1-i/k)^{1-i/k})^k}$$
$$\leq \alpha\, C\, n^{c \ln(2k/3+1)} \left(\frac{\alpha^{2/3}}{(2/3)^{2/3}(1/3)^{1/3}}\right)^k = \alpha\, C\, n^{c \ln(2k/3+1)} \left(\frac{3\alpha^{2/3}}{2^{2/3}}\right)^k.$$

It follows that

$$P(k) \leq 2n \sum_{i=k/2}^{2k/3} \binom{k}{i} P(i+1) \leq 2n\, k\, \alpha\, C\, n^{c \ln(2k/3+1)} \left(\frac{3\alpha^{2/3}}{2^{2/3}}\right)^k \leq C\, n^{c \ln k} \alpha^k,$$

for sufficiently large constants C and c and for $\alpha = 27/4$. The claim follows.

2.1 A Refined Algorithm

The algorithm of the previous section can be refined thanks to the following observation. Let S be an optimal Steiner tree. Consider the Steiner-separator $s \in S$ leading to the most balanced partition (T_1, T_2) of the terminals, $|T_1| \leq |T_2|$. In case of a tie, we choose s such that the forest $\mathcal{R}_2$ associated to T_2 contains the smallest possible number of nodes. In the worst possible case, $|T_1| = k/3$ and $|T_2| = 2k/3$. Note that in such case the forest $\mathcal{R}_2$ cannot be formed by a unique subtree of S. This is because otherwise the root s' of such a subtree would contradict the minimality of s. It follows that we can further partition $\mathcal{R}_2$ in two sub-forests $\mathcal{R}_{2a}$ and $\mathcal{R}_{2b}$. Let T_{2x} be the terminals of $\mathcal{R}_{2x}$, $x \in \{a, b\}$. Without loss of generality, we assume that $|T_{2a}| \leq |T_{2b}|$. Again by the minimality of s, we have $|T_{2b}| \leq |T_1| = k/3$. In fact, otherwise the partition $(T_{2b}, T_{2a} \cup T_1)$ would be more balanced than $(T_1, T_2) = (T_1, T_{2a} \cup T_{2b})$. It follows that $|T_{2a}| = |T_{2b}|$, that is in the subproblem induced by T_2 there is a perfectly balanced partition (see Figure 1 for an example).

This argument can be generalized in the following way. Let $\gamma \in (0, 1/6)$ be a given parameter. With the same notation as above, suppose $|T_2| \geq (2/3 - \gamma)k$. Then it must be $|T_{2a}| \leq |T_{2b}| \leq |T_1| \leq (1/3 + \gamma)k$. As a consequence, $\frac{|T_{2b}|}{|T_2|} \leq \frac{1/3+\gamma}{2/3-\gamma}$. For $\gamma < 1/15$ this gives a more balanced partition than the one guaranteed by Lemma 2 (which ensures $|T_{2b}|/|T_2| \leq 2/3$ only).

The idea is then to modify the algorithm of the previous section in the following way:

- For any partition (T_1, T_2) considered, if $|T_2| \geq (2/3 - \gamma)k$, then in the subproblem corresponding to T_2 consider only partitions (T_{2a}, T_{2b}) satisfying $|T_{2a}| \leq |T_{2b}| \leq \frac{1/3+\gamma}{2/3-\gamma}|T_2|$.

We can ideally partition the subproblems generated in two classes: (i) the subproblems with $|T_2| < (2/3-\gamma)k$ and (ii) the subproblems with $|T_2| \geq (2/3-\gamma)k$. For subproblems of type (i) the larger is γ, the better is the recurrence obtained

in the current step. For subproblems of type (ii), the smaller is γ, the better is the recurrence obtained in the following step. Optimizing $\gamma \in (0, 1/15)$ we obtain the following result, whose proof is omitted for lack of space.

Theorem 2. *For a proper choice of the parameter γ, the algorithm above solves the Steiner tree problem in time $O(5.96^k n^{O(\log k)})$ and polynomial space.*

In the following we will denote by `smallST` the algorithm of Theorem 2. We remark that `smallST` is *not* fixed-parameter-tractable because of the factor $n^{O(\log k)}$ in its running time. Finding a polynomial-space fixed-parameter-tractable algorithm for Steiner tree is left as a challenging open problem.

3 Branching on Small-Load Terminals

In this section we describe a simple, recursive algorithm `steiner` for the Steiner tree problem, taking $O(1.5949^n)$ time and polynomial space. Our algorithm computes the size $st_G(T)$ of an optimal Steiner tree, but it can be easily modified in order to produce one optimal Steiner tree.

The main idea behind our approach is as follows. If $k \leq cn$ for a suitable constant $c < 1$, it is convenient to use the $O(5.96^k n^{O(\log k)})$ algorithm from Section 2. Otherwise, there must be a terminal t which is at distance at most one from "many" other terminals. Thus, if by branching we add to T one or more non-terminals adjacent to t, we can contract a "large" connected component of terminals afterwards (using the Contraction Lemma 1). This phenomenon is not exploited in trivial enumeration, and it is at the base of our refined branching algorithm.

In order to formalize in a convenient way the mentioned phenomenon, we introduce the following definition of *load* of a terminal. Let each non-terminal node $s \in N := V \setminus T$ be initially assigned a load one. Node s evenly distributes its load among the terminals adjacent to it (if any). The final load $w(t)$ of each terminal t is the sum of the loads received by its non-terminal neighbors. As it will be clearer from the analysis, we can branch efficiently on terminals of small load.

We are now ready to describe algorithm `steiner`:

1. (**base**) If $|T| \in \{0, 1\}$, $st_G(T) = 0$:
$$\texttt{steiner}(G, T) = 0.$$

2. (**contraction**) If there is a connected component V' of at least 2 terminals, we apply Lemma 1. Let G' be the graph obtained from G by contracting V' in a node v', and let $T' = T \cup \{v'\} \setminus V'$. Then
$$\texttt{steiner}(G, T) = |V'| - 1 + \texttt{steiner}(G', T').$$

3. (**reduction**) If there is a terminal t adjacent to a unique (non-terminal) node s, we add s to the terminals since s must belong to any Steiner tree (being $k \geq 2$):
$$\texttt{steiner}(G, T) = \texttt{steiner}(G, T \cup \{s\}).$$

4. (**small k**) If $k \leq n/4$, we apply our algorithm `smallST`:

$$\texttt{steiner}(G,T) = \texttt{smallST}(G,T).$$

5. (**simple branch**) If there is a non-terminal s adjacent to at least 3 terminals, we simply branch by either removing s from the graph, or by adding it to the terminals:

$$\texttt{steiner}(G,T) = min\{\texttt{steiner}(G \setminus \{s\}, T), \texttt{steiner}(G, T \cup \{s\})\}.$$

6. (**multiple branch**) Let t be a terminal of minimum load according to the definition above, and let $s_1, \ldots, s_p$ be the (not-terminal) neighbors of t, sorted in decreasing number of adjacent terminals. We branch on the p subproblems obtained by removing $s_1, \ldots, s_{i-1}$, and adding s_i to the terminals, for $i \in \{1, \ldots, p\}$:

$$\texttt{steiner}(G,T) = \min_{i \in \{1,\ldots,p\}} \{\texttt{steiner}(G \setminus \{s_1, \ldots, s_{i-1}\}, T \cup \{s_i\})\}.$$

Observe that Algorithm `steiner` does not work in the weighted case. This is essentially due to the fact that the Contraction Lemma 1 does not extend to such case. Finding an improved algorithm for the weighted Steiner tree problem is an interesting open problem.

4 Analysis

We next analyze algorithm `steiner` with the Measure & Conquer technique described in [13,14]. Recall that $n_N = n - k$ is the number of non-terminals.

Theorem 3. *Algorithm* `steiner` *solves the Steiner tree problem in* $O(1.6011^n)$ *time and polynomial space.*

Proof. The correctness of the algorithm is not hard to check. For $k \leq n/4$ the running time of the algorithm is $O^*(5.96^k) = O^*(5.96^{n/4}) = O^*(1.5625^n)$, so assume that initially $k > n/4$. We let $h := n + n_N$ be the *size* of the problem, and denote by $T(h)$ the time required to solve a problem of size h. We will show by induction that $T(h) = O^*(1.3086^h)$. The claim follows since, being $n_N \leq 3n/4$ by assumption, $O^*(1.3086^h) = O^*(1.3086^{7n/4}) = O^*(1.6011^n)$.

Let $poly(n)$ be the maximum (polynomial) time spent at each step of the algorithm (excluding the recursive calls). For $h = 0$, $k = 0$ and hence $T(h) \leq poly(n) = O^*(1)$. Assume now that $T(h') = O^*(1.3086^{h'})$ for any $h' < h$, and consider the different steps of the algorithm.

Case 1 (base). The problem is solved directly: $T(h) \leq poly(h)$.

Case 2 (contraction). The algorithm generates a unique subproblem containing at most $n - 1$ nodes and n_N non-terminals:

$$T(h) \leq poly(h) + T(h-1) = poly(h) + O^*(1.3086^{h-1}) = O^*(1.3086^h).$$

$(m_1, \ldots, m_p)$	load	nodes removed
$(1,1)$	$4/2$	$1,2$
$(2,1)$	$3/2$	$2,2$
$(2,2)$	$2/2$	$2,3$
$(2,1,1)$	$5/2$	$2,2,3$
$(2,2,1)$	$4/2$	$2,3,3$
$(2,2,2)$	$3/2$	$2,3,4$
$(2,2,2,1)$	$5/2$	$2,3,4,4$
$(2,2,2,2)$	$4/2$	$2,3,4,5$
$(2,2,2,2,2)$	$5/2$	$2,3,4,5,6$

Table 1: Feasible values of $(m_1, \ldots, m_p)$ for multiple branch, with the corresponding load (strictly smaller than 3), and number of nodes removed in each subproblem. The number of non-terminals removed in the ith subproblem is i.

Case 3 (reduction). The algorithm adds s to the set of terminals (and hence removes one node from the non-terminals), and then removes at least one node by Case 2:

$$T(h) \leq 2poly(h) + T(h-2) = 2poly(h) + O^*(1.3086^{h-2}) = O^*(1.3086^h).$$

Case 4 (small k). The problem is solved by applying algorithm `smallST`, in time $O^*(5.96^k)$. Observe that, being $k \leq n/4$, $k = (n+n_N)\frac{n-n_N}{n+n_N} = h\frac{k}{2n-k} \leq h\frac{n/4}{7n/4} = \frac{h}{7}$. Hence the running time is $T(h) = O^*(5.96^k) = O^*(5.96^{h/7}) = O^*(1.2905^h)$.

Case 5 (single branch). Let $p \geq 3$ be the number of terminals adjacent to the selected non-terminal s. The algorithm generates two subproblems. In the first subproblem it removes s from the graph. In the second one it adds s to the terminals, and then it removes p nodes by Case 2. Hence

$$\begin{aligned} T(h) &\leq 2poly(h) + T(h-2) + T(h-1-p) \leq 2poly(h) + T(h-2) + T(h-4) \\ &= 2poly(h) + O^*(1.3086^{h-2}) + O^*(1.3086^{h-4}) = O^*(1.3086^h). \end{aligned}$$

Case 6 (multiple branch). Observe that, being $k > n/4$ by Case 4, the minimum load of a node is at most $\frac{n-k}{k} < \frac{3n/4}{n/4} = 3$. In particular, for the selected terminal t, $w(t) < 3$. Recall that $s_1, \ldots, s_p$ are the (non-terminal) neighbors of t, in decreasing order $m_1, \ldots, m_p$ of the number of adjacent terminals. Note that the load assigned by s_i to t is exactly $1/m_i$. By Case 5 it must be $m_i \in \{1,2\}$ for each i (each non-terminal has between 0 and 2 terminal neighbors). It follows by $w(t) < 3$ and by a simple case enumeration that the sequence $(m_1, \ldots, m_p)$ must be one of the sequences in Table 1.

In the ith subproblem, $i \in \{1, \ldots, p\}$, the algorithm removes nodes $s_1, \ldots, s_{i-1}$ from the graph, and adds node s_i to the terminals, which later determines the removal of m_i nodes by Case 2. Note that in the ith step i non-terminals are removed. Hence, by an easy case-by-case check,

$$\begin{aligned} T(h) &\leq (1+p)poly(h) + \sum_{i=1}^{p} T(h-(i-1)-m_i-i) \\ &= O^*(\sum_{i=1}^{p} 1.3086^{h-(i-1)-m_i-i}) = O^*(1.3086^h). \end{aligned}$$

4.1 A Refined Measure

The running-time analysis can be refined (without modifying the algorithm) by defining the size of the subproblems as $h := n + \alpha n_N$, for a proper constant $\alpha > 0$. Choosing $\alpha = 0.7297$, and by essentially the same analysis as in Theorem 3, we obtain the following result.

Theorem 4. *Algorithm* `steiner` *solves the Steiner tree problem in* $O(1.5949^n)$ *time and polynomial space.*

4.2 An Exponential-Space Algorithm

As a by-product of our approach, we are able to improve on the current best $O^*(1.4143^n)$ exponential-space algorithm as well. This is achieved by modifying algorithm `steiner` in the following way.

- In Step 4 replace `smallST` with the $O^*(2^k)$ algorithm of [5], and increase the corresponding threshold from $k \leq n/4$ to $k \leq 3n/7$.
- In Step 5 increase the threshold number of adjacent terminals from 3 to 5.

As a consequence of these changes, in Step 6 the minimum load of a terminal is strictly less than $\frac{n-3n/7}{3n/7} = \frac{4}{3}$ (instead of 3), and each non-terminal can have between 0 and 4 (instead of 2) adjacent terminals. Note that this implies a different list of feasible local configurations. The same kind of analysis as in Theorem 3 leads to the following result.

Theorem 5. *Algorithm* `steiner`*, modified as above, solves the Steiner tree problem in time* $O(1.3533^n)$ *and exponential space.*

References

1. S. Arora. Polynomial time approximation schemes for Euclidean TSP and other geometric problems. *J. ACM*, 45:753–782, 1998.
2. E. T. Bax. Inclusion and exclusion algorithm for the hamiltonian path problem. *Information Proc. Letters*, 47:203–207, 1993.
3. M. Bern and P. Plassmann. The Steiner tree problem with edge lengths 1 and 2. *Information Proc. Letters*, 32:171–176, 1989.
4. A. Björklund and T. Husfeldt. Inclusion-exclusion algorithms for counting set partitions. In *FOCS 2006*, pages 575–582, IEEE, 2006.
5. A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Fourier meets Möbius: Fast subset convolution. In STOC 2007, pages 67–74, 2007. ACM Press.
6. 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 ICALP 2006, volume 4051 of *LNCS*, pages 667–678. Springer, 2006.
7. H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theor. Comp. Sci.*, 209:1–45, 1998.
8. L. L. Deneen, G. M. Shute, and C. D. Thomborson. A probably fast, provably optimal algorithm for rectilinear Steiner trees. *Random Structures and Algorithms*, 5(4):535–557, 1994.

9. R. G. Downey and M. R. Fellows. *Parameterized complexity*. Springer-Verlag, New York, 1999.
10. S. E. Dreyfus and R. A. Wagner. The Steiner problem in graphs. *Networks*, 1:195–207, 1971/72.
11. D. Eppstein. Quasiconvex analysis of multivariate recurrence equations for backtracking algorithms. *ACM Transactions on Algorithms*, 2(4):492–509, 2006.
12. J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer-Verlag, Berlin, 2006.
13. F. Fomin, F. Grandoni, and D. Kratsch. Measure and conquer: domination - a case study. In *ICALP 2005*, vol. 3580 of LNCS, pages 191–203, 2005.
14. F. Fomin, F. Grandoni, and D. Kratsch. Measure and conquer: a simple $O(2^{0.288\,n})$ independent set algorithm. In SODA 2006, pages 18–25, 2006. ACM Press.
15. B. Fuchs, W. Kern, D. Mölle, S. Richter, P. Rossmanith, and X. Wang. Dynamic programming for minimum Steiner trees. *Theory of Computing Systems*, to appear, 2008.
16. J. L. Ganley. Computing optimal rectilinear Steiner trees: a survey and experimental evaluation. *Discrete Applied Mathematics*, 90(1-3):161–171, 1999.
17. M. R. Garey and D. S. Johnson. The rectilinear Steiner tree problem is NP-complete. *SIAM J. on Applied Mathematics*, 32:826–834, 1977.
18. M. R. Garey and D. S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness*. Freemann, 1979.
19. J. Guo, R. Niedermeier, and S. Wernicke. Parameterized complexity of generalized vertex cover problems. In WADS 2005, volume 3608 of *LNCS*, pages 36–48. Springer, 2005.
20. Y. Gurevich and S. Shelah. Expected computation time for Hamiltonian path problem. *SIAM J. Computing*, 16(3):486–502, 1987.
21. F. K. Hwang, D. S. Richards, and P. Winter. *The Steiner Tree Problem*. North-Holland, Amsterdam, 1992.
22. A. Kahng and G. Robins. *On Optimal Interconnections for VLSI*. Kluwer, Dordrecht, 1995.
23. R. M. Karp. Dynamic programming meets the principle of inclusion and exclusion. *Operation Research Letters*, 1:49–51, 1982.
24. B. Korte, H. J. Prömel, and A. Steger. Steiner trees in VLSI-layout. In *Paths, Flows, and VLSI-Layout*, pages 185–214, 1990.
25. D. Mölle, S. Richter, and P. Rossmanith. A faster algorithm for the Steiner tree problem. In *STACS 2006*, pages 561–570, 2006.
26. R. Niedermeier. *Invitation to fixed-parameter algorithms*, volume 31 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, Oxford, 2006.
27. H. J. Prömel and A. Steger. *The Steiner tree problem*. Advanced Lectures in Mathematics. Friedr. Vieweg & Sohn, Braunschweig, 2002.
28. G. Robins and A. Zelikovsky. Improved Steiner tree approximation in graphs. In *SODA 2000*, pages 770–779, 2000. ACM press.
29. G. Woeginger. Space and time complexity of exact algorithms: Some open problems. In *IWPEC 2004*, volume 3162 of *LNCS*, pages 281–290. Springer-Verlag, Berlin, 2004.