This paper appeared in: Mathematical Programming Computation 9(2), June 2017, 135-202

Dijkstra meets Steiner: a fast exact goal-oriented Steiner tree algorithm

Stefan Hougardy Jannik Silvanus Jens Vygen

June 2, 2014

Abstract

We present a new exact algorithm for the Steiner tree problem in graphs which is based on dynamic programming. Known empirically fast algorithms are primarily based on reductions, heuristics and branching. Our algorithm combines the best known worst-case run time with a fast, often superior, practical performance.

1 Introduction

We consider the well-known Steiner tree problem in graphs: Given an undirected graph G, costs $c : E(G) \to \mathbb{R}_{\geq 0}$ and a terminal set $D \subseteq V(G)$, find a tree Y in G such that $D \subseteq V(Y)$ and $c(E(Y))$ is minimum. The decision version of the Steiner tree problem is one of the classical NP-complete problems [18], it is even NP-complete in the special case that G is bipartite with $c \equiv 1$. Furthermore, it is NP-hard to approximate the Steiner tree problem within a factor of $\frac{96}{95}$ [5]. While for a long time the best known approximation algorithms were combinatorial, including the 1.55-approximation by Robins and Zelikovsky [25], the currently best known approximation algorithm by Byrka et al. [4] uses polyhedral methods to achieve a 1.39-approximation. The Steiner tree problem has many practical applications, in particular in VLSI design [17], where electrical connections are realized by Steiner trees.

From now on, we will refer to $|V(G)|$ by n, $|E(G)|$ by m and $|D|$ by k. Dreyfus and Wagner [8] applied dynamic programming to the Steiner tree problem to obtain an exact algorithm with a run time of $\mathcal{O}(n(n \log n + m) + 2^k n^2 + 3^k n)$ if implemented using Fibonacci heaps [12]. In 1987, Erickson, Monma and Veinott [10] improved the run time to $\mathcal{O}(3^k n + 2^k(n \log n + m))$ using a very similar approach. In 2006, Fuchs et al. [13] proposed an algorithm with a run time of $\mathcal{O}((2+\delta)^k n^{\left(\ln(\frac{1}{\delta})/\delta\right)^\zeta})$ for every sufficiently small $\delta > 0$ and $\zeta > \frac{1}{2}$, improving the exponential dependence on k from 3^k to $(2+\delta)^k$. Vygen [29] developed an algorithm with a worst-case run time of $\mathcal{O}(nk2^{k+\log_2(k)\log_2(n)})$, which is the fastest known algorithm if $f(n) < k < g(n)$ for some $f(n) = \text{polylog}(n)$ and $g(n) = \frac{n}{2} - \text{polylog}(n)$. However, for $k < 4 \log n$, the run time obtained by

Erickson, Monma and Veinott [10] is still the best known. See [29] for a more detailed analysis of the run times mentioned above.

For graphs with treewidth t, one can solve the Steiner tree problem in time $\mathcal{O}(nt^{\mathcal{O}(1)})$ [3]. An implementation of this algorihm was evaluated in [11]. Polzin and Vahdati Daneshmand [23] proposed an algorithm with a worst-case run time of $\mathcal{O}(n2^{b \log b+3b+\log b})$ where b is a parameter closely related to the pathwidth of G. They use this algorithm as a subroutine in their successful reduction-based Steiner tree solver [21, 28].

Except for the last mentioned algorithm, these results have played a very limited role in practice. Instead, empirically successful algorithms rely on preprocessing and reduction techniques, heuristics and branching: First, reductions [2, 9, 22, 27] are applied to reduce the size of the graph and the number of terminals, guaranteeing that optimum solutions of the reduced instance correspond to optimum solutions of the original instance. These reductions are not limited to simple local edge elimination tests, but may also rely on linear programming formulations and optimum solutions of partial instances. Primal and dual [1, 31] heuristics yield good upper and lower bounds, in many cases even resulting in a provably optimum solution. If these methods do not already solve the instance, enumerative algorithms are used. To this end, various authors [1, 6, 19] perform branch and cut. However, the solver by Polzin and Vahdati Daneshmand [21, 28], which achieved the best results so far, uses a branch & bound approach, where high effort is put into single nodes to minimize the number of branching nodes.

We propose a dynamic programming based algorithm with a worst-case run time of $\mathcal{O}(3^k n+2^k(n \log n+m))$, matching the best known result for small k. Our new algorithm achieves competitive results on VLSI instances, even without the use of preprocessing. Instead, good practical performance is achieved by using future cost estimates, which is a well-known concept to speed up shortest path computations [14], and effectively pruning partial solutions.

The rest of this paper is organized as follows: In Section 2, we describe our algorithm and show how to use lower bounds to improve its performance in practice. Examples of such lower bounds are given in Section 3. Section 4 introduces a pruning technique to further improve practical performance. Section 5 contains a few implementation details and computational results.

2 The Algorithm

For a set $X \subseteq V(G)$, we denote by $\mathrm{smt}(X)$ the length of a shortest Steiner tree for the terminal set X. Our algorithm needs an arbitrary root terminal $t \in D$. We will call the remaining terminals $D' := D \setminus \{t\}$ source terminals. Now consider the function $l : V(G) \times 2^{D'} \to \mathbb{R}_{\geq 0}$ with $l(v, I) := \mathrm{smt}(\{v\} \cup I)$. Then, $l(t, D')$ gives the cost of an optimum Steiner tree. The algorithms by Dreyfus and Wagner [8] and Erickson et al. [10] as well as our algorithm use dynamic programming to compute l.

The former two compute l as follows: For each i from 1 to $|D'|$, they consider all $I \subseteq D'$ with $|I| = i$ one after another and then compute $l(v, I)$ for all $v \in V(G)$. This way, it is guaranteed that when computing $l(v, I)$, the values $l(w, I')$ for all $w \in V(G)$ and $I' \subset I$ are already known. However, this leads to an exponential best case run time of $\Omega(2^k n)$. In contrast, our new algorithm considers all terminal sets simultaneously using a labeling technique similar to Dijkstra's algorithm [7]. This way, we do not necessarily have to compute all values of l. We can further improve practical performance by pruning and using future cost estimates, a generalization of a speedup technique known from Dijkstra's algorithm.

Our new algorithm labels from the source terminals towards the root t. More precisely, the algorithm labels elements of $V(G) \times 2^{D'}$. Each label (v, I) represents a shortest Steiner tree found so far connecting v with $I \subseteq D'$. Each time a label (v, I) becomes active, all neighbors w of v are checked and updated if the Steiner tree represented by (v, I) plus the edge $\{v, w\}$ leads to a better solution for (w, I) than previously known. This operation is well-known from Dijkstra's algorithm. In addition, for all disjoint sets $I' \subseteq D' \setminus I$ it is checked whether the Steiner trees for (v, I) and (v, I') combined to a tree for $(v, I \cup I')$ lead to a better solution than previously known.

To allow a simpler presentation, we restrict ourselves to instances without edges of zero cost. Such edges can be contracted in a trivial preprocessing step, preserving optimum solutions.

Now, we introduce the notion of feasible lower bounds, which are used by the algorithm to estimate the future cost of a label (v, I). The future cost of a label is the cost to complete the corresponding partial Steiner tree, which connects v with I, to a Steiner tree connecting all terminals by a tree connecting v with $D \setminus I$.

Definition 1. *Let (G, c, D) be an instance of the Steiner tree problem and $t \in D$. A function $\mathcal{L} : V(G) \times 2^D \to \mathbb{R}_{\geq 0}$ is called a* feasible lower bound *if*

$$\mathcal{L}(t, \{t\}) = 0$$

and

$$\mathcal{L}(v, I) \leq \mathcal{L}(w, I') + \mathrm{smt}((I \setminus I') \cup \{v, w\})$$

for all $v, w \in V(G)$ and $\{t\} \subseteq I' \subseteq I \subseteq D$.

Note that the values $\mathcal{L}(v, I)$ for $t \notin I$ do not affect whether $\mathcal{L}$ is a feasible lower bound. Also note that by choosing $I' = \{t\}$ and $w = t$, we have $\mathcal{L}(v, I) \leq \mathrm{smt}(I \cup \{v\})$, so a feasible lower bound by definition indeed is a lower bound on the length of a shortest Steiner tree. Moreover, if $e = \{v, w\} \in E(G)$ is an edge, by choosing $I' = I$, we have

$$\mathcal{L}(v, I) \leq \mathcal{L}(w, I) + \mathrm{smt}(\{v, w\}) \leq \mathcal{L}(w, I) + c(e).$$

This shows that feasible lower bounds generalize future costs (also known as feasible potentials), which are used in practice to speed up Dijkstra's algorithm by considering reduced edge costs [14]. In fact, our algorithm applied to the case $|D| = 2$ is identical to Dijkstra's algorithm using future costs in the very same way.

To construct the Steiner tree corresponding to (t, D'), each label (v, I) is equipped with backtracking data $b(v, I) \subseteq V(G) \times 2^{D'}$. If $b(v, I)$ is not empty, it will always either be of the form $b(v, I) = \{(w, I)\}$ where w is a neighbor of v or of the form $b(v, I) = \{(v, I_1), (v, I_2)\}$ where I_1 and I_2 form a partition of I into non-empty sets. In the first case, i.e., $b(v, I) = \{(w, I)\}$, the Steiner tree represented by the label (v, I) contains exactly one edge incident to v, which is $\{v, w\}$. In the second case, i.e., $b(v, I) = \{(v, I_1), (v, I_2)\}$, the Steiner tree represented by (v, I) contains at least two edges incident to v and thus can be split into two Steiner trees for the terminal sets $I_1 \cup \{v\}$ and $I_2 \cup \{v\}$, where I_1 and I_2 form a partition of I into nonempty sets. To be precise, it may happen that the subgraph of G corresponding to some label (v, I) contains cycles. However, since we ruled out edges of zero cost, this can only be the case as long as the label is not permanent.

By $P \subseteq V(G) \times 2^{D'}$ we denote the set of permanently labeled elements. For a vertex $v \in V(G)$, we denote by $\delta(v)$ the set of edges incident to v. Note that $\mathcal{L} \equiv 0$ is always a feasible lower bound, which may serve as an example to help understanding the algorithm. Other examples of feasible lower bounds will be discussed in Section 3.

Theorem 2. *The Dijkstra-Steiner algorithm works correctly.*

Proof. We will prove that the following invariants always hold when line 5 is executed:

(a) For each nonempty $I \subseteq D'$ and $v \in V(G)$ with $l(v, I) < \infty$:

  (a1) $l(v,I) = \begin{cases} c(\{v,w\}) + l(w,I) & \text{if } b(v,I) = \{(w,I)\}, \\ \sum_{(v,I') \in b(v,I)} l(v,I') & \text{otherwise,} \end{cases}$

  (a2) $I \cup \{v\} = \{v\} \cup \dot{\bigcup}_{(w,I') \in b(v,I)} I'$,

  (a3) backtrack(v, I) returns a connected subgraph T of G containing $\{v\} \cup I$ with $c(T) \leq l(v, I)$. If T is a tree, we have $c(T) = l(v, I)$.

(b) For each nonempty $I \subseteq D'$ and $v \in V(G)$ with $(v, I) \in P$:

$$l(v, I) = \text{smt}(\{v\} \cup I).$$

(c) For each nonempty $I \subseteq D'$ and $v \in V(G)$ with $(v, I) \notin P$:

  (c1) $l(v, I) \geq \text{smt}(\{v\} \cup I)$,

  (c2) If $I = \{v\}$, then $l(v, I) = 0$, otherwise
  $l(v, I) \leq \min_{\{v,w\} \in \delta(v), (w,I) \in P} (l(w, I) + c(\{v, w\}))$ and
  $l(v, I) \leq \min_{I = I_1 \dot\cup I_2 \text{ and } (v,I_1),(v,I_2) \in P} (l(v, I_1) + l(v, I_2))$.

This paper appeared in: Mathematical Programming Computation 9(2), June 2017, 135-202

Dijkstra-Steiner algorithm

Input : A connected undirected graph G, costs $c : E(G) \to \mathbb{R}_{>0}$, a terminal set $D \subseteq V(G)$, a root terminal $t \in D$, and a feasible lower bound $\mathcal{L} : V(G) \times 2^D \to \mathbb{R}_{\geq 0}$.

Output: A shortest Steiner tree for D in G.

```
$l(s,\{s\}) := 0$ for all $s \in D' := D \setminus \{t\}$ and $l(v,I) := \infty$ for all other $(v,I) \in V(G) \times 2^{D'}$;
$b(v,I) := \emptyset$ for all $(v,I) \in V(G) \times 2^{D'}$;
$P := V(G) \times \{\emptyset\}$;
while $(t, D') \notin P$ do
    Choose $(v,I) \in (V(G) \times 2^{D'}) \setminus P$ minimizing $l(v,I) + \mathcal{L}(v, D \setminus I)$;
    for all $e = \{v,w\} \in \delta(v)$ do
        if $l(v,I) + c(e) < l(w,I)$ then
            $l(w,I) := l(v,I) + c(e)$;
            $b(w,I) := \{(v,I)\}$;
        end
    end
    for all $\emptyset \neq I' \subseteq D' \setminus I$ with $(v,I') \in P$ do
        if $l(v,I) + l(v,I') < l(v, I \cup I')$ then
            $l(v, I \cup I') := l(v,I) + l(v,I')$;
            $b(v, I \cup I') := \{(v,I),(v,I')\}$;
        end
    end
    $P := P \cup \{(v,I)\}$;
end
return backtrack($t, D'$);

Procedure backtrack($v, I$)
    $T := (\{v\}, \emptyset)$;
    $V(T) := V(T) \cup \bigcup_{(w,I') \in b(v,I)} V($backtrack($w, I'$)$)$;
    $E(T) := \bigcup_{(w,I') \in b(v,I)} E($backtrack($w, I'$)$)$;
    if $b(v,I) = \{(w,I)\}$ for a neighbor $w$ of $v$ then
        $E(T) := E(T) \cup \{\{v,w\}\}$;
    end
    return $T$;
```

(d) There is a label (v, I) that can be chosen, i.e., $(V(G) \times 2^{D'}) \setminus P$ is not empty.

Assuming (a) – (d), the correctness of the algorithm directly follows: Once we have $(t, D') \in P$, (b) implies $l(t, D') = \text{smt}(\{t\} \cup D') = \text{smt}(D)$. Furthermore, (a3) implies the algorithm returns a connected subgraph T of G containing D with $c(T) \leq l(t, D') = \text{smt}(D)$. Since there are no edges of zero cost, T indeed is a tree.
Clearly, after line 3 these invariants hold. We have to prove that lines 5 to 18 preserve (a), (b), (c) and (d).
To this end, let (v, I) be the label chosen in line 5 in some iteration. Clearly, lines 5 to 18 preserve (a) and (c2). Since (c) held before the current iteration, we have $l(v, I) \geq \text{smt}(\{v\} \cup I)$. This directly implies

$$\begin{aligned} l(v, I) + c(\{v, w\}) &\geq \text{smt}(\{v\} \cup I) + c(\{v, w\}) \\ &\geq \text{smt}(\{v, w\} \cup I) \\ &\geq \text{smt}(\{w\} \cup I). \end{aligned}$$

Also, if $\emptyset \neq I' \subseteq D' \setminus I$ is a set chosen in line 12 leading to the change of $l(v, I \cup I')$, we have

$$\begin{aligned} l(v, I \cup I') &= l(v, I) + l(v, I') \\ &\geq \text{smt}(\{v\} \cup I) + \text{smt}(\{v\} \cup I') \\ &\geq \text{smt}(\{v\} \cup I \cup I'), \end{aligned}$$

so (c1) indeed is preserved.
We now show that $l(v, I) \leq \text{smt}(\{v\} \cup I)$. We can assume $I \neq \{v\}$, since $l(v, \{v\}) = 0 = \text{smt}(\{v\})$. Let Y be a Steiner tree for $\{v\} \cup I$ in G and w.l.o.g. we assume all leaves of Y are contained in $\{v\} \cup I$. For a vertex $w \in V(Y)$, let Y_w be the subtree of Y containing all vertices x for which the unique $x-v-$path in Y contains w.
We will now find a vertex $w \in V(Y)$, a nonempty terminal set $I' \subseteq I \cap V(Y_w)$ and a subtree Y' of Y_w such that

(I) $(w, I') \notin P$,

(II) $l(w, I') \leq c(Y')$,

(III) Y' is a subtree of Y_w containing $I' \cup \{w\}$,

(IV) $Y - Y'$ is a tree containing $(I \setminus I') \cup \{v, w\}$.

Here, by $Y - Y'$, we refer to the graph $((V(Y) \setminus V(Y')) \cup \{w\}, E(Y) \setminus E(Y'))$.
Assuming we have a triple (w, I', Y') satisfying (I) – (IV), $l(v, I) \leq \text{smt}(\{v\} \cup I)$ can easily be proved: Since $\mathcal{L}$ is a feasible lower bound, we have

$$\begin{aligned} \mathcal{L}(w, D \setminus I') &\leq \mathcal{L}(v, D \setminus I) + \text{smt}((I \setminus I') \cup \{v, w\}) \\ &\leq \mathcal{L}(v, D \setminus I) + c(Y - Y') \\ &= \mathcal{L}(v, D \setminus I) + c(Y) - c(Y'). \end{aligned} \tag{1}$$

Adding (II) and (1) yields

$$l(w, I') + \mathcal{L}(w, D \setminus I') \leq c(Y) + \mathcal{L}(v, D \setminus I).$$

By the choice of (v, I) in line 5 we have

$$l(v, I) + \mathcal{L}(v, D \setminus I) \leq l(w, I') + \mathcal{L}(w, D \setminus I'),$$

so

$$l(v, I) \leq c(Y).$$

It remains to be shown that we can find such a triple (w, I', Y'). We call $w \in V(Y)$ *proper* if $(w, I \cap V(Y_w)) \in P$ before the execution of line 18. If we have a leaf $w \in V(Y) \setminus \{v\}$ which is not proper, we set $I' = \{w\}$ and $Y' = (\{w\}, \emptyset)$, clearly satisfying (I) – (IV). Otherwise, since v is not proper, there is a vertex w in Y which is not proper but all neighbors $w_1, \ldots, w_j$ of w in Y_w are proper. Let J be an inclusion-wise minimal subset of $\{1, \ldots, j\}$ such that $\big(w, \bigcup_{i \in J}(I \cap V(Y_{w_i}))\big) \notin P$. Since w is not proper and $V(G) \times \{\emptyset\} \subseteq P$, J exists and J is not empty. For $i \in \{1, \ldots, j\}$, let $Y_{w_i \to w}$ be the subgraph of Y_w with $V(Y_{w_i \to w}) = \{w\} \cup V(Y_{w_i})$ and $E(Y_{w_i \to w}) = \{\{w, w_i\}\} \cup E(Y_{w_i})$. We define

(i) $I' = \bigcup_{i \in J}(I \cap V(Y_{w_i}))$,

(ii) $V(Y') = \bigcup_{i \in J} V(Y_{w_i \to w})$ and

(iii) $E(Y') = \bigcup_{i \in J} E(Y_{w_i \to w})$.

See Figure 1 for an illustration of this setting.

The conditions (I), (III) and (IV) are satisfied by construction. If $J = \{i\}$ for some $1 \leq i \leq j$, by (c2), we have

$$\begin{aligned} l(w, I') &\leq l(w_i, I') + c(\{w_i, w\}) \\ &\leq \mathrm{smt}(\{w_i\} \cup I') + c(\{w_i, w\}) \\ &\leq c(Y_{w_i}) + c(\{w_i, w\}) \\ &= c(Y'). \end{aligned}$$

Otherwise choose an arbitrary $i \in J$ and see again by (c2) that

$$\begin{aligned} l(w, I') &\leq l(w, I' \setminus V(Y_{w_i})) + l(w, I' \cap V(Y_{w_i})) \\ &\leq \mathrm{smt}(\{w\} \cup (I' \setminus V(Y_{w_i}))) + \mathrm{smt}(\{w\} \cup (I' \cap V(Y_{w_i}))) \\ &\leq c(Y' - Y_{w_i \to w}) + c(Y_{w_i \to w}) \\ &= c(Y') - c(Y_{w_i \to w}) + c(Y_{w_i \to w}) \\ &= c(Y'). \end{aligned}$$

By the same argument applied to (t, D'), we also see that there always is a label $(v, I) \notin P$ with $l(v, I) < \infty$ which can be chosen in line 5 as long as $(t, D') \notin P$, so (d) is preserved as well. □

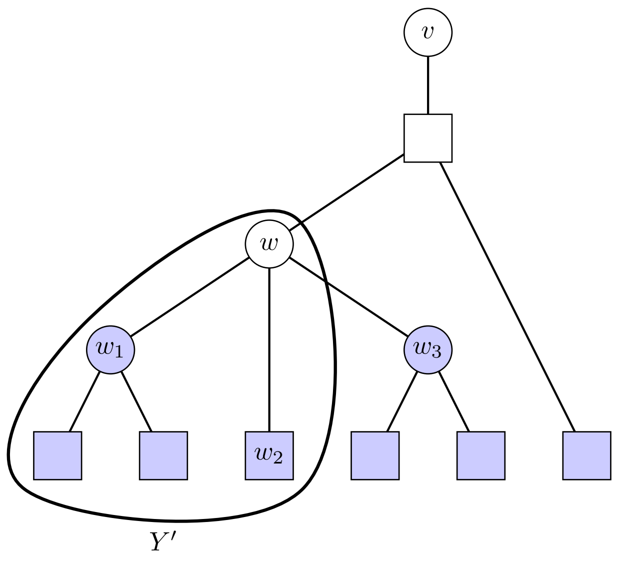

Figure 1: A possible configuration with $J = \{1, 2\}$. Vertices in I are drawn as squares, proper vertices are drawn in blue.

Note that in line 5, one can actually choose any label $(v, I) \notin P$ with $l(v, I) + \mathcal{L}(v, D \setminus I) \leq l(w, I') + \mathcal{L}(w, D \setminus I')$ for all $(w, I') \in (V(G) \times 2^I) \setminus P$. This is a generalization of the choice as specified in the algorithm. However, in our implementation, we always choose a label minimizing $l(v, I) + \mathcal{L}(v, D \setminus I)$.

Theorem 3. *The Dijkstra-Steiner algorithm can be implemented to run in $\mathcal{O}(3^k n + 2^k(n \log n + m) + 2^k n f_{\mathcal{L}})$ time, where $n = |V(G)|$, $m = |E(G)|$, $k = |D|$ and $f_{\mathcal{L}}$ is an upper bound on the time required to evaluate $\mathcal{L}$.*

Proof. Since $|P|$ increases in each iteration, we have at most $n2^{k-1}$ iterations. We use a Fibonacci heap [12] to store all labels $(v, I) \notin P$ with $l(v, I) < \infty$, which allows updates in constant amortized time. Since the heap contains at most $2^{k-1}n$ elements, each execution of line 5 takes $\mathcal{O}(\log(2^{k-1}n)) = \mathcal{O}(k + \log(n))$ amortized time. Line 7 is executed at most $2^k m$ times and each execution takes $O(1)$ amortized time. Furthermore, there are exactly 3^{k-1} pairs $I, I' \subseteq D'$ with $I \cap I' = \emptyset$, since every element in D' can either be contained in I, I' or $D' \setminus (I \cup I')$, independently of the others. Thus, line 12 is executed at most $3^{k-1}n$ times. By caching values of $\mathcal{L}$, we can achieve that we query $\mathcal{L}$ at most once for each label, resulting in an additional run time of $\mathcal{O}(2^k n f_{\mathcal{L}})$. The run time of the backtracking clearly is dominated by the previous tasks, since backtrack is called at most $\mathcal{O}(kn)$ times and requires effort linear in the size of its output. We get a total run time of

$$\mathcal{O}(2^k n(\log n + k) + 2^k m + 3^k n + 2^k n f_{\mathcal{L}}) = \mathcal{O}(3^k n + 2^k(n \log n + m) + 2^k n f_{\mathcal{L}}),$$

since $2^k k = \mathcal{O}(3^k)$. □

This paper appeared in: Mathematical Programming Computation 9(2), June 2017, 135-202

In Section 3, we will see that there are non-trivial feasible lower bounds $\mathcal{L}$ which can be used in the algorithm while still achieving a worst-case run time of $\mathcal{O}(3^k n + 2^k(n \log n + m))$, matching the result of [10].

3 Lower Bounds

Roughly speaking, the larger the feasible lower bound $\mathcal{L}$ is, the faster our algorithm will be. Before we describe examples of feasible lower bounds, we note:

Proposition 4. *Let $\mathcal{L}$ and $\mathcal{L}'$ be feasible lower bounds. Then,* $\max(\mathcal{L}, \mathcal{L}')$ *also is a feasible lower bound.*

Proof. Let $v, w \in V(G)$ and $I' \subseteq I \subseteq D$. Then

$$\begin{aligned}\mathcal{L}(v, I) &\leq \mathcal{L}(w, I') + \operatorname{smt}((I \setminus I') \cup \{v, w\}) \\ &\leq \max(\mathcal{L}(w, I'), \mathcal{L}'(w, I')) + \operatorname{smt}((I \setminus I') \cup \{v, w\})\end{aligned}$$

and

$$\begin{aligned}\mathcal{L}'(v, I) &\leq \mathcal{L}'(w, I') + \operatorname{smt}((I \setminus I') \cup \{v, w\}) \\ &\leq \max(\mathcal{L}(w, I'), \mathcal{L}'(w, I')) + \operatorname{smt}((I \setminus I') \cup \{v, w\}),\end{aligned}$$

so

$$\max(\mathcal{L}(v, I), \mathcal{L}'(v, I)) \leq \max(\mathcal{L}(w, I'), \mathcal{L}'(w, I')) + \operatorname{smt}((I \setminus I') \cup \{v, w\}).$$

□

Proposition 4 allows the combination of arbitrary feasible lower bounds. Now, we present three types of feasible lower bounds. A simple feasible lower bound can be obtained by considering terminal sets of bounded cardinality:

Definition 5. *Let (G, c, D) be an instance of the Steiner tree problem, $t \in D$ and let $j \geq 1$ be an integer. Then, $\mathcal{L}_j$ is defined as*

$$\mathcal{L}_j(v, I) = \max_{\{t\} \subseteq J \subseteq I \cup \{v\}, |J| \leq j+1} \operatorname{smt}(J)$$

for $v \in V(G)$ and $\{t\} \subseteq I \subseteq D$. For $v \in V(G)$ and $I \subseteq D$ with $t \notin I$, set $\mathcal{L}_j(v, I) = 0$.

Lemma 6. *Let (G, c, D) be an instance of the Steiner tree problem, $t \in D$ and let $j \geq 1$ be an integer. Then, $\mathcal{L}_j$ is a feasible lower bound. Furthermore, we can implement $\mathcal{L}_j$ such that after a preprocessing time of $\mathcal{O}(3^k + (2k)^{j-1} n + k^{j-1}(n \log n + m))$, we can evaluate $\mathcal{L}_j(v, I)$ for every $v \in V(G)$ and $\{t\} \subseteq I \subseteq D$ in time $\mathcal{O}(|I|^{j-1})$, where $n = |V(G)|$, $m = |E(G)|$ and $k = |D|$.*

Proof. Let $v, w \in V(G)$ and $\{t\} \subseteq I' \subseteq I \subseteq D$. To prove that $\mathcal{L}_j$ is a feasible lower bound, we have to show

$$\max_{\substack{\{t\}\subseteq J\subseteq I\cup\{v\}\\ |J|\le j+1}} \operatorname{smt}(J) \le \max_{\substack{\{t\}\subseteq J'\subseteq I'\cup\{w\}\\ |J'|\le j+1}} \operatorname{smt}(J') \qquad + \operatorname{smt}((I\setminus I')\cup\{v,w\}).$$

Consider the map $f : I \cup \{v\} \to I' \cup \{w\}$ with $f(x) = w$ for $x \notin I'$ and $f(x) = x$ otherwise. Let J be a set with $\{t\} \subseteq J \subseteq I \cup \{v\}$ and $|J| \le j + 1$. Set $J' = \{f(x) : x \in J\}$. Then, clearly $\{t\} \subseteq J' \subseteq I' \cup \{w\}$ and $|J'| \le |J| \le j + 1$. Moreover,

$$\operatorname{smt}(J) \le \operatorname{smt}(J') + \operatorname{smt}((I \setminus I') \cup \{v, w\}).$$

To achieve the given run time, we first compute $l(v, I)$ for all $v \in V(G)$ and $I \subseteq D, |I \setminus \{t\}| \le j - 1$ in time $\mathcal{O}((2k)^{j-1}n + k^{j-1}(n \log n + m))$ using a modified variant of the Dijkstra-Steiner algorithm. More precisely, we do not use a lower bound, do not use a root terminal and consider terminal sets of increasing cardinality, very similar to [10]. There are $\mathcal{O}(k^{j-1})$ sets I with $I \subseteq D, |I \setminus \{t\}| \le j-1$. Since we consider terminal sets of increasing cardinality one after another, we always have at most n labels in the Fibonacci heap. As a set of cardinality j has 2^j subsets, there are $\mathcal{O}((2k)^{j-1}n)$ updates of supersets.

Then, to evaluate $\mathcal{L}_j(v, I)$, we exploit

$$\begin{aligned}\mathcal{L}_j(v,I) &= \max_{\{t\}\subseteq J\subseteq I\cup\{v\}, |J|\le j+1} \operatorname{smt}(J)\\ &= \max\left(\max_{J\subseteq I, |J|\le j-1} \operatorname{smt}(J\cup\{v,t\}), \max_{\{t\}\subseteq J\subseteq I, |J|\le j+1} \operatorname{smt}(J)\right),\end{aligned}$$

where the first expression can be computed in $\mathcal{O}(|I|^{j-1})$ time using the precomputed values $l(v, J \cup \{t\})$ and the second expression does not depend on v and can be computed in advance for every $I \subseteq D$ in $\mathcal{O}(3^k)$ time, again using the precomputed values of l. □

Thus, for small j, e.g., $j \le 3$, $\mathcal{L}_j$ can be efficiently computed. In experiments without pruning, this lower bound is useful on low-dimensional instances like planar rectilinear grid graphs. However, pruning has a much larger impact and eliminates this effect.

We now present a more effective lower bound. We will use the notion of *1-trees*, which have long been studied [16] as a lower bound for the traveling salesman problem. Given a complete graph H with metric edge costs and a special vertex $v \in V(H)$, a 1-tree for v and H is a tree spanning on $H - v$ together with two additional edges connecting v with the tree. Since every tour consists of a path, which is a spanning tree, and a vertex connected to the endpoints of the path, every tour is a 1-tree. Thus, a 1-tree of minimum cost is a lower bound on the cost of a tour of minimum cost. Since such a tour of minimum cost is at most twice as expensive as a minimum Steiner tree, we can use 1-trees to get a lower bound for the Steiner tree problem.

For $v, w \in V(G)$, we denote by $d(v, w)$ the cost of a shortest path connecting v and w in G. Furthermore, for a set of vertices $X \subseteq V$, we denote by G_X the *distance graph* of X, which is the subgraph of the metric closure of G induced by X. Note that since the edge costs in G_X are the costs of shortest paths with respect to positive edge costs in G, the edge costs in G_X are always metric. Moreover, for $X \subseteq V(G)$, we denote by $\mathrm{mst}(X)$ the cost of a minimum spanning tree in G_X.

Definition 7. *Let (G, c, D) be an instance of the Steiner tree problem and $t \in D$. Then, the* 1-tree bound $\mathcal{L}_{\text{1-tree}}$ *is defined as*

$$\mathcal{L}_{\textit{1-tree}}(v, I) = \min_{i,j \in I : i \neq j \vee |I|=1} \frac{d(v,i) + d(v,j)}{2} + \frac{\mathrm{mst}(I)}{2}$$

for $v \in V(G)$ and $\{t\} \subseteq I \subseteq D$. For $v \in V(G)$ and $I \subseteq D \setminus \{t\}$, we set $\mathcal{L}_{\textit{1-tree}}(v, I) = 0$.

Lemma 8. *Let (G, c, D) be an instance of the Steiner tree problem and $t \in D$. Then, $\mathcal{L}_{\textit{1-tree}}$ is a feasible lower bound.*

Proof. Let $v, w \in V(G)$ and $\{t\} \subseteq I' \subseteq I \subseteq D$. We will show

$$2\mathcal{L}_{\text{1-tree}}(v, I) \leq 2\mathcal{L}_{\text{1-tree}}(w, I') + 2\,\mathrm{smt}((I \setminus I') \cup \{v, w\}),$$

which translates to

$$\begin{aligned} &\min_{i,j \in I : i \neq j \vee |I|=1} (d(v,i) + d(v,j)) + \mathrm{mst}(I) \\ \leq &\min_{i,j \in I' : i \neq j \vee |I'|=1} (d(w,i) + d(w,j)) + \mathrm{mst}(I') + 2\,\mathrm{smt}((I \setminus I') \cup \{v, w\}). \end{aligned}$$

Consider a minimum spanning tree T_1 in $G_{I'}$ and a Steiner tree T_2 for $(I \setminus I') \cup \{v, w\}$. Furthermore, let $j_1, j_2 \in I'$ with $j_1 \neq j_2 \vee |I'| = 1$. This setting is illustrated in Figure 2.
First, we construct a tour C in $G_{(I \setminus I') \cup \{v,w\}}$ of cost at most $2c(T_2)$ using the standard double tree argument: If we double each edge in T_2, the graph gets Eulerian and we can find a Eulerian cycle. If we visit the vertices in the order the Eulerian cycle visits them and skip already visited vertices, we obtain a tour of at most the same cost exploiting that the edge costs in the distance graph are metric.
We can decompose the tour into two paths P_1 and P_2 in $G_{I \cup \{v,w\}}$ with endpoints v and w such that $(I \setminus I') \cup \{v, w\} = V(P_1) \cup V(P_2)$ and $c(P_1) + c(P_2) = c(C)$. Now, for $i \in \{1, 2\}$, we define $P'_i = (V(P_i) \cup \{j_i\}, E(P_i) \cup \{\{w, j_i\}\})$, which is the path obtained by appending the edge $\{w, j_i\}$ to P_i.
If $I_1 := (I \setminus I') \cap V(P_1)$ is empty, let j'_1 be j_1, else, let j'_1 be the first terminal in I_1 when traversing P_1 from v to w.
Similarly, if $I_2 := (I \setminus I') \setminus V(P_1) \subseteq V(P_2)$ is empty, set $j'_2 = j_2$, else let j'_2 be the first terminal in I_2 when traversing P_2 from v to w.
Since I_1 and I_2 are disjoint and do not contain $j_1, j_2 \in I'$, we have

$$j'_1 = j'_2 \implies (j_1 = j'_1 = j'_2 = j_2 \wedge I \setminus I' = \emptyset),$$

which together with

$$j_1 = j_2 \implies |I'| = 1$$

implies that

$$j'_1 = j'_2 \implies |I| = 1.$$

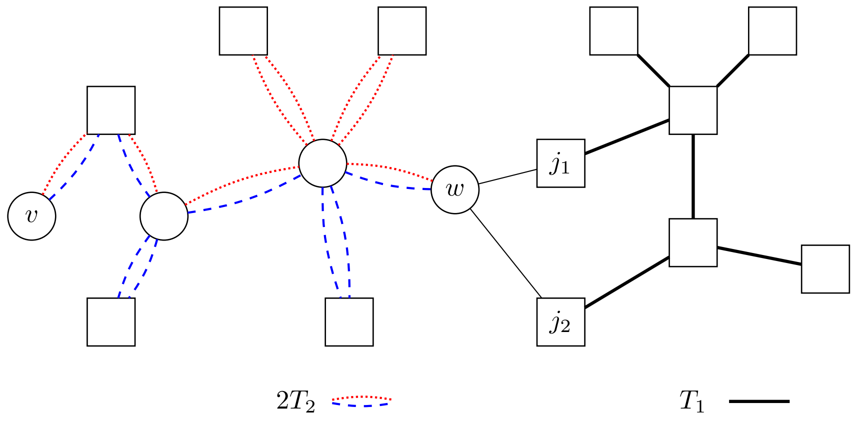

Figure 2: The minimum spanning tree T_1, the double Steiner tree $2T_2$ and the edges $\{w, j_1\}$ and $\{w, j_2\}$. Edges in $2T_2$ contributing to P_1 are colored red and edges contributing to P_2 are colored blue. Vertices in I are drawn as squares.

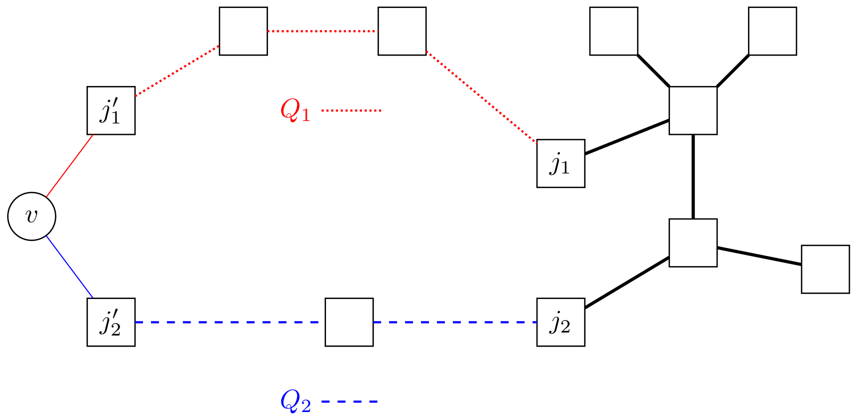

Figure 3: j'_1, j'_2, Q_1 and Q_2 in the same setting as in Figure 2.

Then, let Q_i be the path in G_I obtained from the subpath of P'_i from j'_i to j_i by skipping w. This is illustrated in Figure 3. We have

$$d(v, j'_1) + d(v, j'_2) + d(Q_1) + d(Q_2) \leq d(w, j_1) + d(w, j_2) + 2c(T_2).$$

Also, we have $(I \setminus I') \cup \{j_1, j_2\} \subseteq V(Q_1) \cup V(Q_2)$ and clearly, we can find a spanning tree in G_I with cost at most $d(Q_1) + d(Q_2) + d(T_1)$ by attaching Q_1 and Q_2 to T_1.
Therefore,

$$\begin{aligned}&\min_{i,j \in I : i \neq j \vee |I|=1} (d(v,i) + d(v,j)) + \mathrm{mst}(I)\\ &\leq d(v, j_1') + d(v, j_2') + d(Q_1) + d(Q_2) + d(T_1)\\ &\leq d(w, j_1) + d(w, j_2) + d(T_1) + 2c(T_2).\end{aligned}$$

□

Lemma 9. *Let (G, c, D) be an instance of the Steiner tree problem and $t \in D$. Then, we can implement $\mathcal{L}_{\text{1-tree}}$ such that after a preprocessing time of $\mathcal{O}(k(n \log n + m) + 2^k k^2)$, we can evaluate $\mathcal{L}_{\text{1-tree}}$ for every $v \in V(G)$ and $\{t\} \subseteq I \subseteq D$ in time $\mathcal{O}(|I|)$, where $n = |V(G)|$, $m = |E(G)|$ and $k = |D|$.*

Proof. First, for every terminal $s \in D$, we compute $d(v, s)$ for all $v \in V(G)$ in $\mathcal{O}(n \log n + m)$ time using Dijkstra's algorithm implemented with a Fibonacci heap [12]. For every $I \subseteq D$, we compute $\mathrm{mst}(I)$ in $\mathcal{O}(|I|^2)$ time using Prim's algorithm [24]. This results in a total preprocessing time of $\mathcal{O}(k(n \log n + m) + 2^k k^2)$. Clearly,

$$\min_{i,j \in I : i \neq j \vee |I|=1} (d(v,i) + d(v,j))$$

can be evaluated in $\mathcal{O}(|I|)$ time if $d(v, i)$ is known for all $v \in V(G)$ and $i \in I$. □

Theorem 10. *Let (G, c, D) be an instance of the Steiner tree problem, $t \in D$ and $j \geq 1$ be an integer. Then, we can compute $\mathrm{smt}(D)$ in time $\mathcal{O}(3^k n + 2^k(n \log n + m))$ using the Dijkstra-Steiner algorithm with $\mathcal{L} = \max(\mathcal{L}_j, \mathcal{L}_{\text{1-tree}})$.*
□

The 1-tree lower bound exploits the fact that 1-trees can be used to compute lower bounds on the minimum cost of a tour, which in turn is at most twice as expensive as a minimum cost Steiner tree. Using more preprocessing and evaluation time, we can eliminate the loss of approximating tours by 1-trees by using *optimum tours* to get lower bounds. While it may sound unreasonable to use optimum solutions for an NP-hard problem to speed up another algorithm, it turns out we can compute optimum tours for the union of sets of terminals and at most one vertex quite fast if there are only few terminals. This is due to the fact that the length of an optimum tour in $G_{I \cup \{v\}}$ only depends on the distances from terminals to v and shortest Hamiltonian paths with given endpoints in G_I, which can be computed in advance. For a set of vertices $X \subseteq V(G)$, we denote by $\mathrm{tsp}(X)$ the minimum cost of a Hamiltonian cycle in G_X.

Definition 11. *Let (G, c, D) be an instance of the Steiner tree problem and $t \in D$. Then, the* TSP bound *$\mathcal{L}_{TSP}$ is defined as*

$$\mathcal{L}_{TSP}(v, I) = \frac{\mathrm{tsp}(I \cup \{v\})}{2}$$

for $v \in V(G)$ and $I \subseteq D$.

Lemma 12. *Let (G, c, D) be an instance of the Steiner tree problem and $t \in D$. Then, $\mathcal{L}_{TSP}$ is a feasible lower bound. Moreover, after a preprocessing time of $\mathcal{O}(k(n \log n + m) + 2^k k^3)$, we can evaluate $\mathcal{L}_{TSP}(v, I)$ in time $\mathcal{O}(|I|^2)$ for all $v \in V(G)$ and $I \subseteq D$.*

Proof. Let $v, w \in V(G)$ and $\{t\} \subseteq I' \subseteq I \subseteq D$. We will show

$$2\mathcal{L}_{TSP}(v, I) \leq 2\mathcal{L}_{TSP}(w, I') + 2\,\mathrm{smt}((I \setminus I') \cup \{v, w\}),$$

which is equivalent to

$$\mathrm{tsp}(I \cup \{v\}) \leq \mathrm{tsp}(I' \cup \{w\}) + 2\,\mathrm{smt}((I \setminus I') \cup \{v, w\}).$$

First, we choose an optimal tour C_1 in $G_{I' \cup \{w\}}$. Then, we construct a tour C_2 in $G_{(I \setminus I') \cup \{v,w\}}$ of cost at most $2\,\mathrm{smt}((I \setminus I') \cup \{v, w\})$ by doubling the edges of an optimum Steiner tree, finding a Eulerian walk and taking shortcuts. We have $I \cup \{v\} = V(C_1) \cup V(C_2)$ and $w \in V(C_1) \cap V(C_2)$, so we can construct a tour in $G_{I \cup \{v\}}$ by inserting C_2 into C_1 after w and taking shortcuts, which results in a tour of cost of at most

$$\mathrm{tsp}(I' \cup \{w\}) + 2\,\mathrm{smt}((I \setminus I') \cup \{v, w\}).$$

We achieve the given run time using a dynamic programming approach very similar to the TSP algorithm by Held and Karp [15]. The idea is to compute shortest Hamiltonian paths in the distance graph of the terminals for all possible pairs of endpoints. Then, one can evaluate $\mathcal{L}_{TSP}(v, I)$ in $\mathcal{O}(|I|^2)$ time by enumerating all possible pairs of neighbors of v in the tour. □

While in most cases the exponential preprocessing time does not pay off, the TSP bound does improve the performance of our algorithm on instances with few terminals (< 12) and random underlying graphs.

4 Pruning

In this section, we present techniques to speed up the algorithm further by discarding labels (v, I) for which we can prove that they cannot contribute to an optimum solution. This affects the number of iterations, since these labels then are not chosen in line 5 of the algorithm. Also, it speeds up the execution of line 12, since we only have to consider existing labels in the merge step. First, we show how to identify labels that cannot contribute to an optimum solution. Then we show that we can indeed safely discard them in our algorithm.

Definition 13. *Let (G, c, D) be an instance of the Steiner tree problem, $(v, I) \in V(G) \times 2^D$ and T be a Steiner tree for D. A tree T_1 is said to be a (v, I)-*subtree of T *if there exists a tree T_2 such that*

(i) $V(T_1) \cup V(T_2) = V(T)$,

(ii) $V(T_1) \cap V(T_2) = \{v\}$,

(iii) T_1 *is a subtree of* T *containing* $\{v\} \cup I$ *and*

(iv) T_2 *is a subtree of* T *containing* $\{v\} \cup (D \setminus I)$.

For a tree T and a (v, I)-subtree T_1 of T, we will also refer to the corresponding subtree T_2 by $T - T_1$.

Lemma 14. *Let* (G, c, D) *be an instance of the Steiner tree problem and* $t \in D$. *Let* $\mathcal{L}$ *be a feasible lower bound and* $U \geq \text{smt}(D)$. *Furthermore, let* $v \in V(G)$, $I \subseteq D \setminus \{t\}$ *and* T_1 *be a tree in* G *containing* $\{v\} \cup I$ *with*

$$c(T_1) + \mathcal{L}(v, D \setminus I) > U.$$

Then, there is no optimum Steiner tree for D *containing* T_1 *as a* (v, I)*-subtree.*

Proof. Let T be a Steiner tree for D such that T_1 is a (v, I)-subtree of T. Then,

$$\begin{aligned}
c(T) &= c(T_1) + c(T - T_1) \\
&\geq c(T_1) + \text{smt}(\{v\} \cup (D \setminus I)) \\
&= c(T_1) + \text{smt}(\{v, t\} \cup ((D \setminus I) \setminus \{t\})) + \mathcal{L}(t, \{t\}) \\
&\geq c(T_1) + \mathcal{L}(v, D \setminus I) \\
&> U \\
&\geq \text{smt}(D).
\end{aligned}$$

□

Lemma 14 is a trivial exploitation of the lower bound $\mathcal{L}$. Its effect on the run time of the algorithm is rather limited, since we only discard labels that would never have been labeled permanently anyway. In contrast, the following lemma allows significant run time improvements of our algorithm, in particular on geometric instances. An application is illustrated in Figure 4.

Lemma 15. *Let* (G, c, D) *be an instance of the Steiner tree problem,* $v \in V(G)$, $I \subseteq D$ *and* $\emptyset \neq S \subseteq D \setminus I$. *Furthermore, let* T_1 *be a Steiner tree for* $\{v\} \cup I$ *and* Z *be a subgraph of* G *with*

(i) $(I \cup S) \subseteq V(Z)$,

(ii) *each connected component of* Z *contains a terminal in* S *and*

(iii) $c(Z) < c(T_1)$.

Then, there is no optimum Steiner tree T *for* D *in* G *containing* T_1 *as a* (v, I)*-subtree.*

Proof. Let T be a Steiner tree for D in G containing T_1 as a (v, I)-subtree. Then, there exists a tree T_2 containing $\{v\} \cup (D \setminus I)$ with $c(T) = c(T_1) + c(T_2)$. We construct a subgraph T' of G containing D by $T' = T_2 + Z$. As Z contains a path from every vertex in Z to some vertex in S, T_2 is connected and $S \subseteq D \setminus I \subseteq V(T_2)$, T' is connected. Thus,

$$\begin{aligned} \operatorname{smt}(D) &\leq c(T') \\ &\leq c(T_2) + c(Z) \\ &= c(T) - c(T_1) + c(Z) \\ &< c(T). \end{aligned}$$

□

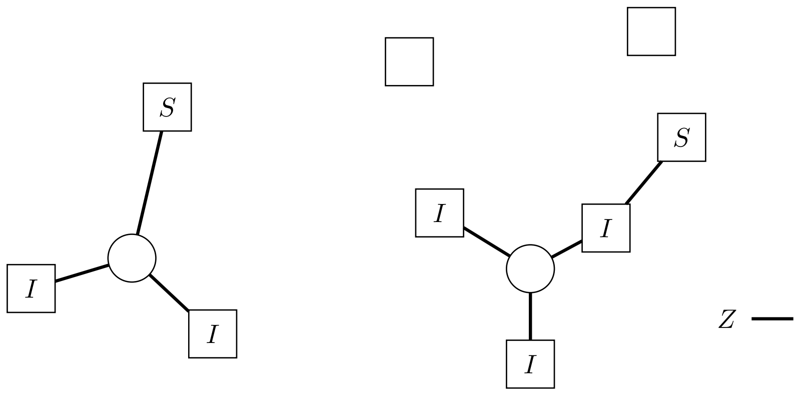

Figure 4: By Lemma 15, no label for the set I with cost strictly larger than $c(Z)$ can be part of an optimum solution. Terminals are drawn as squares, elements of I and S are labeled with the respective set.

Lemmata 14 and 15 allow us to identify labels that cannot contribute to an optimum solution. Theorem 17 shows that we can discard these labels without affecting the correctness of the algorithm. First, we prove an auxiliary lemma used in the proof of Theorem 17:

Lemma 16. *Let (G, c, D) be an instance of the Steiner tree problem and $t \in D$. Let $D' = D \setminus \{t\}$ and $O \subseteq V(G) \times 2^{D'}$ be the set of labels (v, I) with the property that there is an optimum Steiner tree T for D containing a (v, I)-subtree. Furthermore, let $(v, I) \in O$ and T_1 be a tree containing $\{v\} \cup I$ with $c(T_1) = \operatorname{smt}(\{v\} \cup I)$. Then, there is an optimum Steiner tree T for D containing T_1 as a (v, I)-subtree.*

Proof. Since $(v, I) \in O$, there is an optimum Steiner tree T' for D and a tree T_1' which is a (v, I)-subtree of T. Set $T = (T' - T_1') + T_1$. Then, since $T' - T_1'$ is a tree containing $\{v\} \cup (D \setminus I)$ and T_1 is a tree containing $\{v\} \cup I$, T is a connected subgraph of G containing $\{v\} \cup D$. Furthermore, we have

$$\begin{aligned}\operatorname{smt}(D) \leq c(T) &\leq c(T' - T_1') + c(T_1)\\ &= c(T' - T_1') + \operatorname{smt}(\{v\} \cup I)\\ &= c(T') - c(T_1') + \operatorname{smt}(\{v\} \cup I)\\ &\leq c(T')\\ &= \operatorname{smt}(D).\end{aligned}$$

Since we do not have edges of zero cost, this shows T is an optimum Steiner tree and T_1 is a (v, I)-subtree of T. □

We now formalize a general method of pruning:

```
Procedure prune
1 Choose a set $S \subset V(G) \times 2^{D'}$ such that for each $(v, I) \in S$, there is no
  optimum Steiner tree $T$ for $D$ such that backtrack$(v, I)$ is a $(v, I)$-subtree
  of $T$;
2 Set $l(v, I) := \infty$ for all $(v, I) \in S$;
```

Note that when considering a not permanently labeled element (v, I) with $l(v, I) < \infty$, we cannot guarantee that backtrack(v, I) is a tree, since it may contain cycles. However, if we are deciding whether we can prune the label we can assume backtrack(v, I) to be a tree, since otherwise the label can be pruned anyway.

Theorem 17. *The Dijkstra-Steiner algorithm still works correctly if we modify it to execute procedure prune before each execution of line 18.*

Proof. Let $O \subseteq V(G) \times 2^{D'}$ be the set of labels (v, I) with the property that there is an optimum Steiner tree T for D containing a (v, I)-subtree. It suffices that the modified algorithm is correct on O, which we will now prove.
To this end, consider the following invariants, which we will show to hold each time line 5 is executed in the modified algorithm:

(a') For each nonempty $I \subseteq D'$ and $v \in V(G)$ with $l(v, I) < \infty$:

- (a'1) $l(v, I) \leq \sum_{(w,I') \in b(v,I)} l(w, I') + \sum_{(w,I') \in b(v,I): w \neq v} c(\{v, w\})$,
- (a'2) $I \cup \{v\} = \{v\} \cup \dot{\bigcup}_{(w,I') \in b(v,I)} I'$,
- (a'3) backtrack(v, I) returns a connected subgraph T of G containing $\{v\} \cup I$ with $c(T) \leq l(v, I)$. If T is a tree, we have $c(T) = l(v, I)$.

(b') For each nonempty $I \subseteq D'$ and $v \in V(G)$ with $(v, I) \in P \cap O$:
$l(v, I) = \operatorname{smt}(\{v\} \cup I)$.

(c') For each nonempty $I \subseteq D'$ and $v \in V(G)$ with $(v, I) \notin P$:

- (c'1) $l(v, I) \geq \operatorname{smt}(\{v\} \cup I)$, (c'2) If $I = \{v\}$, then $l(v, I) = 0$. Otherwise, let

$$\begin{aligned} x_1 &= \min_{\{v,w\} \in \delta(v),(w,I) \in P} (l(w, I) + c(\{v, w\})), \\ x_2 &= \min_{I = I_1 \dot\cup I_2 \text{ and } (v,I_1),(v,I_2) \in P} (l(v, I_1) + l(v, I_2)), \\ x &= \min(x_1, x_2). \end{aligned}$$

If $x < \infty$, backtrack(v, I) returns a connected subgraph T of G containing $\{v\} \cup I$ with $c(T) \leq x$.

(c'3) If there is an optimum Steiner tree T for D such that backtrack(v, I) returns a (v, I)-subtree T_1 of T, we have $l(v, I) = c(T_1)$.

(d') There is a label (v, I) that can be chosen, i.e., $(V(G) \times 2^{D'}) \setminus P$ is not empty.

The difference between (a) – (d) as used in the proof of Theorem 2 and (a') – (d') are the weaker condition in (a'1), the restriction to labels $(v, I) \in O$ in (b') and the replacement of (c2) by (c'2) and (c'3). Since $(t, D') \in O$, the algorithm is correct assuming that these invariants hold and that we can always find a label $(v, I) \notin P$ in line 5 with $l(v, I) < \infty$.
It is easy to verify that these conditions hold after the initialization.
Lines 5 to 18 preserve (a') and (c'), which can be seen by the arguments given in the proof of Theorem 2. We have to show that lines 5 to 18 preserve (b') and (d') and that prune preserves (a') – (d').
Clearly, prune preserves (a'2) and (a'3), since (a') is only checked for labels (v, I) with $l(v, I) < \infty$. However, prune also preserves (a'1) since (a') is only checked for labels (v, I) with $l(v, I) < \infty$ and the right hand side of the inequality can only be increased by prune.
To verify prune preserves (b'), let $(v, I) \in P \cap O$. Before prune is called, we have by (b') that $l(v, I) = \text{smt}(\{v\} \cup I)$. Since $(v, I) \in O$, by Lemma 16 there is an optimum Steiner tree for D that contains the result of backtrack(v, I) as a (v, I)-subtree. Thus, prune cannot change $l(v, I)$.
Moreover, (c') is preserved by prune: Since prune only increases the l-values, (c'1) and (c'2) are clearly preserved. By the construction of prune, prune cannot change $l(v, I)$ for labels (v, I) satisfying (c'3).
As prune does not modify P, (d') is not affected by prune.
To show lines 5 to 18 preserve (b'), we can use the very same argument as in the proof of Theorem 2. We restrict ourselves to the case where Y is an optimum Steiner tree, which suffices. Then, we choose the triple (w, I', Y') the same way and see that by the choice of that triple we have using (c'2) and (b') that backtrack(w, I') returns a tree Y'' with $c(Y'') \leq c(Y') = \text{smt}(\{w\} \cup I')$. Then, by Lemma 16, there is an optimum Steiner tree for D containing backtrack(w, I') as a (w, I')-subtree, enabling us to use (c'3) to conclude that $l(w, I') = c(Y')$.
Using the same argument applied to (t, D') and an optimum Steiner tree Y for

D, we see that there always is a label $(v, I) \notin P$ with $l(v, I) < \infty$ which can be chosen in line 5 as long as $(t, D') \notin P$, so (d') is preserved as well. □

Of course, in practice we just avoid the creation of such labels instead of removing them immediately after creation.

5 Implementation and Results

We implemented the algorithm using the C++ programming language. In our implementation, we use a binary heap instead of a Fibonacci heap. We represent terminal sets by bitsets using the canonical bijection $2^D \to \{0, \dots, 2^{|D|} - 1\}$. For each vertex $v \in V(G)$, we maintain an array containing the labels (v, I) with $l(v, I) < \infty$ and a hash table storing for each label its index in the array, if it exists. This enables us to access labels very quickly and traverse over the existing labels in linear time, which is important for an efficient implementation of the merge step:

To implement line 12, we have two options. Either we explicitly enumerate all sets $I' \subseteq D' \setminus I$ and check whether the label (v, I') exists, or we traverse over all existing labels at v and omit the labels (v, I') with $I' \cap I \neq \emptyset$. We always choose the option resulting in less sets to be considered.

We implement the pruning rule of Lemma 14 using a shortest-paths Steiner tree heuristic similar to Prim's algorithm [24], maintaining and extending one component at a time. This takes $\mathcal{O}(k(n \log n + m))$ time. Then, we use the length of that Steiner tree as an upper bound and apply Lemma 14 each time we create a new label.

To implement Lemma 15, we maintain an upper bound $U(I)$ on the length of labels for each set $I \subseteq D'$ of terminals, which is initially set to infinity. For each occuring set $I \subseteq D'$, we compute the distance $d(I, D \setminus I) = \min_{x \in I, y \in D \setminus I} d(x, y)$. Then, each time we extract a label (v, I) from the heap, we update $U(I)$ by

$$U(I) := \min\left(U(I),\ l(v, I) + \min(d(I, D \setminus I), d(v, D \setminus I))\right).$$

Also, we keep track of the set $S(I)$ that was used to generate the currently best upper bound for the set I. In the routine described above, we always have $|S| = 1$. However, when merging two sets I_1 and I_2, we can use the sum of their upper bounds as an upper bound for the set $I_1 \cup I_2$ if $S(I_1) \cap I_2 = \emptyset$ or $S(I_2) \cap I_1 = \emptyset$, resulting in $S(I_1 \cup I_2) = (S(I_1) \cup S(I_2)) \setminus (I_1 \cup I_2)$.

Furthermore, we use the 1-tree bound as a lower bound. Of course, we do not compute minimum spanning trees for all subsets of terminals in advance. Instead, each time we consider a set I for the first time, we compute $\mathrm{mst}(D \setminus I)$.

Lacking a good selection strategy, we always choose the last terminal of the instance w.r.t. the order in the instance file as root terminal.

Now, we provide results of the algorithm on instances from the LIN testset in the SteinLib [20] which is the standard benchmark library for exact Steiner tree algorithms. We chose the LIN testset since it contains the hardest available VLSI-derived instances. These are derived from the placement of rectangles in the plane and only contain horizontal and vertical segments. Currently, our implementation is limited to 64 terminals, so we exclude lin28, lin33 and lin37. As a comparison, we use the best published results which were obtained by Polzin and Vahdati Daneshmand [21, 28] using one thread on a machine with 900 MHz SPARC III+ CPUs. Our results were achieved single-threaded on a computer with 3.33 GHz Intel Xeon W5590 CPUs, which is approximately six times faster. The run times by Polzin and Vahdati Daneshmand are given in the last column. Entries which are marked with a * could not be solved by Polzin and Vahdati Daneshmand using their default set of reductions. Instead, stronger reductions had to be used.

Instance	$\|V\|$	$\|E\|$	$\|D\|$	Opt	Time [s]	Time PV [s]
lin01	53	80	4	503	0.000	0.1
lin02	55	82	6	557	0.000	0.1
lin03	57	84	8	926	0.000	0.1
lin04	157	266	6	1239	0.000	0.1
lin05	160	269	9	1703	0.001	0.1
lin06	165	274	14	1348	0.002	0.1
lin07	307	526	6	1885	0.001	0.1
lin08	311	530	10	2248	0.001	0.1
lin09	313	532	12	2752	0.002	0.1
lin10	321	540	20	4132	0.012	0.1
lin11	816	1460	10	4280	0.007	0.2
lin12	818	1462	12	5250	0.010	0.3
lin13	822	1466	16	4609	0.010	0.2
lin14	828	1472	22	5824	0.013	0.2
lin15	840	1484	34	7145	0.065	0.2
lin16	1981	3633	12	6618	0.024	0.5
lin17	1989	3641	20	8405	0.060	0.7
lin18	1994	3646	25	9714	0.710	1.4
lin19	2010	3662	41	13268	15.391	1.4
lin20	3675	6709	11	6673	0.017	1.6
lin21	3683	6717	20	9143	0.047	1.2
lin22	3692	6726	28	10519	0.093	2.1
lin23	3716	6750	52	17560	16.686	2.9
lin24	7998	14734	16	15076	0.104	9.6
lin25	8007	14743	24	17803	0.359	12.6
lin26	8013	14749	30	21757	0.446	16.3
lin27	8017	14753	36	20678	2.494	13.7
lin29	19083	35636	24	23765	2.101	31.7
lin30	19091	35644	31	27684	0.920	87.2
lin31	19100	35653	40	31696	44.678	1002 *
lin32	19112	35665	53	39832	218.123	3559 *

Instance	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert D\rvert$	Opt	Time [s]	Time PV [s]
lin34	38282	71521	34	45018	15.988	9144 *
lin35	38294	71533	45	50559	35.522	8194 *
lin36	38307	71546	58	55608	66.686	763693 *

Table 1: Results on the LIN testset.

Even when compensating for hardware differences, our algorithm outperforms the reduction based approach on most of these instances, in particular on the instances having large underlying graphs. While the worst-case run time of our algorithm clearly is not reached on these instances, the results are consistent with the exponential dependence on the number of terminals and the quasilinear dependence on the size of the graph.

We have also done experiments with preprocessing using the publicly available implementation *bossa* [30]. However, in most cases the additional preprocessing time does not pay off.

The impact of future cost estimates and in particular pruning on practical run times depends a lot on the instance structure. For example, on instances where edges incident to terminals are very expensive, our pruning implementation has little impact. Examples for such instances are the so called incidence cost instances and group Steiner tree instances, which can be modeled as Steiner tree instances by introducing a terminal for each group and connecting the terminal to all vertices of the group using edges of very high cost. On these instances, other approaches still achieve clearly better results. However, one may find better lower bounds and pruning strategies that are effective on these instances.

Note that due to the dynamic programming nature of our algorithm, it can also be used to compute all optimum Steiner trees or even all Steiner trees up to a given length. If we enumerate all Steiner trees up to a length of $\mathrm{Opt} + \Delta$, we have to relax the pruning implementations by Δ and continue labeling until all labels (v, I) with $l(v, I) \leq \mathrm{Opt} + \Delta$ are permanent. Also, we have to save all predecessors instead of only one optimum predecessor. Then, we can recursively combine Steiner trees for subsets of terminals. In practice, the additional effort is linear in the size of the output, allowing the enumeration of millions of near-optimum Steiner trees in seconds. See [26] for details.

6 Conclusions

In this paper, we have presented a new algorithm based on dynamic programming to solve the Steiner tree problem in graphs to optimality. The algorithm combines a fast theoretical worst-case run time with competitive results on real-world instances. The dynamic programming idea by Dreyfus and Wagner has been used extensively to obtain Steiner tree algorithms with fast theoretical worst-case behavior. However, in the field of practical solving, it has rather been disregarded prior to this work. Compared to other exact algorithms, our algorithm depends much less on effective preprocessing and performs well on large graphs. Our approach is very general and not limited to the lower bounds and pruning strategies proposed in this paper.

References

[1] de Aragão, M., Uchoa, E., Werneck, R.: Dual heuristics on the exact solution of large Steiner problems. In: Proceedings of the Brazilian Symposium on Graphs, Algorithms and Combinatorics GRACO'01. Fortaleza (2001)

[2] Beasley, J.: An algorithm for the Steiner problem in graphs. Networks **14**, 147–159 (1984)

[3] Bodlaender, H.L., Cygan, M., Kratsch, S., Nederlof, J.: Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. In: F.V. Fomin, R. Freivalds, M. Kwiatkowska, D. Peleg (eds.) Automata, Languages, and Programming, *Lecture Notes in Computer Science*, vol. 7965, pp. 196–207. Springer Berlin Heidelberg (2013)

[4] Byrka, J., Grandoni, F., Rothvoß, T., Sanità, L.: Steiner tree approximation via iterative randomized rounding. J. ACM **60**(1), 6:1–6:33 (2013)

[5] Chlebík, M., Chlebíková, J.: The Steiner tree problem on graphs: Inapproximability results. Theoretical Computer Science **406**(3), 207 – 214 (2008)

[6] Chopra, S., Gorres, E., Rao, M.: Solving the Steiner tree problem on a graph using branch and cut. ORSA Journal on Computing **4**, 320–335 (1992)

[7] Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik **1**, 269–271 (1959)

[8] Dreyfus, S.E., Wagner, R.A.: The Steiner problem in graphs. Networks **1**(3), 195–207 (1971)

[9] Duin, C., Volgenant, A.: An edge elimination test for the Steiner problem in graphs. Operations Research Letters **8**, 79–83 (1989)

[10] Erickson, R.E., Monma, C.L., Veinott Jr., A.F.: Send-and-split method for minimum-concave-cost network flows. Math. Oper. Res. **12**(4), 634–664 (1987)

[11] Fafianie, S., Bodlaender, H., Nederlof, J.: Speeding up dynamic programming with representative sets. In: G. Gutin, S. Szeider (eds.) Parameterized and Exact Computation, *Lecture Notes in Computer Science*, vol. 8246, pp. 321–334. Springer International Publishing (2013)

[12] Fredman, M.L., Tarjan, R.E.: Fibonacci heaps and their uses in improved network optimization algorithms. J. ACM **34**(3), 596–615 (1987)

[13] Fuchs, B., Kern, W., Mölle, D., Richter, S., Rossmanith, P., Wang, X.: Dynamic programming for minimum Steiner trees. Theory of Computing Systems **41**(3), 493–500 (2007)

[14] Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics **SSC-4(2)**, 100–107 (1968)

[15] Held, M., Karp, R.M.: A dynamic programming approach to sequencing problems. Journal of the Society for Industrial and Applied Mathematics **10**(1), 196–210 (1962). DOI 10.2307/2098806

[16] Held, M., Karp, R.M.: The traveling salesman problem and minimum spanning trees. Operations Research **18**, 1138–1162 (1970)

[17] Held, S., Korte, B., Rautenbach, D., Vygen, J.: Combinatorial optimization in VLSI design. In: Combinatorial Optimization - Methods and Applications, pp. 33–96. IOS Press (2011)

[18] Karp, R.M.: Reducibility among combinatorial problems. In: R. Miller, J. Thatcher (eds.) Complexity of Computer Computations, pp. 85–103. Plenum Press (1972)

[19] Koch, T., Martin, A.: Solving Steiner tree problems in graphs to optimality. Networks **32**, 207–232 (1998)

[20] Koch, T., Martin, A., Voß, S.: SteinLib: An updated library on Steiner tree problems in graphs. In: D.Z. Du, X. Cheng (eds.) Steiner Trees in Industry, pp. 285–325. Kluwer Academic Publishers, Dordrecht (2001)

[21] Polzin, T.: Algorithms for the Steiner problem in networks. Ph.D. thesis (2004)

[22] Polzin, T., Vahdati, S.: Extending reduction techniques for the Steiner tree problem: A combination of alternative- and bound-based approaches. Tech. Rep. MPI-I-2001-1-007, Max-Planck-Institut für Informatik (2001)

[23] Polzin, T., Vahdati Daneshmand, S.: Practical partitioning-based methods for the Steiner problem. In: C. Àlvarez, M. Serna (eds.) Experimental Algorithms, *Lecture Notes in Computer Science*, vol. 4007, pp. 241–252. Springer Berlin Heidelberg (2006)

[24] Prim, R.C.: Shortest connection networks and some generalizations. Bell System Technology Journal **36**, 1389–1401 (1957)

[25] Robins, G., Zelikovsky, A.: Tighter bounds for graph Steiner tree approximation. SIAM Journal on Discrete Mathematics **19**, 122–134 (2005)

[26] Silvanus, J.: Fast exact Steiner tree generation using dynamic programming. Master's thesis, Research Institute for Discrete Mathematics, University of Bonn (2013)

[27] Uchoa, E., de Aragão, M., Ribeiro, C.: Preprocessing Steiner problems from VLSI layout. Tech. Rep. MCC 32/99, Catholic University of Rio de Janeiro, Rio de Janeiro (1999)

[28] Vahdati Daneshmand, S.: Algorithmic approaches to the Steiner problem in networks. Ph.D. thesis (2004)

[29] Vygen, J.: Faster algorithm for optimum Steiner trees. Inf. Process. Lett. **111**(21-22), 1075–1079 (2011)

[30] Werneck, R.F., Uchoa, E., Ribeiro, C.C., Poggi de Aragão, M.: bossa (2003). URL `http://www.cs.princeton.edu/~{}rwerneck/bossa/`. Accessed: February 2014

[31] Wong, R.: A dual ascent approach for Steiner tree problems on a directed graph. Mathematical Programming **28**(3), 271–287 (1984)