Zuse Institute Berlin

Takustr. 7
14195 Berlin
Germany

DANIEL REHFELDT, THORSTEN KOCH

Implications, conflicts, and reductions for Steiner trees

ZIB Report 20-28 (November 2020)

Zuse Institute Berlin
Takustr. 7
14195 Berlin
Germany

Telephone: +49 30-84185-0
Telefax: +49 30-84185-125

E-mail: bibliothek@zib.de
URL: http://www.zib.de

ZIB-Report (Print) ISSN 1438-0064
ZIB-Report (Internet) ISSN 2192-7782

Implications, conflicts, and reductions for Steiner trees

Daniel Rehfeldt, Thorsten Koch

TU Berlin, Chair of Software and Algorithms for Discrete Optimization,
Str. des 17. Juni 135, 10623 Berlin, Germany

Zuse Institute Berlin,
Takustr. 7, 14195 Berlin, Germany
{rehfeldt,koch}@zib.de

November 11, 2020

Abstract

The Steiner tree problem in graphs (SPG) is one of the most studied problems in combinatorial optimization. In the last 10 years, there have been significant advances concerning approximation and complexity of the SPG. However, the state of the art in (practical) exact solution of the SPG has remained largely unchallenged for almost 20 years. While the DIMACS Challenge 2014 and the PACE Challenge 2018 brought renewed interest into Steiner tree problems, even the best new SPG solvers cannot match the state of the art on the vast majority of benchmark instances.

The following article seeks to once again advance exact SPG solution. The article is based on a combination of three concepts: Implications, conflicts, and reductions. As a result, various new SPG techniques are conceived. Notably, several of the resulting techniques are (provably) stronger than well-known methods from the literature, used in exact SPG algorithms. Finally, by integrating the new methods into a branch-and-cut algorithm we obtain an exact SPG solver that is not only competitive with, but even outperforms the current state of the art on a large collection of benchmark sets. Furthermore, we can solve several instances for the first time to optimality.

1 Introduction

Given an undirected connected graph $G = (V, E)$, edge costs $c : E \to \mathbb{Q}_{>0}$ and a set $T \subseteq V$ of *terminals*, the *Steiner tree problem in graphs* (SPG) is to find a tree $S \subseteq G$ with $T \subseteq V(S)$ such that $c(E(S))$ is minimized. The SPG is a classic $\mathcal{NP}$-hard problem [23], and one of the most studied problems in combinatorial optimization. Part of its theoretical appeal might be attributed to the fact that the SPG generalizes two other classic combinatorial optimization problems: Shortest paths, and minimum spanning trees. On the practical side, many applications can be modeled as SPG or closely related problems, see e.g. [5, 27].

The SPG has seen numerous theoretical advances in the last 10 years, bringing forth significant improvements in complexity and approximability. See e.g. [4, 15] for approximation, and [24, 28,

45] for complexity results. However, when it comes to (practical) exact algorithms, the picture is significantly more bleak. After flourishing in the 1990s and early 2000s, algorithmic advances came to a staggering halt with the (joint) PhD theses of Polzin and Vahdati Daneshman almost 20 years ago [30, 44]. The authors introduced a wealth of new results and algorithms for SPG, and combined them in an exact solver that drastically outperformed all previous results from the literature. Their work is also published in a series of articles [31, 32, 33, 34, 35]. Unfortunately, however, their solver is not publicly available.

The 11th DIMACS Challenge in 2014, dedicated to Steiner tree problems, brought renewed interest to the field of exact algorithms. In the wake of the challenge, several new exact SPG solvers were introduced in the literature [13, 14, 17, 29]. Overall, the 11th DIMACS Challenge brought notable progress on the solution of notoriously hard SPG instances that had been designed to defy known solution techniques, see [26, 39]. Several of these instances could be solved for the first time to optimality. However, on the vast majority of instances from the literature, [30, 44] (whose solver did not compete at the DIMACS Challenge) stayed out of reach: For many benchmark instances, their solver is even two orders of magnitude or more faster, and it can furthermore solve substantially more instances to optimality—including those introduced at the DIMACS Challenge [36]. In 2018, the 3rd PACE Challenge [3] took place, dedicated to fixed-parameter tractable algorithms for SPG. Thus, the PACE Challenge considered mostly instances with a small number of terminals, or with small tree-width. Solvers that successfully participated in the PACE Challenge are for example described in [18, 21]. Still, even for these special problem types, the solver by [30, 44] remained largely unchallenged, see e.g. [21].

The following article aims to once again advance the state of the art in exact SPG solution.

1.1 Contribution

This article is based on a combination of three concepts: Implications, conflicts, and reductions. As a result, various new SPG techniques are conceived. The main contributions are as follows.

- By using a new implication concept, a distance function is conceived that provably dominates the well-known bottleneck Steiner distance. As a result, several reduction techniques that are stronger than results from the literature can be designed.
- We show how to derive conflict information between edges from the above methods. Further, we introduce a new reduction operation whose main purpose is to introduce additional conflicts. Such conflicts can for example be used to generate cuts for the IP formulation.
- We introduce a more general version of the powerful so-called extended reduction techniques. We furthermore enhance this framework by using both the previously introduced new distance concept, and the conflict information.
- Finally, we integrate the components into a branch-and-cut algorithm. Besides preprocessing, domain propagation, and cuts, also primal heuristics can be improved (by using the new implication concept). The practical implementation is realized as an extension of the branch-and-cut solver SCIP-Jack [14].

The resulting exact SPG solver outperforms the current state-of-the-art solver from [30, 44] on many well-established benchmark sets from the literature. Furthermore, it can solve several instances for the first time to optimality.

1.2 Preliminaries and notation

We write $G := (V, E)$ for an undirected graph, with vertices V and edges E. We set $n := |V|$ and $m := |E|$. We denote the vertices and edges of any subgraph $S \subseteq G$ by $V(S)$ and $E(S)$, respectively. For a walk W we likewise denote the set of vertices and the set of edges it contains by $V(W)$ and $E(W)$. For any $U \subseteq V$ we define the *cut* $\delta(U) := \{\{u, v\} \in E \mid u \in U, v \in V \setminus U\}$. We write $\delta_G(U)$ to emphasize that the cut is defined with respect to graph G. For $v \in V$ we write $\delta(v) := \delta(\{v\})$. For any $v \in V$ we define its *neighborhood* as $N(v) := \{w \in W \mid \{v, w\} \in \delta(v)\}$.

Given edge costs $c : E \mapsto \mathbb{Q}_{\geq 0}$, the triplet (V, E, c) is referred to as *network*. By $d(v, w)$ we denote the cost of a shortest path (with respect to c) between vertices $v, w \in V$. For any (distance) function $\tilde{d} : \binom{V}{2} \mapsto \mathbb{Q}_{\geq 0}$, and any $U \subseteq V$ we define the *$\tilde{d}$-distance graph* on U as the network

$$D_G(U, \tilde{d}) := (U, \binom{U}{2}, \tilde{c}), \tag{1}$$

with $\tilde{c}(\{v, w\}) := \tilde{d}(v, w)$ for all $v, w \in U$. If $\tilde{d}$ is the standard distance (i.e. $\tilde{d} = d$), we write $D_G(U)$ instead of $D_G(U, d)$. If a given $\tilde{d} : \binom{V}{2} \mapsto \mathbb{Q}_{\geq 0}$ is symmetric, we occasionally write, with a slight abuse of notation, $\tilde{d}(e)$ instead of $\tilde{d}(v, w)$ for an edge $e = \{v, w\}$.

2 From implications to reductions

Reduction techniques have been a key ingredient in exact SPG solvers, see e.g. [9, 25, 43, 32]. Among these techniques, the *bottleneck Steiner distance* introduced in [12] is arguably the most important one, being the backbone of several powerful reduction methods. This section introduces a (provably) stronger distance concept, and discusses several applications for improved reduction methods.

2.1 The bottleneck Steiner distance

Let P be a simple path with at least one edge. The *bottleneck length* [12] of P is

$$bl(P) := \max_{e \in E(P)} c(e). \tag{2}$$

Let $v, w \in V$. Let $\mathcal{P}(v, w)$ be the set of all simple paths between v and w. The *bottleneck distance* [12] between v and w is defined as

$$b(v, w) := \inf\{bl(P) \mid P \in \mathcal{P}(v, w)\}, \tag{3}$$

with the common convention that $\inf \emptyset = \infty$. Note that $b(v, w)$ is equal to the bottleneck length of the path between v and w on any minimum spanning tree of (G, c), as observed in [8].

Now consider the distance graph $D := D_G(T \cup \{v, w\})$. Let b_D be the bottleneck distance in D. Define the *bottleneck Steiner distance* or *special distance* [12] between v and w as

$$s(v, w) := b_D(v, w). \tag{4}$$

The arguably best known bottleneck Steiner distance reduction method is based on the following criterion, which allows for edge deletion [12].

Theorem 1. *Let $e = \{v, w\} \in E$. If $s(v, w) < c(e)$, then no minimum Steiner tree contains e.*

Note that bottleneck Steiner distances can be computed in polynomial time, but in practice (heuristic) approximations are used. See [32] for a state-of-the-art algorithm.

2.2 A stronger bottleneck concept

In the following we describe a generalization of the bottleneck Steiner distance. Initially, for an edge $e = \{v, w\}$ define the *restricted bottleneck distance* $\bar{b}(e)$ [32] as the bottleneck distance between v and w on $(V, E \setminus \{e\}, c)$.

The basis of the new bottleneck Steiner concept is formed by a node-weight function that we introduce in the following. For any $v \in V \setminus T$ and $F \subseteq \delta(v)$ define

$$p^+(v, F) = \max\left\{0, \sup\left\{\bar{b}(e) - c(e) \mid e \in \delta(v) \cap F, e \cap T \neq \emptyset\right\}\right\}. \tag{5}$$

We call $p^+(v, F)$ the *F-implied profit* of v. Note that $p^+(v, \{e\}) = \infty$ for an $e \in \delta(v)$ if and only if all Steiner trees contain e. The following observation motivates the subsequent usage of the implied profit. Assume that $p^+(v, \{e\}) > 0$ for an edge $e \in \delta(v)$. If a Steiner tree S contains v, but not e, then there is a Steiner tree S' with $e \in E(S')$ such that $c(E(S')) + p^+(v, \{e\}) \leq c(E(S))$.

Let $v, w \in V$. Consider a finite walk $W = (v_1, e_1, v_2, e_2, ..., e_r, v_r)$ with $v_1 = v$ and $v_r = w$. We say that W is a (v, w)-walk. For any $k, l \in \mathbb{N}$ with $1 \leq k \leq l \leq r$ define the subwalk $W(k, l) := (v_k, e_k, v_{k+1}, e_{k+1}, ..., e_l, v_l)$. W will be called *Steiner walk* if $V(W) \cap T \subseteq \{v, w\}$ and v, w are contained exactly once in W (the latter condition could be omitted, but has been added for ease of presentation). The set of all Steiner walks from v to w will be denoted by $\mathcal{W}_T(v, w)$. With a slight abuse of notation we define $\delta_W(u) := \delta(u) \cap E(W)$ for any walk W and any $u \in V$. Define the *implied Steiner cost* of a Steiner walk $W \in \mathcal{W}_T(v, w)$ as

$$c_p^+(W) := \sum_{e \in E(W)} c(e) \; - \sum_{u \in V(W) \setminus \{v,w\}} p^+\left(u, \delta(u) \setminus \delta_W(u)\right). \tag{6}$$

Further, set

$$P_W^+ := \{u \in V(W) \mid p^+\big(u, \delta(u) \setminus \delta_W(u)\big) > 0\} \cup \{v, w\}. \tag{7}$$

Define the *implied Steiner length* of W as

$$l_p^+(W) := \max\{c_p^+(W(v_k, v_l)) \mid 1 \leq k \leq l \leq r, \; v_k, v_l \in P_W^+\}. \tag{8}$$

Define the *implied Steiner distance* between v and w as

$$d_p^+(v, w) := \min\{l_p^+(W) \mid W \in \mathcal{W}_T(v, w)\}. \tag{9}$$

Note that $d_p^+(v, w) = d_p^+(w, v)$. At last, consider the distance graph $D^+ := D_G(T \cup \{v, w\}, d_p^+)$. Let b_{D^+} be the bottleneck distance in D^+. Define the *implied bottleneck Steiner distance* between v and w as

$$s_p(v, w) := b_{D^+}(v, w). \tag{10}$$

Note that $s_p(v, w) \leq s(v, w)$ and that the inequality can be strict. Indeed, $\frac{s(v,w)}{s_p(v,w)}$ can become arbitrarily large. Thus, the following result provides a strictly stronger reduction criterion than Theorem 1.

Theorem 2. *Let $e = \{v, w\} \in E$. If $s_p(v, w) < c(e)$, then no minimum Steiner tree contains e.*

Proof. Assume $s_p(v, w) < c(e)$ and let S be a Steiner tree with $e \in E(S)$. We will show the existence of a Steiner tree S' with $e \notin E(S')$ such that $c(E(S')) \leq c(E(S))$, which concludes the proof. First, remove e from S to obtain a new subgraph $\tilde{S}$, which consists of exactly two connected components. Assume that each connected component contains at least one terminal (otherwise the proof is already finished). Consider a (v, w)-path P in D^+ such that $bl_{D^+}(P) = b_{D^+}(v, w)$. Let $\{t, u\}$ be an edge on P such that t and u are in different connected components of $\tilde{S}$ (where

t and u are considered in the original SPG). Let $\tilde{S}^t$ and $\tilde{S}^u$ be the connected components of $\tilde{S}$ such that $t \in V(\tilde{S}^t)$ and $u \in V(\tilde{S}^u)$. By the definition of the bottleneck length it holds that

$$d_p^+(t,u) \leq s_p(v,w). \tag{11}$$

Let $W \in \mathcal{W}_T(t,u)$ such that

$$l_p^+(W) = d_p^+(t,u). \tag{12}$$

Assume that W is given as $W = (v_1, e_1, ..., e_r, v_r)$. Define $b := \min\{k \in \{1,...,r\} \mid v_k \in V(\tilde{S}^u)\}$ and $a := \max\{k \in \{1,...,b\} \mid v_k \in V(\tilde{S}^t)\}$. Further, define $x := \max\{k \in \{1,...,a\} \mid v_k \in P_W\}$ and $y := \min\{k \in \{b,...,r\} \mid v_k \in P_W\}$. By definition, $x \leq a < b \leq y$ and furthermore:

$$\sum_{e \in E(W(a,b))} c(e) - \sum_{v \in V(W(a,b)) \setminus \{v_x, v_y\}} p^+\big(v, \delta(v) \setminus \delta_{W(x,y)}\big) \leq c_p^+(W(x,y)). \tag{13}$$

Reconnect $\tilde{S}^t$ and $\tilde{S}^u$ by $W(a,b)$, which yields a connected subgraph S_0' with $T \subseteq V(S_0')$. Assume that S_0' is a tree (otherwise remove any redundant edges).[1] It holds that

$$\sum_{e \in E(S_0')} c(e) \leq \sum_{e \in E(S)} c(e) + \sum_{e \in E(W(a,b))} c(e) - c(\{v,w\}). \tag{14}$$

Let $v_1^+, v_2^+, ..., v_z^+$ be the vertices in $P_{W(a,b)} \setminus \{v_a, v_b\}$. Choose for each $i = 1, ..., z$ an edge $e_i^+ \in \delta(v_i^+) \setminus \delta_{W(x,y)}(v_i^+)$ such that $e_i^+ \cap T \neq \emptyset$ and

$$\overline{b}(e_i^+) - c(e_i^+) = p^+(v, \delta(v) \setminus \delta_{W(x,y)}). \tag{15}$$

Note that all e_i^+ are pairwise disjoint (just as the v_i^+).

We will construct Steiner trees S_i' for $i \in \{1, ..., z\}$ that satisfy

$$\sum_{e \in E(S_i')} c(e) \leq \sum_{e \in E(S_0')} c(e) - \sum_{k=1}^{i} p^+(v_k^+, \delta(v) \setminus \delta_{W(x,y)}). \tag{16}$$

and

$$\bigcup_{k=i+1}^{z} \{e_k^+\} \cap E(S_i') = \emptyset, \tag{17}$$

and

$$V(S_i') = V(S_0'). \tag{18}$$

One readily verifies that S_0' satisfies (16)-(18). Let $i \in \{1, ..., z\}$ and assume that (16)-(18) hold for S_{i-1}'. Thus, $e_i^+ \notin E(S_{i-1}')$. Let P_i be the (unique) path in S_{i-1}' between v_i^+ and the terminal t_i with $\{t_i\} = e_i^+ \cap T$. Choose any $\tilde{e}_i \in E(P)$ with $c(\tilde{e}_i) = bl(P_i)$. Define the tree S_i' by $V(S_i') := V(S_{i-1}')$ and $E(S_i') := \big(E(S_{i-1}') \setminus \{\tilde{e}_i\}\big) \cup \{e_i^+\}$. We claim that S_i' satisfies (16)-(18). Equality (17) follows from the fact that all e_i^+ are disjoint. And (18) follows from the construction of S_i'. For (16), observe that by definition of the bottleneck distance it holds that $c(\tilde{e}_i) \geq \overline{b}(e_i^+)$ and therefore

$$\overline{b}(e_i^+) - c(e_i^+) \leq c(\tilde{e}_i) - c(e_i^+). \tag{19}$$

Thus, equation (15) implies that S_i' satisfies (16).

[1] Because we assume all edges to be of positive cost, S_0' will in fact always be a tree.

Finally, set $S' := S'_z$. Because of (18) it holds that $T \subseteq V(S')$. Furthermore, one obtains:

$$\sum_{e \in E(S')} c(e) \overset{(16)}{\leq} \sum_{e \in E(S'_0)} c(e) - \sum_{k=1}^{z} p^+(v_k^+, \delta(v_k^+) \setminus \delta_{W(x,y)}) \tag{20}$$

$$\overset{(14)}{\leq} \sum_{e \in E(S)} c(e) + \sum_{e \in E(W(a,b))} c(e) - c(\{v,w\}) - \sum_{k=1}^{z} p^+(v_k^+, \delta(v_k^+) \setminus \delta_{W(x,y)}) \tag{21}$$

$$\overset{(13)}{\leq} \sum_{e \in E(S)} c(e) - c(\{v,w\}) + c_p^+(W(x,y)) \tag{22}$$

$$\overset{(12)}{\leq} \sum_{e \in E(S)} c(e) - c(\{v,w\}) + l_p^+(W) \tag{23}$$

$$\overset{(11)}{\leq} \sum_{e \in E(S)} c(e) - c(\{v,w\}) + s_p(v,w) \tag{24}$$

$$\leq \sum_{e \in E(S)} c(e), \tag{25}$$

where the last inequality follows from the initial assumptions. □

Furthermore, we define the *restricted implied bottleneck Steiner distance* $\overline{s}_p(v,w)$ between any $v, w \in V$ as the implied bottleneck Steiner distance between v and w in the SPG $(V, E \setminus \{\{v,w\}\}, c)$. One obtains the following corollary.

Corollary 3. *Let $e = \{v,w\} \in E$. If $\overline{s}_p(v,w) \leq c(e)$, then at least one minimum Steiner tree does not contain e.*

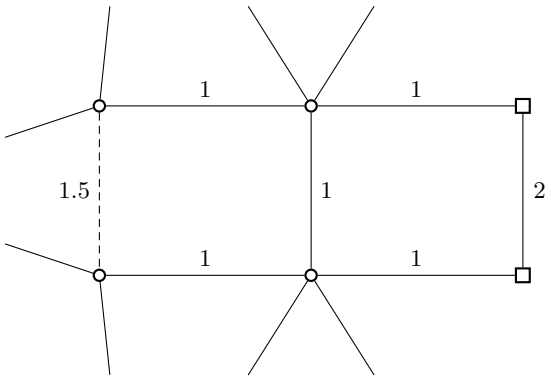

Figure 1: Segment of a Steiner tree instance. Terminals are drawn as squares. The dashed edge can be deleted by employing Theorem 2.

Figure 1 shows a segment of an SPG instance for which Theorem 2 allows for the deletion of an edge, but Theorem 1 does not. The implied bottleneck Steiner distance between the endpoints of the dashed edge is 1—corresponding to a walk along the four non-terminal vertices. The edge can thus be deleted. In contrast, the (standard) bottleneck Steiner distance between the endpoints is 1.5 (corresponding to the edge itself). Unfortunately, already computing the implied Steiner distance is hard, as the following proposition shows.

Proposition 4. *Computing the implied Steiner distance is $\mathcal{NP}$-hard.*

The proposition can for example be proved by a reduction from the Hamiltonian path problem, similar to a reduction for the prize-collecting Steiner distance concept in [42]. We note that it would also be possible to use the implied Steiner distance concept introduced in this article to generalize the Steiner distance concept used for the prize-collecting Steiner tree problem; see [37] for a definition that dominates the original one from [42]. However, formulating and proving this generalization is quite technical, and the computational benefit seems limited.

Finally, despite this $\mathcal{NP}$-hardness, one can devise heuristics that provide useful upper bounds on s_p. We will discuss one such heuristic in the next section.

2.3 Approximating the implied Steiner bottleneck distance

This sections describes one of the heuristics we use to delete edges by using an approximation of s_p. Starting from a vertex v_0, the heuristic tries to delete several edges of $\delta(v_0)$ at once. Initially, define a distance array $\tilde{d}$ and a predecessor array $pred$ as follows. For all $u \in V \setminus (\{v_0\} \cup N(v_0))$: $\tilde{d}[u] := \infty$ and $pred[u] := null$. For all $u \in N(v_0)$: $\tilde{d}[u] := c(\{v_0, u\})$ and $pred[u] := v_0$. Moreover, set $\tilde{d}[v_0] := 0$ and $pred[v_0] := v_0$. Finally, set $Q := N(v_0)$.

While $Q \neq \emptyset$ let $v := \arg\min_{u \in Q} \tilde{d}[u]$. For all $\{v, w\} \in \delta(v)$ proceed as follows. First, set $p_{vw} := \max\{p^+(v, \{e\}) \mid e \in \delta(v) : w, pred[v] \notin e\}$. If

$$\tilde{d}[v] + c(\{v, w\}) - \min\left\{c(\{v, w\}), p_{vw}, \tilde{d}[v]\right\} < \tilde{d}[w], \tag{26}$$

then set $\tilde{d}[w]$ to the left hand side of (26) and add w to Q. Further, set $pred[w] := v$. If (26) holds and $w \in N(v_0)$, then we can delete edge $\{v, w\}$.

Note that on the left hand side of (26) a possibly smaller value than p_{vw} is subtracted to prevent the algorithm from circling. Furthermore, note that a terminal might be used more than once for a profit calculation p_{vw} on one walk. However, since we subtract only a bounded part of the profit from the distance value in (26), the algorithm still works correctly. Note that one can extend the algorithm to cover the case of equality for edge deletion. In this case, one also needs to check whether (26) is satisfied with equality if $w \in N(v_0)$. In practice, one should bound the maximum number of visited edges. Additionally, one can abort the algorithm if $\min_{u \in Q} \tilde{d}[u] > \max_{e \in \delta(v_0)} c(e)$.

The above algorithm is also useful for finding a simple path between endpoints of an edge that is not longer than the edge itself. Other authors, e.g. [19, 32], suggest to run a shortest path algorithm from both endpoints of each edge of the given SPG for this purpose. However, running the above algorithm from each vertex is usually considerably faster in practice.

2.4 Bottleneck Steiner reductions beyond edge deletion

This section discusses applications of the implied bottleneck Steiner distance that allow for additional reduction operations: Edge contraction and node replacement. We start with the former. For an edge e and vertices v, w define $b_e(v, w)$ as the bottleneck distance between v and w on $(V, E \setminus \{e\}, c)$. With this definition, we define a generalization of the classic NSV reduction test from [10].

Proposition 5. *Let $\{v, w\} \in E$ and $t_i, t_j \in T, t_i \neq t_j$ such that: If*

$$s_p(v, t_i) + c(\{v, w\}) + s_p(w, t_j) \leq b_{\{v,w\}}(t_i, t_j), \tag{27}$$

then there is a minimum Steiner S with $\{v, w\} \in E(S)$.

Proof sketch. Unfortunately, the use of the implied bottleneck Steiner distance makes the proof of the proposition far more difficult than that of the original result from [10]. To avoid an abundance of technicalities, we therefore only provide a proof sketch.

Assume there is an optimal solution S such that $\{v, w\} \notin E(S)$. Remove from $E(S)$ an edge on the (unique) path between t_i and t_j in S of maximum cost. This operation results in two disjoint trees: S_i with $t_i \in S_i$ and S_j with $t_j \in S_j$. By definition of $b_{\{v,w\}}(t_i, t_j)$ it holds that

$$c(E((S_i)) + c(E(C(S_j)) + b_{\{v,w\}}(t_i, t_j) \leq C(S). \tag{28}$$

Now the sketchy part starts: Similar to the proof of Theorem 2, condition (27) allows us to connect S_i to v such that the resulting tree $\tilde{S}_i$ satisfies

$$c(E(\tilde{S}_i)) \leq c(E(S_i)) + s_p(v, t_i). \tag{29}$$

Equivalently, we can connect S_j to w with the result satisfying

$$c(E(\tilde{S}_j)) \leq c(E(S_j)) + s_p(w, t_j). \tag{30}$$

However, the above is only true, because the two Steiner walks that correspond to $s_p(v, t_i)$ and $s_p(w, t_j)$ in (29) and (30), respectively, have no vertex in common. If they had a vertex in common, one could build a new Steiner walk W_0 with $l_p^+(W_0) \leq s_p(v, t_i) + s_p(w, t_j)$ out of the two above Steiner walks, such that W_0 connects S_i and S_j. This walk W_0 could then be used to reconnect S_i and S_j to a Steiner tree of weight smaller than $b_{\{v,w\}}(t_i, t_j)$.

Finally, we define $\tilde{S}$ as the union of $\tilde{S}_i$, $\tilde{S}_j$, and $\{v, w\}$. This connected subgraph is not necessarily a tree, but can be made one without increasing $C(\tilde{S})$ by deleting an edge from each cycle. From 28, (29), and (30) it follows that

$$c(E(\tilde{S})) \leq c(E(S)), \tag{31}$$

which concludes the proof. □

If criterion (27) is satisfied, one can contract edge $\{v, w\}$ and make the resulting vertex a terminal. The original criterion from [10] uses the standard distance in (27) instead of the implied bottleneck Steiner distance. We note that using the (standard) bottleneck Steiner distance in (27) does not improve the original test. However, using the implied bottleneck Steiner distance leads to a strictly stronger criterion, as the example in Figure 2 shows. Note that $b_{\{t_1,v_1\}}(t_1, t_3) = 2$ and $s_p(v_1, t_3) = 1$. Thus, (27) is satisfied for edge $\{t_1, v_1\}$ and terminals t_1, t_3.

The following proposition allows one to identify edges that are candidates for edge contraction. Afterwards, the bottleneck distances can be computed for all these edges in $O(m + n \log n)$ amortized time [9].

Proposition 6. *Let $\{v, w\} \in E$ and $t_i, t_j \in T, t_i \neq t_j$. If (27) holds, then there is a minimum spanning tree S_{MST} on (V, E, c) such that $\{v, w\} \in E(S_{MST})$.*

Proof. Assume there is spanning tree S such that $\{v, w\} \notin S$. Remove from $E(S)$ an edge on the (unique) path between t_i and t_j in S of maximum cost. By definition of $b_{\{v,w\}}(t_i, t_j)$ it holds that

$$c(E(S_i)) + c(E(S_j)) + b_{\{v,w\}}(t_i, t_j) \leq c(E(S)). \tag{32}$$

This operation results in two disjoint trees: S_i with $t_i \in S_i$ and S_j with $t_j \in S_j$. If v and w are in different trees, one can add $\{v, w\}$ to connect S_i and S_j and obtain a spanning tree of no higher cost than S. Otherwise, assume that $v, w \in V(S_j)$. Let W_i be a Steiner walk from v to

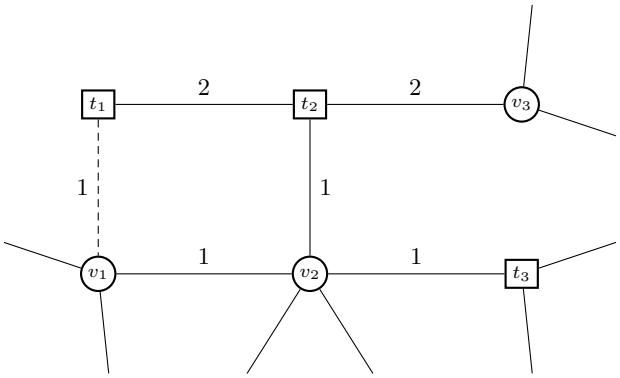

Figure 2: Segment of a Steiner tree instance. Terminals are drawn as squares. The dashed edge can be contracted by employing Proposition 5.

t_i with $l_p^+(W_i) = s_p(v, t_i)$. There is at least one edge $\{p, q\} \in E(W_i)$ such that $p \in V(S_i)$ and $q \in V(S_j)$. By definition it holds that $c(\{p, q\}) \leq l_p^+(W_i)$. Thus, one can add both $\{p, q\}$ and $\{v, w\}$ to S_i, S_j to obtain a connected spanning subgraph S'. Because of condition (27) and (32) it holds that

$$c(E(S')) \leq c(E(S)). \tag{33}$$

Delete any edge other than $\{v, w\}$ on the cycle in $E(S')$ that includes $\{v, w\}$. In this way one obtains a spanning tree S'' of no higher cost than S. □

This section closes with a reduction criterion based on the standard bottleneck Steiner distance. Besides being a new technique, this result also serves to highlight the complications that arise if one attempts to formulate similar conditions based on the implied bottleneck Steiner distance.

Proposition 7. *Let $D := D_G(T, d)$. Let Y be a minimum spanning tree in D. Write its edges $\{e_1^Y, e_2^Y, ..., e_{|T|-1}^Y\} := E(Y)$ in non-ascending order with respect to their weight in D. Let $v \in V \setminus T$. If for all $\Delta \subseteq \delta(v)$ with $|\Delta| \geq 3$ it holds that:*

$$\sum_{i=1}^{|\Delta|-1} d(e_i^Y) \leq \sum_{e \in \Delta} c(e), \tag{34}$$

then there is at least one minimum Steiner tree S such that $|\delta_S(v)| \leq 2$.

If the conditions (34) are satisfied for a vertex $v \in V \setminus T$, one can *pseudo-eliminate* [10] or *replace* [30] vertex v, i.e., delete v and connect any two vertices $u, w \in N(v)$ by a new edge $\{u, w\}$ of weight $c(\{v, u\}) + c(\{v, w\})$.

The SPG depicted in Figure 3 exemplifies why Proposition 7 cannot be formulated by using the implied Steiner distance. The weight of the minimum spanning tree Y for $D_G(T, d)$ is 4, but the weight of a minimum spanning tree with respect to the implied bottleneck Steiner distance is 2. Similarly also the BD_m reduction technique from [10] cannot be directly formulated by using the implied bottleneck distance. Still, it is possible to formulate a similar criterion that makes use of the implied bottleneck distance. Unfortunately, both the result and the corresponding proof are more involved than those of their edge elimination counterparts (see Theorem 2). Thus, we omit the details here. The important point is to make sure that the selected Steiner walks do

not overlap at vertices with a positive implied profit. However, these techniques have not been implemented yet.

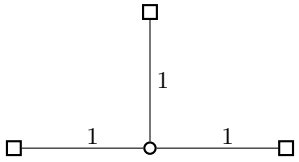

Figure 3: SPG instance. Terminals are drawn as squares

3 From reductions to conflicts

This section shows an additional advantage of the just introduced node replacement reduction: The creation of conflicts between the newly inserted edges. Furthermore, a new replacement operation is introduced.

3.1 Node replacement

Unfortunately, this section requires some additional technicalities regarding reduction methods. Recall that we have seen three types of reductions so far: Edge deletion, edge contraction, and node replacement. For simplicity, we assume in the following that a reduction is only performed if it retains *all* optimal solutions. E.g., we only delete an edge if we can shown that there is no minimum Steiner tree that contains this edge. We say that such a reduction is *valid*. We start with an SPG instance $I = (G, T, c)$, and consider a series of subsequent, valid reductions (of one of the three above types) that are applied to I. In each reduction step $i \geq 0$, the current instance $I^{(i)} = (G^{(i)}, T^{(i)}, c^{(i)})$ is transformed to instance $I^{(i+1)} = (G^{(i+1)}, T^{(i+1)}, c^{(i+1)})$. We set $I^{(0)} := I$. We define ancestor information for each $i = 0, 1, ..., k$ by $\Pi^{(i)} : E^{(i)} \to \mathcal{P}(E)$ and $\Pi^{(i)}_{FIX} \subseteq E$. We set $\Pi^{(0)}(e) := \{e\}$ for all $e \in E$, and $\Pi^{(0)}_{FIX} = \emptyset$. Consider a reduced instance $I^{(i)}$. If we contract an edge $e \in E^{(i)}$, we set $\Pi^{(i+1)}_{FIX} := \Pi^{(i)}_{FIX} \cup \Pi^{(i)}(e)$; otherwise, we set $\Pi^{(i+1)}_{FIX} := \Pi^{(i)}_{FIX}$. If we replace a vertex $v \in V^{(i)}$, we set for each newly inserted edge $\{u, w\}$—with $u, w \in N(v)$—$\Pi^{(i+1)}(\{u, w\}) := \Pi^{(i)}(\{v, u\}) \cup \Pi^{(i)}(\{v, w\})$. For all other remaining edges e we set $\Pi^{(i+1)}(e) := \Pi^{(i)}(e)$. Overall, one observes the following.

Observation 1. *Let I be an SPG and let $I^{(k)}$ be the SPG obtained from performing a series of k valid reductions on I. For any Steiner tree $S^{(k)}$ for $I^{(k)}$, the tree S with*

$$E(S) = \bigcup_{e \in E^{(k)}} \Pi^{(k)}(e) \cup \Pi^{(k)}_{FIX}$$

is a Steiner tree for I, and it holds that

$$c\left(E(S)\right) = c^{(k)}\left(E^{(k)}(S^{(k)})\right) + c\left(\Pi^{(k)}_{FIX}\right).$$

Furthermore, if $S^{(k)}$ is optimal for $I^{(k)}$, then S is optimal for I.

[33] observed that two edges that originate from a common edge by a series of replacements cannot both be contained in a minimum Steiner tree. Using the above notation, we can formulate

the condition as follows: If $e_1, e_2 \in E^{(k)}$ satisfy $\Pi^{(k)}(e_1) \cap \Pi^{(k)}(e_2) \neq \emptyset$, then there is no minimum Steiner tree that contains both e_1 and e_2. As we will see in Section 4, such conflict information can be used for further reductions.

In the following, we will introduce an edge conflict criterion that is strictly stronger than the one from [33]. Initially, we define additional ancestor information for each $i = 0, 1, ..., k$. Namely, sets of *replacement ancestors* $\Lambda^{(i)} : E^{(i)} \to \mathcal{P}(\mathbb{N})$, and $\Lambda^{(i)}_{FIX} \in \mathcal{P}(\mathbb{N})$. We set $\Lambda^{(0)}(e) := \emptyset$ for all $e \in E$, and $\Lambda^{(0)}_{FIX} := \emptyset$. Further, we define $\lambda^{(0)} := 0$. Consider a reduced instance $I^{(i)}$. If we contract an edge $e \in E^{(i)}$, we set $\Lambda^{(i+1)}_{FIX} := \Lambda^{(i)}_{FIX} \cup \Lambda^{(i)}(e)$. If we replace a vertex $v \in V^{(i)}$, we set $\lambda^{(i+1)} := \lambda^{(i)} + 1$. Further, we define the replacement ancestors for each newly inserted edge $\{u, w\}$, with $u, w \in N(v)$, as follows:

$$\Lambda^{(i+1)}(\{u, w\}) := \Lambda^{(i)}(\{v, u\}) \cup \Lambda^{(i)}(\{v, w\}) \cup \{\lambda^{(i)}\}.$$

If no node replacement is performed, we set $\lambda^{(i+1)} := \lambda^{(i)}$.

Proposition 8. *Let I be an SPG and let $I^{(k)}$ be the SPG obtained from performing a series of k valid reductions on I. Further, let $e_1, e_2 \in E^{(k)}$. If $\Lambda^{(k)}(e_1) \cap \Lambda^{(k)}(e_2) \neq \emptyset$, then no minimum Steiner tree $S^{(k)}$ for $I^{(k)}$ contains both e_1 and e_2.*

Proof. Suppose that there is a minimum Steiner tree $S^{(k)}$ with $e_1, e_2 \in E^{(k)}(S^{(k)})$. Let $x \in \Lambda^{(k)}(e_1) \cap \Lambda^{(k)}(e_2)$. Let i be the first reduction iteration with $\lambda^{(i)} = x$. We may assume that $i = 1$. Otherwise, we can define additional ancestor information $\overline{\Pi}$ and $\overline{\Lambda}$ starting from $I^{(i-1)}$, and perform the reductions from iteration i to iteration k. Let v be the vertex that is replaced in iteration $i = 1$. Note that $x = \lambda^{(1)} = 1$. From Observation 1 we know that the tree S defined by $E(S) = \bigcup_{e \in E^{(k)}} \Pi^{(k)}(e) \cup \Pi^{(k)}_{FIX}$ is a minimum Steiner tree for I. However, because of $\lambda^{(1)} \in \Lambda^{(k)}(e_1) \cap \Lambda^{(k)}(e_2)$, we have that $\left|\left(\Pi^{(k)}(e_1) \cup \Pi^{(k)}(e_2)\right) \cap \delta_S(v)\right| \geq 3$. This implies however, that replacing v is not valid—a contradiction. □

Corollary 9. *Let I, $I^{(k)}$ as in Proposition 8, and let $e \in E^{(k)}$. If $\Lambda^{(k)}(e) \cap \Lambda^{(k)}_{FIX} \neq \emptyset$, then no minimum Steiner tree $S^{(k)}$ for $I^{(k)}$ contains e.*

Note that any edge e as in Corollary 9 can be deleted.

3.2 Edge replacement

This subsection introduces a new replacement operation, whose primary benefit lies in the conflicts it creates. We start with a condition that allows us to perform this operation.

Proposition 10. *Let $e = \{v, w\} \in E$ with $e \cap T = \emptyset$. Define*

$$\mathcal{D} := \{\Delta \subseteq (\delta(v) \cup \delta(w)) \setminus \{e\} \mid \Delta \cap \delta(v) \neq \emptyset, \Delta \cap \delta(w) \neq \emptyset\}.$$

For any $\Delta \in \mathcal{D}$ let

$$U_\Delta := \{u \in V \mid \{u, v\} \in \Delta \vee \{u, w\} \in \Delta\}.$$

If for all $\Delta \in \mathcal{D}$ with $|\Delta| \geq 3$ the weight of a minimum spanning tree on $D_G(U_\Delta, s)$ is smaller than $c(\Delta)$, then each minimum Steiner tree S satisfies $\delta_S(v) \leq 2$ and $\delta_S(w) \leq 2$.

The proposition can be proven by using Corollary 12, which will be introduced in Section 4. If the condition of Proposition 10 is successful, we can perform what we will call a *path replacement* of e: We delete e and add for each pair $p, q \in V$ with $p \in N(v) \setminus \{w\}$, $q \in N(w)) \setminus \{v\}$, $p \neq q$ an edge $\{p, q\}$ with weight $c(\{p, v\}) + c(\{v, w\}) + c(\{q, w\})$. At first glance, the apparent increase in the

number of edges by this operations seems highly disadvantageous. However, due to the increased weight, the new edges can often be deleted by using the criterion from Theorem 2. Furthermore, an edge does not need to be inserted if any two of the three edges it originates from have a common replacement ancestor. Indeed, we only perform a path replacement if at most one of the new edges needs to be inserted. The case that all new edges can be deleted is in principle also covered by the extended reduction technique introduced in the next section (albeit being potentially far more expensive). If exactly one new edge remains, we create new replacement ancestors as follows: Let $\hat{e} = \{p, q\}$ be the newly inserted edge. Initially, set $\lambda^{(i+1)} := \lambda^{(i)}$ and $\Lambda^{(i+1)}(\hat{e}) := \Lambda^{(i)}(\{p, v\}) \cup \Lambda^{(i)}(\{v, w\}) \cup \Lambda^{(i)}(\{v, q\})$. Next, for each $e' \in (\delta(v) \cup \delta(w)) \setminus \{e\}$ increment $\lambda^{(i+1)}$, and add $\lambda^{(i+1)}$ to $\Lambda^{(i+1)}(\hat{e})$ and $\Lambda^{(i+1)}(e')$. One can show that Proposition 8 remains valid if path replacement is added to the list of valid reduction operations.

Figure 4 illustrates an application of Proposition 10. In this example, all but one replacement edges can be deleted by using a simple alternative path argument. While the number of edges remains unchanged, six new conflicts are created.

(a) SPG instance segment

(b) Segment after edge replacement

Figure 4: Segment of a Steiner tree instance (showing only non-terminals). All edges except for the dashed ones have unit weight. The dashed edge in (4a) has been replaced in (4b). All edges that are in conflict with the replacement edge in (4b) are drawn in bold.

4 From Steiner distances and conflicts to extended reduction techniques

At the end of the last section we have seen a reduction method that inspects a number of trees (of depth 3) that extend an edge considered for replacement. This section continues along this path, based on the reduction concepts introduced so far.

Given a tree Y (e.g. a single edge), *extended reduction techniques* use an enumeration of trees that contain Y to show that there is an optimal Steiner tree that does not contain Y. The trees are built by iteratively enlarging or *extending* Y. During this process, reduction, conflict, and implication techniques are employed to rule out these extensions of Y. In this way, extended reduction techniques are loosly related to the concepts of probing and conflict (graph) analysis for MIP, see e.g. [1, 40].

The idea of extension was first introduced in [46] for the rectilinear Steiner tree problem. Later the idea was adopted by [43] for the SPG. The next advancement came in [11], where backtracking was used, together with a number of new reduction criteria for the enumerated trees. Finally, [33] introduced the up-to-now strongest extended reduction techniques, which improved and complemented the previous results. The authors showed that their sophisticated

(a) Peripherally contained tree

(b) Not peripherally contained tree

Figure 5: Illustration of peripherally inclusion. The bold subtree is peripherally contained in the entire tree in Figure 5a, but not in Figure 5b.

algorithm could drastically reduce the size of many benchmark SPG instances, and even allowed for the solution of previously intractable instances.

In the following, we introduce new extended reduction algorithms that (provably) dominate those by [33].

4.1 The framework

For a tree Y in G, let $L(Y) \subseteq V(Y)$ be the set of its leafs. We start with several definitions from [33]. Let Y' be a tree with $Y' \subseteq Y$. The *linking set* between Y and Y' is the set of all vertices $v \in V(Y')$ such that there is a path $Q \subseteq Y$ from v to a leaf of Y with $V(Q) \cap V(Y') = \{v\}$. Note that Q can consist of a single vertex. Y' is *peripherally contained* in Y if the linking set between Y and Y' is $L(Y')$. Figure 5 exemplifies this concept. To motivate those definitions, consider a path Q without inner terminals between vertices v and w. For Q to not be peripherally contained in a minimum Steiner tree it is sufficient that $s(v, w)$ is smaller than the weight of Q. However, this condition is not sufficient to show that Q is not contained in a minimum Steiner tree. However, if Q is indeed contained in a minimum Steiner tree, at least one of its inner vertices needs to be of degree greater 2 in this tree. Thus, we can exploit this observation to enumerate extensions of Q from those inner vertices and attempt to rule those extensions out. Such kind of deductions are used in extended reduction techniques.

For any $P \subseteq V(Y)$ with $|P| > 1$ let Y_P be the union of the (unique) paths between any $v, w \in P \cup V(Y)$ in Y. Note that Y_P is a tree, and that $Y_P \subseteq Y$ holds. P is called *pruning set* if it contains the linking set between Y_P and Y. Additionally, we will use the following new definition: P is called *strict pruning set* if it is equal to the linking set between Y_P and Y. Figure 6 provides an example of pruning and strict pruning sets. One readily verifies the following property of pruning sets.

Observation 2. *Let Y be a tree, and let $Y' \subseteq Y$ be a tree that is peripherally contained in Y. Further, let $P \subseteq V(Y')$. If P is a pruning set for Y', then P is also a pruning set for Y. If P is a strict pruning set for Y', then P is also a strict pruning set for Y.*

Additionally, we define a stronger, and new, inclusion concept. Consider a tree $Y \subseteq G$, and a subtree Y'. Let P be a pruning set for Y'. We say that Y' is *P-peripherally contained* in Y if P is a pruning set for Y. Now let P be a strict pruning set for Y'. We say that Y' is *strictly*

(a) Pruning set

(b) Strict pruning set

Figure 6: Illustration of pruning and strict pruning sets. The filled vertices in Figure 6a form a (non-strict) pruning set, whereas the filled vertices in Figure 6b constitute a strict pruning set.

P-peripherally contained in Y if P is a strict pruning set for Y. From Observation 2 one obtains the following important property.

Observation 3. *Let $Y \subseteq G$ be a tree, let $Y' \subseteq Y$ be a subtree, and let P be a pruning set for Y'. If Y' is peripherally contained in Y, then Y' is also P-peripherally contained in Y.*

In fact, we will use the contraposition of the observation: If Y' is not P-peripherally contained in Y, then Y' is not peripherally contained in Y. Note that an equivalent property holds for strict pruning sets.

Given a tree Y and a set $E' \subseteq E$, we write with a slight abuse of notation $Y + E'$ for the subgraph with the edge set $E(Y) \cup E'$. Algorithm 1 shows a high level description of the extended reduction framework used in this article. The framework is similar to the one introduced in [33], but more general.[2] Note that the algorithm is recursive.

A possible input for Algorithm 1 is an SPG instance together with a single edge. If the algorithm returns *true*, the edge can be deleted. Besides ExtensionSets, which is described in Algorithm 2, the extended reduction framework contains the following subroutines:

- RuledOut(I, Y, P) is given an SPG $I = (G, T, c)$, a tree $Y \subseteq G$, and a pruning set P for Y such that $V(Y_P) \cap T \subseteq L(Y_P)$. The routine returns *true* if Y is shown to not be P-peripherally contained in any minimum Steiner tree. Otherwise, the routine returns $false$.
- RuledOutStrict(I, Y, P) is given an SPG $I = (G, T, c)$, a tree $Y \subseteq G$, and a strict pruning set P for Y such that $V(Y_P) \cap T \subseteq L(Y_P)$. The routine returns *true* if Y is shown to not be strictly P-peripherally contained in any minimum Steiner tree. Otherwise, the routine returns $false$.
- StrictPruningSets(I, Y) is given an SPG $I = (G, T, c)$, a tree $Y \subseteq G$. It returns a subset of all strict pruning sets for Y. A typical strict pruning set is $L(Y)$.
- Truncate(I, Y) is given an SPG $I = (G, T, c)$, and a tree $Y \subseteq G$. The routine returns *true* if no further extensions of Y should be performed; otherwise the routine returns $false$.

[2] We note, however, that the framework presented in [33] is (slightly) erroneous.

- PROMISING(I, Y, v) is given an SPG $I = (G, T, c)$, a tree $Y \subseteq G$, and a vertex $v \in L(Y)$. The routine returns *true* if further extensions of Y from v should be performed; otherwise the routine returns $false$.

The usage of P-peripheral inclusion in RULEDOUT might appear somewhat awkward, but is necessary for ruling-out not only trees (as in line 2 of Algorithm 1), but also all possible extension via a single edge (as in line 4 of Algorithm 2).

Algorithm 1: EXTENDED-RULEDOUT

```
   Data: SPG instance I = (G, T, c), tree Y with Y ∩ T ⊆ L(Y)
   Result: true if Y is shown to not be peripherally contained in any minimum Steiner
           tree; false otherwise
 1 foreach P ∈ STRICTPRUNINGSETS(I, Y) do
 2 |   if RULEDOUTSTRICT(I, Y, P) then return true
 3 end
 4 if TRUNCATE(I, Y) then return false
 5 foreach v ∈ L(Y) do
 6 |   if v ∈ T or not PROMISING(I, Y, v) then continue
 7 |   success := true
 8 |   foreach E' ∈ EXTENSIONSETS(I, Y, v) do
 9 |   |   if not EXTENDED-RULEDOUT(I, Y + E') then
10 |   |   |   success := false
11 |   |   |   break
12 |   |   end
13 |   end
14 |   if success then return true
15 end
16 return false
```

Algorithm 2: EXTENSIONSETS

```
   Data: SPG instance I = (G, T, c), tree Y, vertex v ∈ V(Y)
   Result: Set Γ ⊆ P(δ(v)) such that for all γ ∈ P(δ(v)) \ Γ, the tree Y + γ is not
           peripherally contained in any minimum Steiner tree.
 1 Q := ∅
 2 R := ∅
 3 foreach e := {v, w} ∈ δ(v) \ E(Y) do
 4 |   if RULEDOUT(I, Y + {e}, L(Y) ∪ {w}) then continue
 5 |   if RULEDOUTSTRICT(I, Y + {e}, L(Y) ∪ {w}) then
 6 |   |   R := R ∪ {e}
 7 |   |   continue
 8 |   end
 9 |   Q := Q ∪ {e}
10 end
11 return P(Q) ∪ R
```

In Lines 1-3 of Algorithm 1, we try to peripherally rule-out tree Y. If that is not possible, we try to recursively extend Y in Lines 5-15. Since (given positive edge weights) no minimum Steiner tree has a non-terminal leaf, we can extend from any of the non-terminal leaves of

Y. Note that ruling-out all extensions along one single leaf is sufficient to rule-out Y. The correctness of Extended-RuledOut can be proven by induction (under the assumption that the subroutines are correct). We also remark that it is under certain conditions possible to replace the condition *not peripherally contained in any minimum Steiner tree* by the condition *not peripherally contained in at least one minimum Steiner tree*. See also the discussion following Theorem 11.

Although the extended reduction framework shown in Algorithm 1 looks simple, an efficient realization is highly intricate. Not least, because the interaction of many different algorithmic components needs to be taken into account. Also, the re-use of intermediate results obtained during the tree extension (such as bottleneck Steiner distances) is non-trivial. Indeed, the implementation of extended reduction techniques for this article encompasses more than 20 000 lines of C code. We just note here that we have only implemented extensions in a depth-first-search manner. I.e., we extend only from leaves that are farthest away from the initial tree Y. A stronger, but potentially more expensive, alternative is to employ full backtracking, as partially done in [33]. In the following, we concentrate on mathematical descriptions of the subroutines for ruling-out enumerated trees.

4.2 Reduction criteria

In this section we introduce several elimination criteria used within RuledOut and RuledOutStrict. In fact, both of these routines consist of several subalgorithms that check different criteria for eliminating the given tree. Note that any criterion that is valid for RuledOut is also valid for RuledOutStrict. We also note that several of the criteria in this section are similar to results from [30, 33], but are all stronger. Throughout this section we consider a graph $G = (V, E)$ and an SPG instance $I = (G, T, c)$.

Consider a tree $Y \subseteq G$, and a pruning set P for Y such that $V(Y_P) \cap T \subseteq L(Y_P)$. For each $p \in P$ let $\overline{Y}_p \subset Y$ such that $V(\overline{Y}_p)$ is exactly the set of vertices $v \in V(Y_P)$ that satisfy the following: For any $q \in P \setminus \{p\}$ the (unique) path in Y from v to q contains p. Note that when removing $E(Y_P)$ from Y, each non-trivial connected component equals one $\overline{Y}_p$. Further, note that $p \in V(\overline{Y}_p)$ for all $p \in P$. Let $G_{Y,P} = (V_{Y,P}, E_{Y,P})$ be the graph obtained from $G = (V, E)$ by contracting for each $p \in P$ the subtree $\overline{Y}_p$ into p. For any parallel edges, we keep only one of minimum weight. We identify the contracted vertices $V(\overline{Y}_p)$ with the original vertex p. Overall, we thus have $V_{Y,P} \subseteq V$. Let $c_{Y,P}$ be the edge weights on $G_{Y,P}$ derived from c. Let

$$T_{Y,P} := \big(T \cap V_{Y,P}\big) \cup \{p \in P \mid T \cap V(\overline{Y}_p) \neq \emptyset\}. \tag{35}$$

Finally, let $s_{Y,P}$ be the bottleneck Steiner distance on $(G_{Y,P}, T_{Y,P}, c_{Y,P})$. With these definitions at hand, we are able to formulate a reduction criterion that generalizes a number of results from the literature. See [19, 30] for similar, but weaker, conditions.

Theorem 11. *Let $Y \subseteq G$ be a tree, and let P be a pruning set for Y such that $V(Y_P) \cap T \subseteq L(Y_P)$. Let $I_{Y,P}$ be the SPG on the distance network $D_{G_{Y,P}}\big(V_{Y,P}, s_{Y,P}\big)$ with terminal set P. If the weight of a minimum Steiner tree for $I_{Y,P}$ is smaller than $c(E(Y_P))$, then Y is not P-peripherally contained in any minimum Steiner tree for I.*

Proof. Let S be a (not necessarily minimum) Steiner tree for I such that Y is P-peripherally contained in S. Let $S_{Y,P}$ be a minimum Steiner tree for $I_{Y,P}$. Let $\tilde{S} \subset G$ be the forest defined as follows:

$$V(\tilde{S}) := (V(S) \setminus V(Y_P)) \cup V(S_{Y,P}), \tag{36}$$

$$E(\tilde{S}) := E(S) \setminus E(Y_P). \tag{37}$$

Let $\tilde{\mathcal{C}}$ be the set of connected components of $\tilde{S}$. Further, let $f : V \to \tilde{\mathcal{C}} \cup \{\emptyset\}$ such that $f(v) = \tilde{C}$ if $v \in V(\tilde{C})$ for a $\tilde{C} \in \tilde{\mathcal{C}}$, and $f(v) = \emptyset$ otherwise. Note that each $\tilde{C} \in \tilde{\mathcal{C}}$ contains at least one vertex of P, and thus also at least one vertex of $S_{Y,P}$. Also, $f(v) \neq \emptyset$ for all $v \in V(S_{Y,P})$. Further, note that for each of the contracted subtrees $\overline{Y}_p$ there is a $\tilde{C} \in \tilde{\mathcal{C}}$ with $\overline{Y}_p \subseteq \tilde{C}$. In the following, we will iteratively connect all the components in $\tilde{\mathcal{C}}$.

While $|\tilde{\mathcal{C}}| > 1$ proceed as follows. Choose a $(v, w) \in E(S_{Y,P})$ with $f(v) \neq f(w)$ such that $s_{Y,P}(v, w)$ is minimized. Let W be a (v, w)-walk in $G_{Y,P}$ corresponding to $s_{Y,P}(v, w)$. Because of $f(v) \neq f(w)$, there is at least one subwalk $Q = W(q, r)$ of W such that $f(q), f(r) \neq \emptyset$, $f(q) \neq f(r)$, and $f(u) = \emptyset$ for all $u \in V(Q) \setminus \{q, r\}$. Note that $c(E(Q)) \leq s_{Y,P}(v, w)$, because $f(t) \neq \emptyset$ for all $t \in T$. As long as such a path Q exists, proceed as follows. Add Q to $\tilde{S}$, and remove from $E(S_{Y,P})$ an (arbitrary) edge of the path between $f(q)$ and $f(r)$ in $S_{Y,P}$. Also, update $\tilde{\mathcal{C}}$ and f. Note that the weight of the removed edge (with respect to $s_{Y,P}$) is at most $s_{Y,P}(q, r)$.

Once $|\tilde{\mathcal{C}}| = 1$, one notes that the summed up weight of all newly inserted paths (with respect to c) does not exceed the weight of $S_{Y,P}$ (with respect to $s_{Y,P}$). Because the weight of $S_{Y,P}$ is smaller than $c(E(Y_P))$, we obtain from the construction of $\tilde{S}$ that

$$c(E(\tilde{S})) < c(E(S)), \tag{38}$$

which concludes the proof. □

In practice, one does not need to explicitly form $G_{Y,P}$. Instead, one can use the (original) bottleneck Steiner distances between the connected components of the graph induced by $E(Y) \setminus E(Y_P)$. Note that one can also extend Theorem 11 to the case of equality if at least one vertex of Y_P is not contained in any of the paths corresponding to the s values used for edges of $S_{Y,P}$. However, in the context of extended reduction techniques one needs to be careful to not discard all of several equivalent extensions. We omit the quite technical details, but merely note that allowing for equality (and adding suitable checks) can have a significant impact for some instances.

In practice, computing a minimum Steiner tree (or even an approximation) on $D_{G_{Y,P}}\big(V_{Y,P}, s_{Y,P}\big)$ is often too expensive. In such cases, the following corollary provides a strong alternative.

Corollary 12. *Let Y, P as in Theorem 11. Let (P', P'') be a partition of P. Let F' be a minimum spanning tree on $D_{G_{Y,P}}\big(P', s_{Y,P}\big)$, and let z' be the weight of F'. Let F'' be a minimum spanning tree in $D_{G_{Y,P}}\big(T_{Y,P}, s_{Y,P}\big)$. Write $\{e_1^{F''}, e_2^{F''}, ..., e_{|P''|-1}^{F''}\} := E_{Y,P}(F)$ such that $s_{Y,P}(e_i^{F''}) \geq s_{Y,P}(e_j^{F''})$ for $i < j$. Define*

$$z'' := \sum_{i=1}^{|P''|} s_{Y,P}(e_i^{F''}). \tag{39}$$

If $z' + z'' < c(E(Y_P))$, then Y is not P-peripherally contained in any minimum Steiner tree for I.

Proof. First, note that if $P'' = \emptyset$, then the corollary follows directly from Theorem 11, because z' is a lower bound on the weight of a minimum Steiner tree in $I_{Y,P}$. Thus, we assume $P'' \neq \emptyset$ in the following.

Suppose there is a minimum Steiner tree S for I such that Y is P-peripherally contained in S. Define $\tilde{S}$ as in the proof of Theorem 11. Further, proceed as in the proof of Theorem 11 to reconnect all connected components of $\tilde{S}$ that contain a vertex from P'. As a result, $\tilde{S}$ has

at most $|P''|+1$ connected components. Because S is assumed to be optimal, each connected component of $\tilde{S}$ contains at least one terminal. Thus, we can reconnect the remaining connected components similarly to Theorem 11, by using paths corresponding to edges of F''. We need to add at most $|P''|$ such paths. Overall, we have increased the weight of $\tilde{S}$ by at most $z'+z''$. From $z'+z'' < c(E(Y_P))$ we obtain that

$$c(E(\tilde{S})) < c(E(S)), \tag{40}$$

which contradicts the optimality of S. □

As for Theorem 11, the contractions in Corollary 12 should only be performed implicitly in practice. Furthermore, one requires a careful implementation to avoid a recomputation from scratch of the two minimum spanning trees in Corollary 12 for each enumerated tree in Algorithm 1.

Next, let $Y \subseteq G$ be a tree with pruning set P, and let $v, w \in V(Y)$ and let Q be the path between v, w in Y. We define a *pruned tree bottleneck* between v and w as a subpath $Q(a,b)$ of Q that satisfies $|\delta_Y(u)| = 2|$ and $u \notin P$ for all $u \in V(Q(a,b)) \setminus \{a,b\}$, $V(Q(a,b)) \cap T \subseteq \{a,b\}$, and maximizes $c(V(Q(a,b)))$. The weight $c(V(Q(a,b)))$ of such a pruned tree bottleneck is denoted by $b_{Y,P}(v,w)$. Using this definition and the implied bottleneck Steiner distance, we obtain the following result.

Proposition 13. *Let Y be a tree, let P be a pruning set for Y, and let $v, w \in V(Y)$. If $s_p(v,w) < b_{Y,P}(v,w)$, then Y is not P-peripherally contained in any minimum Steiner tree.*

The proposition can be proven in a similar way as Theorem 2 (and is indeed a generalization of the latter).

Another criterion can be devised by using the reduced costs of the well-known bidirected cut formulation [47] for SPG. This formulation is based on the observation that any optimal Steiner arborescence for the bidirected equivalent of a given SPG instance with arbitrary root $r \in T$ corresponds to an optimal Steiner tree for the original SPG. Let $D = (V, A)$ be the bidirected equivalent of G, and let $r \in T$. Consider a dual solution for the bidirected cut formulation, with reduced costs $\tilde{c}$, and with objective value $\tilde{L}$. Further, for any $v, w \in V$, let $\tilde{d}(v,w)$ be the length for a shortest, directed path from v to w in A with respect to the reduced costs. From the observation that an optimal Steiner arborescence cannot contain any cycles, we obtain the following result with standard linear programming arguments:

Proposition 14. *Let Y be a tree. Let $P = \{p_1, ..., p_k\}$ be a strict pruning set for Y such that there is a $k' \leq k$ with $p_i \in T$ if and only if $i > k'$. Further, assume that $V(Y_P) \cap T \subseteq L(Y_P)$, and $|P| < |T|$. The weight of any Steiner tree that strictly P-peripherally contains Y is at least*

$$\tilde{L} + \min_{i \in \{1,...,k\}} \max_{\{t_1,..t_{i-1},t_{i+1},...,t_{k'}\} \subseteq T \setminus V(Y_P)} \{\tilde{d}(r,p_i) + \sum_{j \leq k', j \neq i} \tilde{d}(p_j,t_j)\} \tag{41}$$

Given an upper bound on the cost of a minimum Steiner tree, this proposition can be used in the RuleOutStrict routine. In practice, we only use a lower bound on the max subterm in (41).

Finally, another important reduction criteria is constituted by edge conflicts—this result follows directly from Proposition 8.

Corollary 15. *Let $I^{(k)}$ be an SPG obtained from performing a series of k valid reductions on an SPG I. Let $Y \subseteq G^{(k)}$ be a tree, and let P a pruning set for Y. If there are distinct edges $e_1, e_2 \in E^{(k)}(Y)$ such that $\Lambda^{(k)}(e_1) \cap \Lambda^{(k)}(e_2) \neq \emptyset$, then Y is not P-peripherally contained in any minimum Steiner tree.*

5 Exact solution

This section describes how to use the techniques introduced so far for the exact solution of SPG. The new methods have been implemented as an extension of the branch-and-cut solver SCIP-Jack [14].

5.1 Branch-and-cut

As shown in [32], reduction techniques are the most important ingredient in a state-of-the-art SPG solver. While [32] uses linear programming and branch-and-bound mostly to trigger further reductions, we employ a proper branch-and-cut approach, based on [14]. We enhance several vital components of branch-and-cut algorithms. The most natural application of reduction methods is within presolving. However, one can also use them within domain propagation, translating the deletion of edges into variable fixings in the integer programming model. The edge conflicts described in this article can be used for generating clique cuts, which are well-known for general MIPs [2]. Finally, also primal heuristics are improved. First, the stronger reduction methods enhance primal heuristics that involve the solution of auxiliary SPG instances, such as from the combination of several Steiner trees. Second, the implication concept introduced in this article can be used to directly improve a classic SPG heuristic, as shown in the following.

Implications and the shortest path heuristic

The simple 2-approximation for SPG introduced by [41] has been widely used in the literature and is perhaps the best known primal heuristic for SPG. The algorithm starts with a tree S consisting of a single vertex and iteratively connects S by a shortest paths to a terminal closest to S. As a simple postprocessing step, one can compute a minimum spanning tree on $(V(S), E[S])$ and iteratively remove non-terminal leaves. An efficient implementation is given in [6]. This section shows how to use the implication concept introduced in Section 2.2 to (empirically) improve the algorithm.

Let $v_0 \in V$, and initially set $S := \{v_0\}$. Define a distance array $\tilde{d}$ and a predecessor array $pred$ by $\tilde{d}[u] := \infty$, $pred[u] := null$ for all $u \in V \setminus \{v_0\}$, and $\tilde{d}[v_0] := 0$, $pred[v_0] := v_0$. Define for all $v \in V \setminus T$:

$$\tilde{p}(v) := \max\left\{0, \sup\left\{\bar{b}(e) - c(e) \mid e = \{v, w\} \in \delta(v), w \in T \setminus V(S)\right\}\right\}. \tag{42}$$

For all $v \in T$ set $\tilde{p}(v) := 0$. Essentially, (42) is a weaker version of the implied profit from Section 2.2. Finally, set $Q := \{v_0\}$.

While $Q \neq \emptyset$ let $v := \arg\min_{u \in Q} \tilde{d}[u]$. If $v \in T$, add the path P from v to S, marked by the predecessor array, to S, add $V(P)$ to Q, and set $\tilde{d}[u] := 0$ for all $u \in V(P)$. Furthermore, update (42). For all $\{v, w\} \in \delta(v)$ proceed as follows. If

$$\tilde{d}[v] + c(\{v, w\}) - \min\left\{c(\{v, w\}), \tilde{p}(v), \tilde{d}[v]\right\} < \tilde{d}[w], \tag{43}$$

then set $\tilde{d}[w]$ to the left hand side of (43), and add w to Q. Further, set $pred[w] := v$.

Note that (43) provides a bias for paths computed by the heuristic to include vertices of implied profit. In this way, the distance associated with a path also reflects the cost needed to connect additional terminals later on. Note that the minimum spanning tree computed during postprocessing will always contain the edge associated with each vertex of positive implied profit contained in S. For performance reasons, we use in (42) instead of $\bar{b}(e)$ for $e = \{v, w\}, w \in T$, the value $\min_{e' \in \delta(w) \setminus \{e\}} c(e')$.

Computational experiments on the benchmark instances from the next section have shown that the above modifications improve the solution quality of the shortest path heuristic in a surprisingly consistent manner: When run 100 times from different starting points after SPG presolving (as is the default in SCIP-Jack), the solution quality of the heuristic is improved for more than 85 % of the instances. We also note that the shortest path heuristic is used as a subroutine in several more involved heuristics applied by SCIP-Jack, see [14].

5.2 Computational results

This section provides computational results for the new solver. In particular, we compare its performance with the updated result of the solver by [30, 44] published in [36]. The computational experiments were performed on Intel Xeon CPUs E3-1245 with 3.40 GHz and 32 GB RAM. According to the DIMACS benchmark software [7], this computer is 1.59 times faster than the machine used in [36]. While the authors of the current article do not have access to the machine used in [36], preliminary experiments on different machines have shown that the DIMACS score is a good estimate for the performance of the new solver. Thus, we have scaled the run-times reported in the following accordingly. We use the same LP solver as [36]: CPLEX 12.6 [20]. All results were obtained single-threaded. For the comparison we use a diverse range of well-established benchmark sets from the SteinLib [26] and the 11th DIMACS Challenge, see Table 1. The second columns gives the number of instances of the test-set. The third and fourth column give the range of $|V|$ and $|E|$ for the instances of the test-set. *Status* denotes whether a benchmark set contains yet unsolved instances.

Table 1: Details on benchmark sets.

Name	# Instances	$\|V\|$	$\|E\|$	Status	Description
E	20	2500	3125 - 62500	solved	Sparse, random graphs from OR-Library [16].
2R	27	2000	11600	solved	3-D cross grid graphs from SteinLib.
ALUE	15	940 - 34479	1474 - 55494	solved	Grid graphs with holes (non-geometric) from VLSI design. Introduced in [25].
vienna-s	85	1991 - 89596	3176 - 148583	solved	Instances derived from telecommunication network design. Introduced in [27]
ES10000FST	1	27019	39407	solved	Originally rectilinear Steiner tree instances. From SteinLib.
SP	8	6 - 3997	9 - 10278	solved	Constructed hard instances, combination of odd-wheels and odd cycles. From SteinLib.
GEO-adv	23	7565 - 71184	11521 - 113616	solved	Instances derived from telecommunication network design. Introduced in [27].
Cophag14	21	16 - 15473	23 - 93394	solved	Originally obstacle-avoiding rectilinear Steiner tree instances. From 11th DIMACS Challenge.
WRP3	63	84 - 3168	149 - 6220	solved	Instances derived from wire-routing problems. Introduced in [48].
PUC	50	64 - 4096	192 - 28512	unsolved	Constructed hard instances, (almost) uniform hypercubes, and bipartite graphs. From [39].

First, the impact of the s_p based reduction methods on the preprocessing strength is reported. For the reduced cost based reductions we use the dual-ascent heuristic from [47]. We use six benchmark sets from the literature; three from the DIMACS Challenge, and three from the SteinLib. Table 2 shows in the first column the name of the test-set, followed by its number of instances. The next columns show the percentual average number of nodes and edges of the instances after the preprocessing without (column three and four), and with (columns five and six) the s_p based methods. The last two columns reports the percentual relative change between the previous results. It can be seen that the s_p methods allow for a significant additional reduction of the problem size. This behavior is rather remarkable, given the variety of powerful reduction methods already included in SCIP-Jack. Furthermore, the instances from the *GEO-*

adv set come already in preprocessed form; by means of the reduction package from [43]. While no run-times are reported in the table, we note that the overall run-time of the preprocessing notably decreases when the s_p based methods are used. Furthermore, even for other test-sets where the s_p methods are less successful, one does not observe an increase in the run-time of the preprocessing.

Table 2: Average remaining nodes and edges after preprocessing.

		base preprocessing		$+s_p$ techniques		relative change	
Test set	#	nodes [%]	edges [%]	nodes [%]	edges [%]	nodes [%]	edges [%]
ALUE	15	1.4	1.4	0.8	0.9	**-75.0**	**-55.6**
vienna-s	85	8.5	8.1	6.1	5.8	**-39.3**	**-39.6**
WRP3	63	49.5	50.6	45.6	45.8	**-8.6**	**-10.4**
Copenhag14	21	39.4	37.1	35.4	32.1	**-11.3**	**-15.6**
GEO-adv	23	29.1	30.7	26.9	28.2	**-8.2**	**-8.9**
ES10000FST	1	47.2	52.1	38.1	42.2	**-23.9**	**-21.1**

Next, we compare the solver by [30, 44] and the new solver with respect to the mean time, the maximum time, and the number of solved instances. For the mean time we use the well-established shifted geometric mean [2] with a shift of 10. Note that the use of an arithmetic mean would bias strongly in favor of the new solver, which is especially faster on harder instances. Table 3 provides the results for a time limit of two hours. The second column shows the number of instances in the test-set. Column three shows the mean time taken by the solver of [30, 44], column four shows the mean time of the new solver. The next column gives the relative speedup of the new solver. The next three columns provide the same information for the maximum run-time, the last two columns give the number of solved instances.

Table 3: Computational comparison of the solver developed for this article and the solver described in [30, 44]

		mean time (sh. geo. mean)			maximum time			# solved	
Test set	#	P.&V.D. [s]	new [s]	speedup	P.&V.D. [s]	new [s]	speedup	P.&V.D.	new
E	20	0.3	0.3	1.0	**3.4**	4.8	0.7	20	20
2R	27	5.0	**3.0**	**1.7**	43.9	**12.2**	**3.6**	27	27
ALUE	15	1.4	3.2	0.4	**14.8**	35.3	0.4	15	15
vienna-s	85	7.8	**3.9**	**2.0**	623.5	**74.0**	**8.4**	85	85
ES10000FST	1	138.0	**122.4**	**1.1**	138.0	**122.4**	**1.1**	1	1
SP	8	81.0	**19.7**	**4.1**	>7200	**598.4**	**>12.0**	6	**8**
GEO-adv	23	158.7	**62.7**	**2.5**	6476.5	**926.6**	**7.0**	23	23
Cophag14	21	27.7	**12.6**	**2.2**	>7200	**3300.2**	**>2.2**	20	**21**
WRP3	63	22.8	**18.8**	**1.2**	6073.2	**3646.2**	**1.7**	63	63
PUC	50	2458.1	**1910.2**	**1.3**	>7200	>7200	1.0	12	**14**

Overall, the new solver performs stronger on eight of the ten test-sets. Often by a significant margin, such as for *GEO-adv* or *SP*. Only on *ALUE* the new solver performs significantly worse. A possible reason is the existence of small node-separators on these instances. These separators are heavily exploited by partitioning methods in [30, 44]. However, our solver includes no algorithms yet to exploit such separators. It should be noted that the same behavior can be observed for the related test-set *ALUT* which is not included in Table 3. On the other hand, there are also several other benchmark sets for which the new solver is faster, but which are not contained in Table 3, since they are similar to already included ones (e.g. *GEO-org*, *1R*, *WRP3*, *LIN*).

For most of the test-sets in Table 3, the new solver is around an order of magnitude or more faster than the previous version of SCIP-Jack described in [38], both with respect to the mean and maximum time. Even if one merely considers the enhancements described in this article (as compared to methods already known in the literature), the speed-up is still huge, as shown in Table 4. By *base* we denote the version of the new solver, where all enhancements described in this article that were not known before are deactivated. For example, this version still includes extended reduction techniques, but only in the form already described in [33]. Note that this version still includes several new (auxiliary) algorithms and data-structures that were developed as part of this article, but are not explicitly described. These include, in particular, additional algorithms and data-structures used for the extended reduction techniques. By *full* we denote the complete version of the new solver, which is the same as the one used in Table 3.

It can be seen that the *full* version performs better on all test-sets. It is not only faster, but also solves more instances. The most notable case occurs for the *ALUE* test-set, where the *full* version can solve all instances in at most 35.3 seconds, whereas the *base* version leaves two instances unsolved after two hours. Overall, the performance improvement is rather remarkable if one considers the wide range of sophisticated methods already included in the *base* version.

Table 4: Computational comparison of two versions of the new solver.

		mean time (sh. geo. mean)			maximum time			# solved	
Test set	#	base [s]	full [s]	speedup	base [s]	full [s]	speedup	base	full
E	20	0.3	0.3	1.0	8.8	**4.8**	**1.8**	20	20
2R	27	6.9	**3.0**	**2.3**	26.1	**12.2**	**2.1**	27	27
ALUE	15	20.8	**3.2**	**6.5**	>7200	**35.3**	**>204.0**	13	**15**
vienna-s	85	18.3	**3.9**	**4.7**	134.6	**74.0**	**1.8**	85	85
ES10000FST	1	305.3	**122.4**	**2.5**	305.3	**122.4**	**2.5**	1	1
SP	8	22.2	**19.7**	**1.1**	602.8	**598.4**	**1.0**	8	8
GEO-adv	23	141.4	**62.7**	**2.3**	2588.5	**926.6**	**2.8**	23	23
Cophag14	21	26.8	**12.6**	**2.1**	>7200	**3300.2**	**>2.2**	20	**21**
WRP3	63	35.0	**18.8**	**1.8**	>7200	**3646.2**	**>2.0**	62	**63**
PUC	50	2102.2	**1910.2**	**1.1**	>7200	>7200	1.0	13	**14**

Finally, we provide results for several large-scale Euclidean Steiner tree problems. For solving such problems, the bottleneck is usually the full Steiner tree concatanation [22]. This concatanation can also be solved as an SPG, however [34]. In Table 5 we give results for Euclidean instances from [22] with 25 thousand (*EST-25k*) and 50 thousand (*EST-50k*) points in the plane. For *EST-25k* the mean and maximum times are between one and two orders of magnitude faster than those of the well-known geometric Steiner tree solver GeoSteiner 5.1 [22]. Moreover, 7 of the 15 instances from *EST-50k* are solved for the first time to optimality—in at most 390 seconds. On the other hand, GeoSteiner cannot solve these instances even after seven days of computation. Unfortunately, [36] does not report results for these instances. However, the solver by [29], which won the heuristic SPG category at the 11th DIMACS Challenge, does not reach the upper bounds from GeoSteiner on any of the *EST-25k* and *EST-50k* instances.

Table 5: Results for Euclidean Steiner tree instances.

Test set	#	mean time [s]	maximum time [s]	# solved
EST-25k	15	66.2	92.4	15
EST-50k	15	286.1	390.0	15

6 Conclusion and outlook

This article has described the combination of implication, conflict, and reduction concepts for the SPG, with the aim of improving the state of the art in exact SPG solution. This combination has spawned several new techniques that (provably) dominate well-known results from the literature, such as the bottleneck Steiner distance. The integration of the new methods into the branch-and-cut solver SCIP-Jack has shown a large impact on exact SPG solution. The new SCIP-Jack could even outperform the long-reigning state-of-the-art solver by [30, 44].

Still, there are several promising routes for further improvement. First, one could improve the newly introduced methods. For example, by using full-backtracking in the extended reduction methods, by improving the approximation of the implied bottleneck Steiner distance, or by adapting the latter for replacement techniques. Second, several powerful methods described in [30, 44] could be added to the new solver, e.g. a stronger IP formulation realized via price-and-cut, or additional reduction techniques via partitioning.

Unlike the solver by [30, 44], the new SCIP-Jack will be made freely available for academic use—as part of the next SCIP release.

7 Acknowledgements

The work for this article has been conducted within the Research Campus Modal funded by the German Federal Ministry of Education and Research (fund number 05M14ZAM), and has also been supported by the DFG Cluster of Excellence MATH+.

References

[1] Tobias Achterberg. Conflict analysis in mixed integer programming. *Discrete Optimization*, 4(1):4–20, 2007.

[2] Tobias Achterberg. *Constraint Integer Programming.* PhD thesis, Technische Universität Berlin, 2007.

[3] Édouard Bonnet and Florian Sikora. The PACE 2018 Parameterized Algorithms and Computational Experiments Challenge: The Third Iteration. In Christophe Paul and Michal Pilipczuk, editors, *13th International Symposium on Parameterized and Exact Computation (IPEC 2018)*, volume 115 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:15, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[4] Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. Steiner Tree Approximation via Iterative Randomized Rounding. *The Journal of the ACM*, 60(1):6, 2013.

[5] Xiuzhen Cheng and Ding-Zhu Du. *Steiner trees in industry*, volume 11. Springer Science & Business Media, 2004.

[6] Marcus Poggi de Aragao and Renato F. Werneck. On the Implementation of MST-based Heuristics for the Steiner Problem in Graphs. In *Proceedings of the 4th International Workshop on Algorithm Engineering and Experiments*, pages 1–15. Springer, 2002.

[7] DIMACS. 11th DIMACS Challenge. `http://dimacs11.zib.de/`, 2015. Accessed: January 10. 2020.

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

[9] C. Duin. *Steiner Problems in Graphs*. PhD thesis, University of Amsterdam, 1993.

[10] C. W. Duin and A. Volgenant. Reduction tests for the Steiner problem in graphs. *Networks*, 19(5):549–567, 1989.

[11] Cees Duin. *Preprocessing the Steiner Problem in Graphs*, pages 175–233. Springer US, Boston, MA, 2000.

[12] C.W. Duin and A. Volgenant. An edge elimination test for the Steiner problem in graphs. *Operations Research Letters*, 8(2):79 – 83, 1989.

[13] Matteo Fischetti, Markus Leitner, Ivana Ljubić, Martin Luipersbeck, Michele Monaci, Max Resch, Domenico Salvagnin, and Markus Sinnl. Thinning out Steiner trees: a node-based model for uniform edge costs. *Mathematical Programming Computation*, 9(2):203–229, Jun 2017.

[14] Gerald Gamrath, Thorsten Koch, Stephen Maher, Daniel Rehfeldt, and Yuji Shinano. SCIP-Jack - A solver for STP and variants with parallelization extensions. *Mathematical Programming Computation*, 9(2):231 – 296, 2017.

[15] Michel X. Goemans, Neil Olver, Thomas Rothvoß, and Rico Zenklusen. Matroids and Integrality Gaps for Hypergraphic Steiner Tree Relaxations. In *Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing*, STOC '12, page 11611176, New York, NY, USA, 2012. Association for Computing Machinery.

[16] Keld Helsgaun. OR-Library. `http://people.brunel.ac.uk/~mastjjb/jeb/info.html`. Accessed: Sep. 18, 2020.

[17] Stefan Hougardy, Jannik Silvanus, and Jens Vygen. Dijkstra meets Steiner: a fast exact goal-oriented Steiner tree algorithm. *Mathematical Programming Computation*, 9(2):135–202, 2017.

[18] Radek Hušek, Dušan Knop, and Tomáš Masařík. Approximation algorithms for Steiner tree based on star contractions: A unified view. *arXiv preprint arXiv:2002.03583*, 2020.

[19] F.K. Hwang, D.S. Richards, and P. Winter. *The Steiner Tree Problem*. Annals of Discrete Mathematics. Elsevier Science, 1992.

[20] IBM. Cplex.

[21] Yoichi Iwata and Takuto Shigemura. Separator-Based Pruned Dynamic Programming for Steiner Tree. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1520–1527, 2019.

[22] Daniel Juhl, David M Warme, Pawel Winter, and Martin Zachariasen. The GeoSteiner software package for computing Steiner trees in the plane: an updated computational study. *Mathematical Programming Computation*, 10(4):487–532, 2018.

[23] R. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[24] Sándor Kisfaludi-Bak, Jesper Nederlof, and Erik Jan van Leeuwen. Nearly ETH-tight algorithms for planar Steiner tree with terminals on few faces. *ACM Transactions on Algorithms (TALG)*, 16(3):1–30, 2020.

[25] Thorsten Koch and Alexander Martin. Solving Steiner tree problems in graphs to optimality. *Networks*, 32:207–232, 1998.

[26] Thorsten Koch, Alexander Martin, and Stefan Voß. SteinLib: An updated library on Steiner tree problems in graphs. In D.-Z. Du and X. Cheng, editors, *Steiner Trees in Industries*, pages 285–325. Kluwer, 2001.

[27] M. Leitner, I. Ljubic, M. Luipersbeck, M. Prossegger, and M. Resch. New Real-world Instances for the Steiner Tree Problem in Graphs. Technical report, ISOR, Uni Wien, 2014.

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

[29] Thomas Pajor, Eduardo Uchoa, and Renato F. Werneck. A robust and scalable algorithm for the Steiner problem in graphs. *Mathematical Programming Computation*, Sep 2017.

[30] Tobias Polzin. *Algorithms for the Steiner problem in networks*. PhD thesis, Saarland University, 2003.

[31] Tobias Polzin and Siavash Vahdati Daneshmand. A comparison of Steiner tree relaxations. *Discrete Applied Mathematics*, 112(1-3):241–261, 2001.

[32] Tobias Polzin and Siavash Vahdati Daneshmand. Improved Algorithms for the Steiner Problem in Networks. *Discrete Applied Mathematics*, 112(1-3):263–300, September 2001.

[33] Tobias Polzin and Siavash Vahdati Daneshmand. *Extending Reduction Techniques for the Steiner Tree Problem*, pages 795–807. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[34] Tobias Polzin and Siavash Vahdati Daneshmand. On Steiner trees and minimum spanning trees in hypergraphs. *Operations Research Letters*, 31(1):12 – 20, 2003.

[35] Tobias Polzin and Siavash Vahdati Daneshmand. Practical Partitioning-Based Methods for the Steiner Problem. In Carme Àlvarez and María Serna, editors, *Experimental Algorithms*, pages 241–252, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[36] Tobias Polzin and Siavash Vahdati-Daneshmand. The Steiner Tree Challenge: An updated Study. Unpublished manuscript at http://dimacs11.cs.princeton.edu/downloads.html, 2014.

[37] Daniel Rehfeldt and Thorsten Koch. On the exact solution of prize-collecting Steiner tree problems. Technical Report 20-11, ZIB, Takustr. 7, 14195 Berlin, 2020.

[38] Daniel Rehfeldt, Yuji Shinano, and Thorsten Koch. SCIP-Jack: An exact high performance solver for Steiner tree problems in graphs and related problems. In *Modeling, Simulation and Optimization of Complex Processes HPSC 2018*, Proceedings of the 7th International Conference on High Performance Scientific Computing, 2019. accepted for publication.

[39] I. Rosseti, M.P. de Aragão, C.C. Ribeiro, E. Uchoa, and R.F. Werneck. New benchmark instances for the Steiner problem in graphs. In *Extended Abstracts of the 4th Metaheuristics International Conference (MIC'2001)*, pages 557–561, Porto, 2001.

[40] M. W. P. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing*, 6(4):445–454, 1994.

[41] H. Takahashi and Mastsuyama A. An approximate solution for the Steiner problem in graphs. *Mathematica Japonicae*, 24:573 – 577, 1980.

[42] Eduardo Uchoa. Reduction tests for the prize-collecting Steiner problem. *Operations Research Letters*, 34(4):437 – 444, 2006.

[43] Eduardo Uchoa, Marcus Poggi de Arago, and Celso C. Ribeiro. Preprocessing Steiner problems from VLSI layout. *Networks*, 40(1):38–50, 2002.

[44] Siavash Vahdati Daneshmand. *Algorithmic approaches to the Steiner problem in networks*. PhD thesis, Universität Mannheim, 2004.

[45] Jens Vygen. Faster algorithm for optimum Steiner trees. *Information Processing Letters*, 111(21):1075 – 1079, 2011.

[46] Pawel Winter. Reductions for the rectilinear Steiner tree problem. *Networks*, 26(4):187–198, 1995.

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

[48] Martin Zachariasen and Andre Rohe. Rectilinear group steiner trees and applications in vlsi design. *Mathematical Programming*, 94(2-3):407–433, 2003.