Mathematical Programming (2023) 197:903–966
https://doi.org/10.1007/s10107-021-01757-5

FULL LENGTH PAPER

Series B

Implications, conflicts, and reductions for Steiner trees

Daniel Rehfeldt[1] · Thorsten Koch[1,2]

Received: 26 May 2021 / Accepted: 3 December 2021 / Published online: 30 December 2021

Abstract
The Steiner tree problem in graphs (SPG) is one of the most studied problems in combinatorial optimization. In the past 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 advance exact SPG solution once again. 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 that are used in exact SPG algorithms. Finally, by integrating the new methods into a branch-and-cut framework, we obtain an exact SPG solver that is not only competitive with, but even outperforms the current state of the art on an extensive collection of benchmark sets. Furthermore, we can solve several instances for the first time to optimality.

Keywords Steiner tree problem · Optimal solution · Reduction techniques

Mathematics Subject Classification 90C27 · 90C10 · 90C35

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 $S \subseteq G$

✉ Daniel Rehfeldt
rehfeldt@zib.de

Thorsten Koch
koch@zib.de

1 Applied Algorithmic Intelligence Methods Departement, Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany

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

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. [6,27].

The SPG has seen numerous theoretical advances in the last 10 years, bringing forth significant improvements in complexity and approximability. See e.g. [5,15] for approximation, and [24,29,47] 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 Daneshmand almost 20 years ago [31,46]. They 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 [32–36]. 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 [13,14,17,30]. 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 [26,41]. Several of these instances could be solved for the first time to optimality. However, on the vast majority of instances from the literature, Polzin and Vahdati Daneshmand [31, 46] (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 [37]. In 2018, the 3rd PACE Challenge [4] 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 [31,46] 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 integer programming (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 [31,46] 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 $N(v) := \{w \in W \mid \{v, w\} \in \delta(v)\}$. Note that $v \notin N(v)$.

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 network* 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 $D_G(U, d)$. Note that we write usually $\tilde{d}(v, w)$ instead of $\tilde{d}(\{v, w\})$. For an SPG instance on a graph $G = (V, E)$ with terminal set $T \subseteq V$ and edge costs c we write (G, T, c) or (V, E, T, c).

2 From implications to reductions

Reduction techniques have been a key ingredient in exact SPG solvers, see e.g. [9,25, 33,45]. Among these techniques, the *bottleneck Steiner distance* introduced in [11] 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* [11] of P is

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

Let $v, w \in V$. Let $\mathscr{P}(v, w)$ be the set of all simple paths between v and w. The *bottleneck distance* [11] between v and w is defined as

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

with the common convention that $\inf \emptyset = \infty$. It holds that $b(v, w) = \infty$ if and only if v and w are unconnected[1]. Note that $b(v, w)$ is equal to the bottleneck length of the path between v and w on any minimum spanning tree (MST) of (G, c), as observed in [8].

Now consider the distance network $D := D_G(T \cup \{v, w\})$. Let b_D be the bottleneck distance in D. Define the *bottleneck Steiner distance* or *special distance* [11] between v and w as

$$s(v, w) := b_D(v, w).$$

The arguably best known bottleneck Steiner distance reduction method is based on the following criterion, which allows for edge deletion [11].

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

Note the analogy between bottleneck distance applied to the MST problem, and bottleneck Steiner distance applied to the SPG: Any edge $e = \{v, w\}$ that satisfies $b(v, w) < c(e)$ cannot be part of an MST. Otherwise, e could be replaced by an edge of cost at most $b(v, w)$ to obtain a spanning tree of smaller cost. Any edge $e = \{v, w\}$ that satisfies $s(v, w) < c(e)$ cannot be part of a minimum Steiner tree. Otherwise, e could be replaced by a path in G corresponding to an edge in $D = D_G(T \cup \{v, w\})$ with cost at most $b_D(v, w)$. In this case, one would obtain a Steiner tree of smaller cost. We also point out that bottleneck Steiner distances can be computed in polynomial time, but in practice (heuristic) approximations are used. See [33] 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* $\overline{b}(e)$ [33] as the bottleneck distance between v and w on $(V, E \setminus \{e\}, c)$.

[1] While we always assume the graph underlying an SPG instance to be connected, we will use auxiliary graphs that can be unconnected in the following.

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

$$p^+(v, F) := \max\left\{0, \sup\{\overline{b}(e) - c(e) \mid e \in F, e \cap T \neq \emptyset\}\right\}. \tag{2}$$

We call $p^+(v, F)$ the *F-implied profit* of v. 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 $W = (v_1, e_1, v_2, e_2, \dots, e_{r-1}, v_r)$ 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 $W(k, l) := (v_k, e_k, v_{k+1}, e_{k+1}, \dots, e_{l-1}, v_l)$. 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 $\mathscr{W}_T(v, w)$. With a slight abuse of notation we define $\delta_W(u) := \delta(u) \cap E(W)$ for any walk W and any $u \in V$. First, for a Steiner walk $W \in \mathscr{W}_T(v, w)$ define

$$P_W^+ := \{u \in V(W) \mid p^+\left(u, \delta(u) \backslash \delta_W(u)\right) > 0\} \cup \{v, w\}.$$

Define the *implied Steiner cost* of W as

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

Define the *implied Steiner length* of W as

$$l_p^+(W) := \max\{c_p^+(W(v_k, v_\ell)) \mid 1 \leq k \leq \ell \leq r,\ v_k, v_\ell \in P_W^+\}. \tag{3}$$

To understand the usage of the implied Steiner length, consider the SPG instance segment shown in Fig. 1. Assume that edge $\{v_1, v_4\}$ is part of Steiner tree S. Removing this edge from S results in two trees S' and S'' with $v_1 \in V(S')$, $v_4 \in V(S'')$. Consider the Steiner walk $W := (v_1, \{v_1, v_2\}, v_2, \{v_2, v_3\}, v_3, \{v_2, v_3\}, v_2, \{v_2, v_4\}, v_4)$. Note that $p^+\left(v_3, \delta(v_3) \backslash \delta_W(v_3)\right) = 3$, and thus $l_p^+(W) = 4$. We claim that S' and S'' can be reconnected to a Steiner tree $\tilde{S}$ that is of smaller weight than S by using only edges from W. First, assume v_3 is contained in either S' or S''. In this case, we can use the edges $\{v_1, v_2\}, \{v_2, v_3\}$ (if $v_3 \in V(S'')$) or the edges $\{v_2, v_3\}, \{v_2, v_4\}$ (if $v_3 \in V(S')$) to reconnect S' and S''. Second, assume that v_3 is neither contained in S' nor in S''. In this case, also the edge $\{v_3, t_1\}$ cannot be contained in S' or S'': Because S is a Steiner tree, we have $t_1 \in V(S'')$. Indeed, also $\{t_1, v_4\} \in E(S'')$ holds. Reconnect S' and S'' by adding all edges of W that are neither in S' nor in S''. This procedure results in a Steiner tree $\tilde{S}$. Next, add edge $\{v_3, t_1\}$ and remove edge $\{t_1, v_4\}$ from $\tilde{S}$. This exchange reduces the weight of $\tilde{S}$ by $p^+\left(v_3, \delta(v_3) \backslash \delta_W(v_3)\right)$. Thus, the final Steiner tree $\tilde{S}$ satisfies $c(E(\tilde{S})) \leq c(E(S)) - 1$.

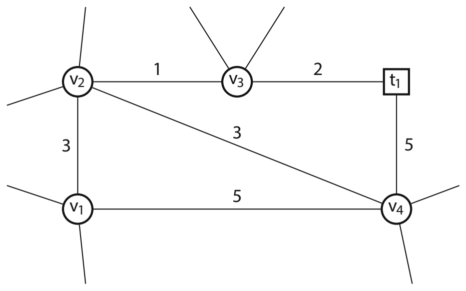

Fig. 1 Segment of an SPG instance. Terminals are drawn as squares

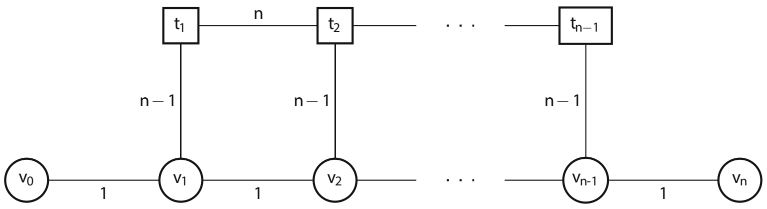

Fig. 2 SPG instance with $\frac{s(v_0, v_n)}{s_p(v_0, v_n)} = n$. Terminals are drawn as squares

With the above discussion in mind, define the *implied Steiner distance* between v and w as

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

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

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

Note that $s_p(v, w) \le s(v, w)$ and that the inequality can be strict. Indeed, $\frac{s(v,w)}{s_p(v,w)}$ can become arbitrarily large, as Fig. 2 shows: It holds that $s(v_0, v_n) = n$, but $s_p(v_0, v_n) = 1$. To see the latter, consider the Steiner walk $W = (v_0, \{v_0, v_1\}, v_1, \ldots, \{v_{n-1}, v_n\}, v_n)$. Each vertex $v_1, \ldots, v_{n-1}$ has an implied profit of 1 in W. Thus, $l_p^+(W) = 1$. Because 1 is the minimum edge cost on any (v_0, v_n)-walk and $s_p(v_0, v_n) \le l_p^+(W)$ by definition, we also have $s_p(v_0, v_n) = 1$.

The above discussion implies that 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 $c(E(S')) \le c(E(S))$, 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). In the following, we will use a Steiner walk to reconnect $\tilde{S}$. First, we show the existence of such a reconnecting Steiner walk that has an implied Steiner length (3) smaller than $c(e)$. Second, we add the edges of this walk to $\tilde{S}$, obtaining a Steiner tree. Third, we follow the same underlying idea as in the discussion for Fig. 1 and apply edge-exchange operations for each vertex of positive implied profit on the Steiner walk. In this way, the weight of $\tilde{S}$ is reduced.

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

$$d_p^+(t, u) \leq s_p(v, w). \tag{4}$$

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

$$l_p^+(W) = d_p^+(t, u). \tag{5}$$

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

$$\sum_{e \in E(W(a,b))} c(e) - \sum_{v \in V(W(a,b)) \setminus \{v_x, v_y\}} p^+ \left(v, \delta(v) \setminus \delta_{W(x,y)}\right) \leq c_p^+ \left(W(x, y)\right). \tag{6}$$

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).[2] It holds that

$$\sum_{e \in E(S_0')} c(e) \leq \sum_{e \in E(S)} c(e) + \sum_{e \in E(W(a,b))} c(e) - c(\{v, w\}). \tag{7}$$

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

$$\overline{b}(e_i^+) - c(e_i^+) = p^+(v_i^+, \delta(v_i^+) \setminus \delta_{W(x,y)}). \tag{8}$$

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

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

We will construct Steiner trees S'_i for $i \in \{1, \dots, z\}$ that satisfy

$$\sum_{e \in E(S'_i)} c(e) \leq \sum_{e \in E(S'_0)} c(e) - \sum_{k=1}^{i} p^+(v_k^+, \delta(v) \setminus \delta_{W(x,y)}), \tag{9}$$

as well as

$$\bigcup_{k=i+1}^{z} \{e_k^+\} \cap E(S'_i) = \emptyset, \tag{10}$$

and

$$V(S'_i) = V(S'_0). \tag{11}$$

One readily verifies that S'_0 satisfies (9)–(11). Let $i \in \{1, \dots, z\}$ and assume that (9)–(11) 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 $\tilde{e}_i \in E(P_i)$ 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 (9)–(11). Equality (10) follows from the fact that all e_i^+ are disjoint. And (11) follows from the construction of S'_i. For (9), observe that by definition of the bottleneck distance it holds that $c(\tilde{e}_i) \geq \overline{b}(e_i^+)$ and therefore

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

Thus, Eq. (8) implies that S'_i satisfies (9).

Finally, set $S' := S'_z$. Because of (11) it holds that $T \subseteq V(S')$. Furthermore, one obtains:

$$\sum_{e \in E(S')} c(e) \overset{(9)}{\leq} \sum_{e \in E(S'_0)} c(e) - \sum_{k=1}^{z} p^+(v_k^+, \delta(v_k^+) \setminus \delta_{W(x,y)}) \tag{12}$$

$$\overset{(7)}{\leq} \sum_{e \in E(S)} c(e) + \sum_{e \in E(W(a,b))} c(e) - c(\{v, w\})$$

$$- \sum_{k=1}^{z} p^+(v_k^+, \delta(v_k^+) \setminus \delta_{W(x,y)}) \tag{13}$$

$$\overset{(6)}{\leq} \sum_{e \in E(S)} c(e) - c(\{v, w\}) + c_p^+(W(x, y)) \tag{14}$$

$$\overset{(5)}{\leq} \sum_{e \in E(S)} c(e) - c(\{v, w\}) + l_p^+(W) \tag{15}$$

$$\overset{(4)}{\leq} \sum_{e \in E(S)} c(e) - c(\{v, w\}) + s_p(v, w) \tag{16}$$

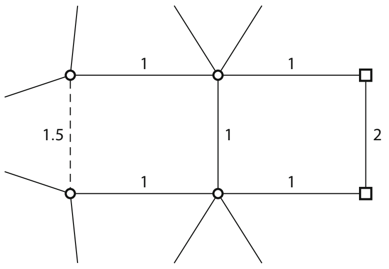

Fig. 3 Segment of a Steiner tree instance. Terminals are drawn as squares. The dashed edge can be deleted by employing Theorem 2

$$\leq \sum_{e \in E(S)} c(e), \tag{17}$$

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 1 *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 3 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 1 *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 [44]. 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 [39] for a definition that dominates the original one from [44]. 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 bottleneck Steiner distance

This section 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 $p_{vw} := \max \left\{ p^+(v, \{e\}) \mid e \in \delta(v) : w, pred[v] \notin e \right\}$. If

$$\tilde{d}[v] + c(\{v, w\}) - \min \left\{ c(\{v, w\}), p_{vw}, \tilde{d}[v] \right\} < \tilde{d}[w], \tag{18}$$

then set $\tilde{d}[w]$ to the left hand side of (18) and add w to Q. Further, set $pred[w] := v$. If (18) holds and $w \in N(v_0)$, then we can delete edge $\{v, w\}$.

Note that on the left hand side of (18) 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 (18), 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 (18) is satisfied with equality if $w \in N(v_0)$. In practice, one should bound the maximum number of visited edges (in the implementation for this article we simply use a fixed bound). 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. [19,33], 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 [12].

Proposition 2 *Let* $\{v, w\} \in E$ *and* $t_i, t_j \in T$, $t_i \neq t_j$ *such that: If*

$$s_p(v, t_i) + c(\{v, w\}) + s_p(w, t_j) \leq b_{\{v,w\}}(t_i, t_j), \tag{19}$$

then there is a minimum Steiner tree 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 [12]. To avoid an abundance of technicalities, we therefore only provide a proof sketch. For a detailed proof see the technical report [38].

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

$$c(E(S_i)) + c(E(S_j)) + b_{\{v,w\}}(t_i, t_j) \le c(E(S)). \tag{20}$$

Now the sketchy part starts: Similar to the proof of Theorem 2, condition (19) allows us to connect S_i to v such that the resulting tree $\tilde{S}_i$ satisfies

$$c(E(\tilde{S}_i)) \le c(E(S_i)) + s_p(v, t_i). \tag{21}$$

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

$$c(E(\tilde{S}_j)) \le c(E(S_j)) + s_p(w, t_j). \tag{22}$$

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 (21) and (22), respectively, have no vertex in common. If they had a vertex in common, one could build a new Steiner walk W_0 with $l_p^+(W_0) \le s_p(v, t_i) + s_p(w, t_j)$ 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 $c(E(\tilde{S}))$ by deleting an edge from each cycle. From (20), (21), and (22) it follows that

$$c(E(\tilde{S})) \le c(E(S)),$$

which concludes the proof. □

If criterion (19) is satisfied, one can contract edge $\{v, w\}$ and make the resulting vertex a terminal. The original criterion from [12] uses the standard distance in (19) instead of the implied bottleneck Steiner distance. We note that using the (standard) bottleneck Steiner distance in (19) does not improve the original test. However, using the implied bottleneck Steiner distance leads to a strictly stronger criterion, as the example in Fig. 4 shows. Note that $b_{\{t_1,v_1\}}(t_1, t_3) = 2$ and $s_p(v_1, t_3) = 1$. Thus, (19) 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 3 *Let $\{v, w\} \in E$ and $t_i, t_j \in T$, $t_i \neq t_j$. If (19) 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 a 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

$$c(E(S_i)) + c(E(S_j)) + b_{\{v,w\}}(t_i, t_j) \le c(E(S)). \tag{23}$$

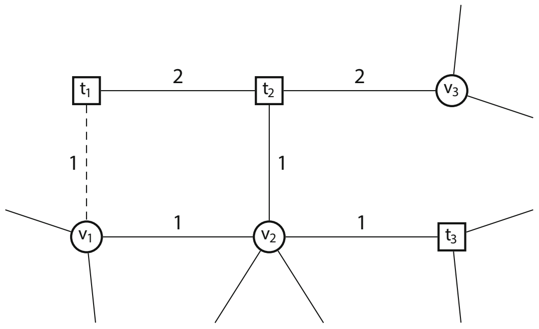

Fig. 4 Segment of a Steiner tree instance. Terminals are drawn as squares. The dashed edge can be contracted by employing Proposition 2

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 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 (19) and (23) it holds that

$$c(E(S')) \leq c(E(S)).$$

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. $\square$

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 4 *Let $D := D_G(T, d)$. Let Y be a minimum spanning tree in D. Write its edges $\{e_1^Y, e_2^Y, \ldots, e_{|T|-1}^Y\} := E(Y)$ 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:

$$\sum_{i=1}^{|\Delta|-1} d(e_i^Y) \leq \sum_{e \in \Delta} c(e), \tag{24}$$

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

The proposition follows from Corollary 3, which we will introduce in Sect. 4.2. If the conditions (24) are satisfied for a vertex $v \in V \setminus T$, one can *pseudo-eliminate* [12] or *replace* [31] 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 Fig. 5 exemplifies why Proposition 4 cannot be formulated by using the implied Steiner distance. The weight of the minimum spanning tree Y for

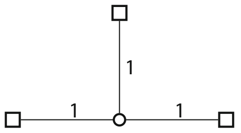

Fig. 5 SPG instance. Terminals are drawn as squares

$D_G(T, d)$ 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 [12] 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.

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. We say that a set $E' \subseteq E$ with $|E'| \geq 2$ is in *conflict* if no minimum Steiner tree contains more than one edge of E'.

3.1 Node replacement

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. For example, we only delete an edge if we can show 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 $i = 0, 1, \ldots, k$ by $\Pi^{(i)} : E^{(i)} \rightarrow \mathscr{P}(E)$ and $\Pi^{(i)}_{FIX} \subseteq E$. Initially, we set

- $\Pi^{(0)}(e) := \{e\}$ for all $e \in E$,
- $\Pi^{(0)}_{FIX} = \emptyset$.

Consider a reduced instance $I^{(i)}$. If we contract an edge $e \in E^{(i)}$, we set $\Pi^{(i+1)}_{FIX} := \Pi^{(i)}_{FIX} \cup \Pi^{(i)}(e)$. For any other operation we set $\Pi^{(i+1)}_{FIX} := \Pi^{(i)}_{FIX}$. If we replace a vertex $v \in V^{(i)}$, then

- for each newly inserted edge $\{u, w\} \subset N(v)$ we set: $\Pi^{(i+1)}(\{u, w\}) := \Pi^{(i)}(\{v, u\}) \cup \Pi^{(i)}(\{v, w\})$,
- 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

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

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

[34] 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 Sect. 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 [34]. Initially, we define additional ancestor information for each $i = 0, 1, \ldots, k$. Namely, sets of *replacement ancestors* $\Lambda^{(i)} : E^{(i)} \to \mathscr{P}(\mathbb{N})$, and $\Lambda^{(i)}_{FIX} \in \mathscr{P}(\mathbb{N})$. 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 $\{u, w\} \subset N(v)$, 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 5 *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 2 *Let I, $I^{(k)}$ as in Proposition 5, 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 2 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 6 *Let $e = \{v, w\} \in E$ with $e \cap T = \emptyset$. Define*

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

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

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

If for all $\Delta \in \mathscr{D}$ 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 3, which will be introduced in Sect. 4. If the condition of Proposition 6 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 operation 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 5 remains valid if path replacement is added to the list of valid reduction operations.

Figure 6 illustrates an application of Proposition 6. 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.

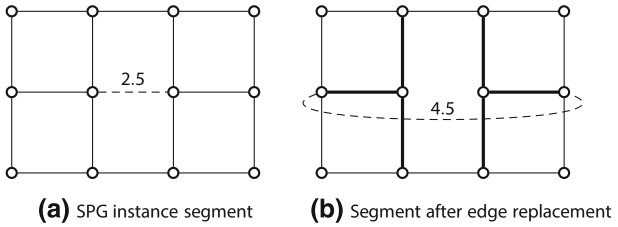

Fig. 6 Segment of a Steiner tree instance (showing only non-terminals). All edges except for the dashed ones have unit weight. The dashed edge in (**a**) has been replaced in (**b**). All edges that are in conflict with the replacement edge in (**b**) are drawn in bold

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 loosely related to the concepts of probing and conflict (graph) analysis for mixed-integer programming (MIP), see e.g. [1,42].

The idea of extension was first introduced in [48] for the rectilinear Steiner tree problem. Later the idea was adopted by [45] for the SPG. The next advancement came in [10], where backtracking was used, together with a number of new reduction criteria for the enumerated trees. Finally, [34] introduced the up-to-now strongest extended reduction techniques, which improved and complemented the previous results. The authors showed that their sophisticated 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 [34].

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 [34]. 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 7 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.

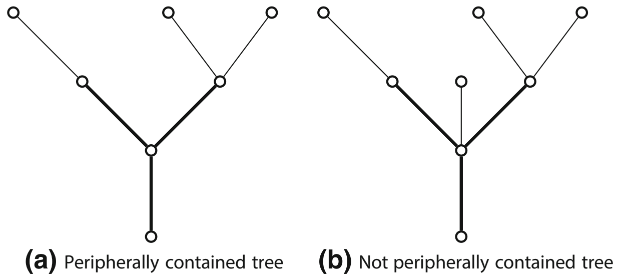
(a) Peripherally contained tree
(b) Not peripherally contained tree

Fig. 7 Illustration of peripherally inclusion. The bold subtree is peripherally contained in the entire tree in (**a**), but not in (**b**)

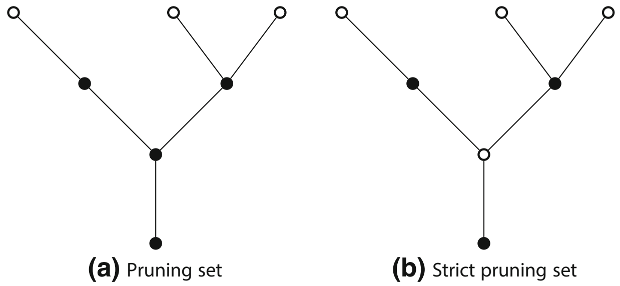

Fig. 8 Illustration of pruning and strict pruning sets. The filled vertices in **a** form a (non-strict) pruning set, whereas the filled vertices in **b** constitute a strict pruning set

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 $v, w \in P$ 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 8 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 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 [34], but more general.[3] 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 $false$.
- 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). We explain the extended reduction framework via an example at the end of Sect. 4.2.

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 5–14. 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 3.

[3] We note, however, that the framework presented in [34] is (slightly) erroneous.

Algorithm 1: Extended- RuledOut

```
Data: SPG instance $I = (G, T, c)$, tree $Y$ with $Y \cap T \subseteq L(Y)$
Result: $true$ if $Y$ is shown to not be peripherally contained in any minimum Steiner tree; $false$
        otherwise
1  foreach $P \in$ 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 $v \in L(Y)$ do
6      if $v \in T$ or not Promising($I, Y, v$) then continue $success := true$
7      foreach $E' \in$ ExtensionSets($I, Y, v$) do
8          if not Extended- RuledOut($I, Y + E'$) then
9              $success := false$
10             break
11         end
12     end
13     if $success$ then return $true$
14 end
15 return $false$
```

Algorithm 2: ExtensionSets

```
Data: SPG instance $I = (G, T, c)$, tree $Y$, vertex $v \in V(Y)$
Result: Set $\Gamma \subseteq \mathscr{P}(\delta(v))$ such that for all non-empty $\gamma \in \mathscr{P}(\delta(v))\setminus\Gamma$, the tree $Y + \rho$ is not
        peripherally contained in any minimum Steiner tree.
1  $Q := \emptyset$
2  $R := \emptyset$
3  foreach $e := \{v, w\} \in \delta(v)\setminus E(Y)$ do
4      if RuledOut($I, Y + \{e\}, L(Y) \cup \{w\}$) then
5          continue
6      end
7      if RuledOutStrict($I, Y + \{e\}, L(Y) \cup \{w\}$) then
8          $R := R \cup \{e\}$
9          continue
10     end
11     $Q := Q \cup \{e\}$
12 end
13 return $(\mathscr{P}(Q)\setminus\emptyset) \cup R$
```

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.

We just note here that we have only implemented extensions in a depth-first-search manner: 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 [34]. 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 [31,34], 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 $v \in V(Y)$ that satisfy the following: For any $q \in P \backslash \{p\}$ 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

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

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 [19,31] for similar, but weaker, conditions.

Theorem 3 *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 $I_{Y,P}$. The underlying idea of the proof is as follows: First, we remove Y_P from S. Next, we interconnect all vertices in P. Because of the assumptions of the theorem, this procedure also reconnects S. To obtain a tree that is of smaller weight than S, we use only edges for the reconnection that correspond to edges of $S_{Y,P}$.

Let $\tilde{S} \subset G$ be the forest defined as follows:

$$V(\tilde{S}) := (V(S) \backslash V(Y_P)) \cup V(S_{Y,P}), \tag{25}$$

$$E(\tilde{S}) := E(S) \backslash E(Y_P). \tag{26}$$

Let $\tilde{\mathscr{C}}$ be the set of connected components of $\tilde{S}$. Further, let $f : V \to \tilde{\mathscr{C}} \cup \{\emptyset\}$ such that $f(v) = \tilde{C}$ if $v \in V(\tilde{C})$ for a $\tilde{C} \in \tilde{\mathscr{C}}$, and $f(v) = \emptyset$ otherwise. Note that each $\tilde{C} \in \tilde{\mathscr{C}}$ 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 $\tilde{C} \in \tilde{\mathscr{C}}$ with $\overline{Y}_p \subseteq \tilde{C}$. In the following, we will iteratively connect all the components in $\tilde{\mathscr{C}}$.

While $|\tilde{\mathscr{C}}| > 1$ 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 $u \in V(Q)\backslash\{q, r\}$. 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 $\tilde{\mathscr{C}}$ 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 $|\tilde{\mathscr{C}}| = 1$, 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

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

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 $E(Y)\backslash E(Y_P)$. Note that one can also extend Theorem 3 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 3 *Let Y, P as in Theorem 3. Let (P', P'') be a partition of P. Let F' be an MST on $D_{G_{Y,P}}\big(P', s_{Y,P}\big)$, and let z' be the weight of F'. Let F'' be an MST on $D_{G_{Y,P}}\big(T_{Y,P}, s_{Y,P}\big)$. Write $\{e_1^{F''}, e_2^{F''}, \ldots, e_{|T_{Y,P}|-1}^{F''}\} := E_{Y,P}(F'')$ such that $s_{Y,P}(e_i^{F''}) \geq s_{Y,P}(e_j^{F''})$ for $i < j$. Define*

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

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 3, 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 3. Further, proceed as in the proof of Theorem 3 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 3, 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

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

which contradicts the optimality of S. □

As for Theorem 3, the contractions in Corollary 3 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 3 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 7 *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).

Based on the SPG instance in Fig. 9, we demonstrate the usage of the extended reduction framework and the above reduction criteria in the following. We aim to replace (or pseudo-eliminate) vertex v_3. To show that this operation is valid, we prove that the tree Y with $V(Y) = \{v_3\} \cup N(v_3)$, $E(Y) = \delta(v_3)$ is not peripherally contained in any minimum Steiner tree. We call Algorithm 1 with Y as defined above. We are neither able to *rule out* Y in Line 2, nor do we *truncate* the search in Line 4. In Line 5, we consider vertex v_5 and mark it as *promising*. The extension sets obtained from Algorithm 2 are: $\{\{t_2, v_5\}\}$, $\{\{v_4, v_5\}\}$, and $\{\{t_2, v_5\}, \{v_4, v_5\}\}$. We (recursively) call Algorithm 1 for each of these three extensions in Line 8.

First, we consider the extension via the edge $\{t_2, v_5\}$. The tree $Y' := Y + \{\{t_2, v_5\}\}$ with pruning set $P = \{t_1, t_2, v_2\}$ can be shown to not be P-peripherally contained in a minimum Steiner tree by using Proposition 7: It holds $s_p(t_1, t_2) = 2 < 2.5 = b_{Y',P}(t_1, t_2)$, where the pruned tree bottleneck corresponds to the edges $\{v_3, v_5\}$ and $\{t_2, v_5\}$.

Next, we consider the extension via the edge $\{v_4, v_5\}$. We are not able to rule out this extension, and thus extend the tree $Y' := Y + \{\{v_4, v_5\}\}$ from vertex v_4. The extension

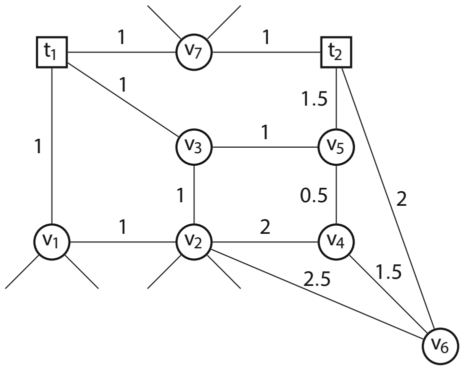

Fig. 9 Segment of a Steiner tree instance. Terminals are drawn as squares. By using the extended reduction framework, one can show that vertex v_3 can be replaced

set obtained from Algorithm 2 is just $\{\{v_4, v_6\}\}$, because any extension of Y' via the edge $\{v_2, v_4\}$ would results in a cycle and can thus be discarded. However, the tree $Y'' := Y' + \{\{v_4, v_6\}\}$ with pruning set $P = \{t_1, v_2, v_6\}$ can be ruled out by using Proposition 7: It holds that $s_p(t_1, v_6) = 2 < 3 = b_{Y'',P}(t_1, v_6)$, where the pruned tree bottleneck corresponds to the edges $\{v_3, v_5\}$, $\{v_4, v_5\}$, and $\{v_4, v_6\}$.

Finally, we consider the extension via the edge set $\{\{t_2, v_5\}, \{v_4, v_5\}\}$. We are not able to rule out this extension, and thus extend the tree $Y' := Y + \{\{t_2, v_5\}, \{v_4, v_5\}\}$ from vertex v_4. As before, the extension set obtained from Algorithm 2 is $\{\{v_4, v_6\}\}$. The tree $Y'' := Y' + \{\{v_4, v_6\}\}$ with pruning set $P = \{t_1, t_2, v_2, v_6\}$ can again be ruled out by using Corollary 3: It holds that $c(E(Y''_P)) = 6.5$, but the weight of an MST on $D_{G_{Y'',P}}(P, s_{Y'',P})$ is 6; the edges of the MST on $D_{G_{Y'',P}}(P, s_{Y'',P})$ are $\{t_1, t_2\}$, $\{t_1, v_2\}$, and $\{t_2, v_6\}$.

In summary, all extensions of the initial tree Y along vertex v_5 are ruled out in the first call of Algorithm 1. Thus, the algorithm returns *true*, which implies that vertex v_3 can be replaced.

Another criterion can be devised by using the reduced costs of the well-known bidirected cut formulation [50] 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 8 *Let Y be a tree. Let $P = \{p_1, \dots, p_k\}$ 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*

$$\tilde{L} + \min_{i \in \{1,\ldots,k\}} \max_{\{t_1,\ldots t_{i-1},t_{i+1},\ldots,t_{k'}\} \subseteq T \setminus V(Y_P)} \left\{ \tilde{d}(r, p_i) + \sum_{j \leq k', j \neq i} \tilde{d}(p_j, t_j) \right\}. \quad (27)$$

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 (27).

Finally, another important reduction criteria is constituted by edge conflicts—this result follows directly from Proposition 5.

Corollary 4 *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 [33], reduction techniques are the most important ingredient in a state-of-the-art SPG solver. While [33] uses linear programming and branch-and-bound mostly to trigger further reductions, we employ a proper branch-and-cut approach, based on [14]. On a high level, the solution process of SCIP- Jack can be naturally divided into three phases.

First, the presolving phase. Here, reduction techniques (combined with primal and dual heuristics) are employed to decrease the problem size. As can be seen in Sect. 5.2 and in the detailed results in the appendix, many instances are already drastically reduced in this phase.

Second, the linear programming (LP) based separation phase at the branch-and-bound root node. SCIP- Jack employs a specialized separation algorithm, see e.g. [14], to compute lower bounds based on the well-known bidirected cut formulation [50]. Additionally, several specialized methods such as primal heuristics and domain propagation are employed. For domain propagation, we also employ a modified version of the extended reduction techniques that makes use of the reduced costs from the LP-relaxation.

Third and finally, a branch-and-bound search is initiated, with the branching being done on the vertices of the graph. In this phase, again primal heuristics and domain propagation are employed. However, SCIP- Jack aims to avoid the branch-and-bound search, and puts much effort into the root node. Indeed, fewer than five percent of the instances used in this article require branching.

We enhance several vital components of this branch-and-cut framework. 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. However, in our implementation the reduction methods are employed far less aggressively in domain propagation than in presolving (so also the time spent in domain propagation is usually less than 10 percent of the time spent in presolving). The edge conflicts described in this article are used for generating clique cuts, which are well-known for general MIPs [2]. We note, however, that the impact of these cuts on the overall solution time is small; even for instances with many edge conflicts the obtained speed-up is usually only a few percent. 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 [43] 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 path 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 [3]. This section shows how to use the implication concept introduced in Sect. 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 $v \in V \setminus T$:

$$\tilde{p}(v) := \max\left\{0, \sup\left\{\overline{b}(e) - c(e) \mid e = \{v, w\} \in \delta(v), w \in T \setminus V(S)\right\}\right\}. \quad (28)$$

For all $v \in T$ set $\tilde{p}(v) := 0$. Essentially, (28) is a weaker version of the implied profit from Sect. 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 (28). For all $\{v, w\} \in \delta(v)$ proceed as follows. If

$$\tilde{d}[v] + c(\{v, w\}) - \min\left\{c(\{v, w\}), \tilde{p}(v), \tilde{d}[v]\right\} < \tilde{d}[w], \quad (29)$$

then set $\tilde{d}[w]$ to the left hand side of (29), and add w to Q. Further, set $pred[w] := v$.

Note that (29) 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. We use the value $\min_{e' \in \delta(w) \setminus \{e\}} c(e')$ instead of $\overline{b}(e)$ for $e = \{v, w\}$, $w \in T$ in (28) for two reasons: First, the value better represents the weight that can be saved when connecting w via v (because the bottleneck edge corresponding to $\overline{b}(e)$ might already be part of the tree computed by the heuristic so far). Second, this value is much faster to compute (and the primal heuristic is executed often as a subroutine within our implementation).

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 results of the solver by [31,46] published in [37]. 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 [37][4]. While the authors of the current article do not have access to the machine used in [37], 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, by multiplying the run-times of SCIP- JACK by 1.59. We use the same LP solver as [37]: CPLEX 12.6 [20]. All results were obtained single-threaded.

For the comparison with the solver by [31,46], we are restricted to the instances used in [37]. Still, the experiments in [37] include a large number of test-sets (both the STEINLIB and the 11th DIMACS Challenge collection). Thus, we only use test-sets with at least one instance that takes more than 10 s to be solved by [37] or our solver. There is one notable exception: We do not consider the test-sets *I320* and *I640* from the STEINLIB; for the following reason: [37] use specialized, non-default settings for several test-sets, including *I320* and *I640*, where they use only "(…) fast calculation of bounds (…)" during branch-and-bound. As we aim to give an unbiased picture of the performance of our solver, we only use our default settings for all instance sets. While we can achieve significant speed-ups on all tests-sets when using specialized settings, the impact is by far strongest on the *I* instances—more than an order of magnitude for the harder instances. We note, however, that we can match the results from [37] on I320 and I640 if we use dual-ascent bounds during branch-and-bound, instead of LP-based ones.

An overview of the test-sets is given in Table 1. The second column gives the number of instances per test-set. The third and fourth columns give the range of nodes

[4] Our machine obtains a score of 488.993589 (with the same compiler as [37]).

and edges per test-set. The fifth column states whether for all instances of the test-set optimal solutions are known.

Impact of implied profit reductions

In the following, 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 [50]. We use seven benchmark sets from the literature; three from the DIMACS Challenge, three from the SteinLib, and one from [22]. 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 report 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 other powerful reduction methods included in SCIP- Jack. Even if the percentage of remaining edges and nodes is already small on average for the base processing (such as for *VLSI*), there are for each of the seven test-sets at least a few instances that are still of large size. These instances can often be significantly reduced by the s_p techniques. While no run times are reported in the table, we note that on each of the seven test-sets the overall run time of the preprocessing (often significantly) decreases when the s_p based methods are used. Furthermore, even for other test-sets where the s_p methods are less (or not at all) successful, one does not observe an increase in the run time of the preprocessing above 10 percent.

Comparison with the state of the art

Next, we compare the solver by Polzin and Vahdati Daneshmand [31,46] and the new solver SCIP- Jack with respect to the mean time, the maximum time, and the number of solved instances. For the mean time we use the shifted geometric mean with a shift of 10. We note that the use of an arithmetic mean would bias strongly in favor of SCIP- Jack, which is especially faster on harder instances.

Table 3 provides the results for a time-limit of 24 h (divided by 1.59 in the case of SCIP- Jack), which is the same time-limit as used in the updated report [37]. The second column shows the number of instances in the test-set. Column three gives the number of instances solved by [37], column four the number of instances solved by SCIP- Jack. Column five shows the mean time taken by [37], column six shows the mean time of SCIP- Jack. The next column gives the relative speedup of SCIP- Jack. The last three columns provide the same information for the maximum run-time.

It can be seen that SCIP- Jack consistently outperforms [37]—both with respect to mean and maximum time. Also, SCIP- Jack solves on each test-set at least as many instances as [37]. The only test-set where [37] prevail is *VLSI*. On this test-set the results of the extended reductions reported in [31] are also stronger, which might be attributed to the use of full-backtracking, which has not yet been implemented in SCIP- Jack.

Table 1 Details on SPG benchmark sets

Name	#	$\|V\|$	$\|E\|$	Status	Description
2R	27	2000	11,600	Solved	3-D cross grid graphs from SteinLib
VLSI	116	90–36,711	135–68,117	Solved	Grid graphs with holes (non-geometric) from VLSI design [25]
Vienna-s	85	1991–89,596	3176–148,583	Solved	Instances derived from telecommunication network design, see [27]
Vienna-a	85	160–34,221	237–50,301	Solved	Presolved versions of the above network design instances
ES10000	1	27,019	39,407	Solved	Originally rectilinear Steiner tree instances. From SteinLib
TSPFST	76	89–17,127	104–27,352	Solved	Originally rectilinear Steiner tree instances. From TSPLIB [40]
GEO-org	23	42,481–235,686	52,552–366,093	Solved	Instances derived from telecommunication network design. From [27]
GEO-a	23	7565–71,184	11,521–113,616	Solved	Presolved versions of the above *GEO-org* instances
Cophag14	21	16–15,473	23–38,928	Solved	Originally obstacle-avoiding rectilinear instances. From 11th DIMACS Challenge

Table 1 continued

Name	#	$\|V\|$	$\|E\|$	Status	Description
WRP4	63	110–1898	188–3060	Solved	Instances derived from wire-routing processing problems [16]
WRP3	62	84–3168	149–6220	Solved	
LIN	37	53–38,418	80–71,657	Solved	Grid graphs with holes (non-geometric) from VLSI design. From SteinLib
SP	8	6–3997	9–10,278	Solved	Constructed hard instances; combination of odd-wheels and odd cycles. From SteinLib
PUC	50	64–4096	192–28,512	Unsolved	Constructed hard instances; hypercubes, and bipartite graphs [41]

Table 2 Average remaining nodes and edges after preprocessing

		Base preprocessing		$+s_p$ techniques		Relative change	
Test-set	#	Nodes [%]	Edges [%]	Nodes [%]	Edges [%]	Nodes [%]	Edges [%]
VLSI	116	0.4	0.4	0.1	0.1	**−75.0**	**−75.0**
Vienna-s	85	3.3	3.0	2.0	1.8	**−39.4**	**−40.0**
WRP4	63	36.2	36.0	33.5	33.0	**−7.5**	**−8.3**
Copenhag14	21	33.7	32.5	32.1	29.4	**−4.7**	**−9.5**
GEO-org	23	6.7	7.6	5.8	6.5	**−13.4**	**−14.5**
ES10000FST	1	24.1	27.1	15.1	16.8	**−37.3**	**−38.0**
ES-R50	15	17.5	22.8	12.6	16.6	**−28.0**	**−27.2**

Bold numbers signify a superior performance

On the other test-sets, the difference in the run-time is especially apparent for the maximum run time. This behavior can be explained by the fact that most test-sets contain many instances that can be solved very fast by both solvers—which brings the mean times closer together. Prominent examples are the *SP* and *Copenhag14* test-sets, for which all instances can be solved by SCIP- JACK within roughly 1 h, whereas [37] leave several instances unsolved even after 24 h.

As already mentioned, most test-sets in Table 3 contain a large number of instances that can be solved by both [37] and our solver in well below 1 s. To mitigate the impact of very easy instances on the average times, we group the instances according to their hardness in the following experiment. We use instance groups $[10^k, 86{,}400]$ for $k = -\infty, 0, 1, 2, 3$. Any group $[10^k, 86{,}400]$ contains each instance from Table 3 such that [37] or SCIP- JACK solves this instance in not less than 10^k, and at most 86,400 s. If an instance can be solved by only one solver within the time-limit, we consider the run-time of the other solver on this instance as 86,400 s. Such groupings are commonly used in computational mathematical optimization (also with the time lower bounds being powers of 10), see e.g. [28,49]. In addition to the shifted geometric mean, Table 4 also provides the arithmetic mean of the run-time for each group. As before, we give the results for both [37] and SCIP- JACK, and report the respective speed-up of SCIP- JACK.

Unsurprisingly, the ratio of the arithmetic mean stays largely unchanged with increasing hardness of the groups. SCIP- JACK is more than a factor of 5 faster than the solver from [37] on all groups. On the other hand, the performance difference with respect to the shifted geometric mean significantly increases with the hardness of the instances. For instances that take more than a 1000 s to be solved by [37] or SCIP- JACK, the latter is even by a factor of more than 7 faster.

Further results

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 [35]. In Table 5 we give results for Euclidean instances from [22] with 25 thousand (*EST-25k*), 50 thousand

Table 3 Computational comparison of the solver developed for this article (*S.-J.*) and the solver described in [31,46] (*P.&V.*)

		# solved		Mean time (sh. geo. mean)			Maximum time		
Test-set	#	P.&V.	S.-J.	P.&V. [s]	S.-J. [s]	Speedup	P.&V. [s]	S.-J. [s]	Speedup
VLSI	116	116	116	0.5	0.8	0.63	53.9	81.9	0.66
TSPFST	76	76	76	1.5	1.1	**1.36**	1161.4	326.9	**3.55**
WRP4	62	62	62	3.2	2.4	**1.33**	106.1	95.2	**1.11**
2R	27	27	27	5.0	2.6	**1.92**	43.9	12.1	**3.63**
Vienna-a	85	85⋆	85	7.2	5.0	**1.44**	441.3	57.1	**7.73**
Vienna-s	85	85⋆	85	7.8	5.9	**1.32**	623.5	57.7	**10.81**
WRP3	63	63	63	22.8	13.4	**1.70**	6073.2	4568.1	**1.33**
GEO-a	23	23⋆	23	158.7	55.8	**2.84**	6476.5	852.4	**7.60**
GEO-org	23	23⋆	23	145.6	59.4	**2.45**	4385.0	834.4	**5.26**
ES10000	1	1	1	138.0	80.9	**1.71**	138.0	80.9	**1.71**
Cophag14	21	20⋆	**21**	27.7	14.8	**1.87**	>86,400	4182.7	>**20.66**
SP	8	6	**8**	159.4	30.2	**5.28**	>86,400	1892.1	>**45.66**
LIN	37	35	**36**	31.3	15.3	**2.05**	>86,400	>86,400	1.00
PUC	50	17⋆	**18**	14, 964.9	11, 568.3	**1.29**	>86,400	>86,400	1.00

Bold numbers signify a superior performance. Times marked by a ⋆ were obtained by P.&V. with (specialized) non-default settings

Table 4 Computational comparison of the solver developed for this article (*S.-J.*) and the solver described in [31,46] (*P.&V.*), with instance groups ordered by hardness

		Shifted geometric mean time			Arithmetic mean time		
Group	#	P.&V. [s]	S.-J. [s]	Speedup	P.&V. [s]	S.-J. [s]	Speedup
[0, 86, 400]	644	12.2	7.9	**1.54**	1235.5	229.0	**5.40**
[1, 86, 400]	342	34.5	19.5	**1.77**	2326.4	431.2	**5.40**
[10, 86, 400]	178	125.4	52.6	**2.38**	4466.6	825.5	**5.41**
[100, 86, 400]	66	1403.2	295.0	**4.76**	11, 999.0	2197.6	**5.46**
[1000, 86, 400]	30	8035.8	1099.0	**7.31**	25, 923.1	4653.8	**5.57**

Bold numbers signify a superior performance

Table 5 Results of SCIP- Jack for Euclidean Steiner tree instances

Test set	#	# solved	Mean time [s]	Maximum time [s]
EST-25k	15	15	43.2	54.6
EST-50k	15	15	128.2	196.5
EST-100k	15	15	477.9	729.7

(*EST-50k*), and 100 thousand (*EST-100k*) 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 197 s. On the other hand, GeoSteiner cannot solve these instances even after seven days of computation. For *EST-100k*, GeoSteiner even leaves 12 of the 15 instances unsolved after one week of computation. In contrast, we solve all these instances to optimality in less than 13 minutes. Overall, we solve 19 instances for the first time to optimality.

Unfortunately, [37] does not report results for these instances. However, the solver by [30], 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*, *EST-50k*, and *EST-100k* instances.

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 [31,46].

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 [31,46] 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 [31,46], the new SCIP- JACK will be made freely available for academic use—as part of the SCIP Optimization Suite 8.

Acknowledgements The authors would like to thank the referees for their helpful comments. 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+.

Funding Open Access funding enabled and organized by Projekt DEAL.

A Detailed computational results

This appendix provides detailed computational results on the problem instances discussed in this article All following tables are structured as follows: First, the name of the respective instance is given. The next three columns give the number of vertices, arcs (after the graph transformation to bidirected SAP), and terminals of the instance. The subsequent segment, labelled "Presolved", provides the size of the preprocessed problem along with the preprocessing time.

The last segment provides first the dual and primal bound, or the optimal solution value if the problem could be solved to proven optimality. Moreover, the number of branch-and-bound nodes (N) and the total run time is given. A time-out is signified by a “>” in front of the termination time. We stress that the reported final execution times include both the preprocessing time and the reading time.

The time limit for the following instances is 54,340 s. This corresponds to 24 h on the machine used by [37] (Tables 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22).

Table 6 Detailed computational results for SPG, test-set 2R

Instance	Original			Presolved				Optimum	N	t [s]
	$\|V\|$	$\|A\|$	$\|T\|$	$\|V\|$	$\|A\|$	$\|T\|$	t [s]			
2r111	2000	11,600	9	0	0	0	0.1	**28,000**	1	0.1
2r112	2000	11,600	9	0	0	0	0.1	**32,000**	1	0.1
2r113	2000	11,600	9	0	0	0	0.1	**28,000**	1	0.1
2r121	2000	11,532	9	0	0	0	0.1	**28,000**	1	0.1
2r122	2000	11,544	9	0	0	0	0.1	**29,000**	1	0.1
2r123	2000	11,508	9	0	0	0	0.1	**25,000**	1	0.1
2r131	2000	11,452	9	0	0	0	0.1	**27,000**	1	0.1
2r132	2000	11,450	9	636	5426	9	0.4	**33,000**	1	0.4
2r133	2000	11,458	9	0	0	0	0.1	**29,000**	1	0.1
2r211	2000	11,600	50	571	4988	34	2.8	**89,000**	3	7.6
2r212	2000	11,600	49	132	818	17	0.9	**80,000**	1	1.1
2r213	2000	11,600	48	279	2104	29	2.0	**76,000**	1	2.4
2r221	2000	11,528	50	0	0	0	1.1	**83,000**	1	1.1
2r222	2000	11,530	50	0	0	0	1.9	**84,000**	1	1.9
2r223	2000	11,540	49	562	4606	40	1.8	**84,000**	1	4.6
2r231	2000	11,474	50	0	0	0	2.3	**86,000**	1	2.3
2r232	2000	11,466	49	453	3358	37	2.0	**87,000**	1	3.3
2r233	2000	11,460	47	0	0	0	1.3	**83,000**	1	1.3
2r311	2000	11,600	95	372	2586	47	1.2	**129,000**	1	2.0
2r312	2000	11,600	92	482	3802	45	1.0	**126,000**	1	2.2
2r313	2000	11,600	97	306	2124	38	1.0	**128,000**	1	1.3
2r321	2000	11,542	92	0	0	0	0.3	**125,000**	1	0.3
2r322	2000	11,506	92	397	2784	43	1.6	**130,000**	1	2.3
2r323	2000	11,528	96	651	4856	64	1.7	**142,000**	1	5.1
2r331	2000	11,472	93	260	1584	40	1.8	**134,000**	1	2.0
2r332	2000	11,490	95	544	3820	50	1.7	**136,000**	1	4.4
2r333	2000	11,482	98	449	2938	54	2.1	**143,000**	1	3.3

Bold numbers signify a superior performance

Table 7 Detailed computational results for SPG, test-set Copenhagen14

Instance	Original			Presolved				Optimum	N	t [s]
	$\|V\|$	$\|A\|$	$\|T\|$	$\|V\|$	$\|A\|$	$\|T\|$	t [s]			
ind1	18	62	10	0	0	0	0.0	**604**	1	0.0
ind2	31	114	10	0	0	0	0.0	**9500**	1	0.0
ind3	16	46	10	0	0	0	0.0	**600**	1	0.0
ind4	74	292	25	0	0	0	0.0	**1086**	1	0.0
ind5	114	456	33	0	0	0	0.0	**1341**	1	0.0
rc01	21	70	10	0	0	0	0.0	**25,980**	1	0.0
rc02	87	352	30	2	2	1	0.0	**41,350**	1	0.0

Table 7 continued

Instance	Original			Presolved				Optimum	N	t [s]
	$\|V\|$	$\|A\|$	$\|T\|$	$\|V\|$	$\|A\|$	$\|T\|$	t [s]			
rc03	109	404	50	0	0	0	0.0	**54,160**	1	0.0
rc04	121	394	70	0	0	0	0.0	**59,070**	1	0.0
rc05	247	972	100	0	0	0	0.0	**74,070**	1	0.0
rc06	2502	12, 488	100	1991	8880	90	1.1	**79,714**	1	4.9
rc07	2740	13, 156	200	2001	8674	139	1.8	**108,740**	7	7.2
rc08	7527	36, 340	200	6840	30, 894	186	4.8	**112,564**	55	156.8
rc09	6128	30, 528	200	5290	24, 238	168	4.0	**111,005**	1	86.3
rc10	1572	6490	500	572	2078	163	0.8	**164,150**	1	1.2
rc11	2858	11, 638	1000	1055	3676	337	2.7	**230,837**	1	3.5
rt01	262	1480	10	0	0	0	0.0	**2146**	1	0.0
rt02	788	3876	50	0	0	0	0.3	**45,852**	1	0.3
rt03	1725	8184	100	1430	6198	82	0.9	**7964**	5	2.7
rt04	9469	45, 486	100	9035	41, 352	94	4.2	**9693**	2717	736.1
rt05	15, 473	77, 856	200	14, 488	68, 570	190	7.2	**51,313**	57	2630.6

Bold numbers signify a superior performance

Table 8 Detailed computational results for SPG, test-set ES10000FST

Instance	Original			Presolved				Optimum	N	t [s]
	$\|V\|$	$\|A\|$	$\|T\|$	$\|V\|$	$\|A\|$	$\|T\|$	t [s]			
es10000fst01	27,019	78,814	10,000	4080	13,246	1621	36.8	**716,174,280**	1	50.9

Table 9 Detailed computational results for ESMT, test-set ESMT-R25

Instance	Original			Presolved				Optimum	N	t [s]
	$\|V\|$	$\|A\|$	$\|T\|$	$\|V\|$	$\|A\|$	$\|T\|$	t [s]			
R25K01EFST	39,277	94,524	25,000	92	294	40	34.7	**98.9612134**	1	40.1
R25K02EFST	39,306	94,978	25,000	59	180	30	38.4	**99.0370878**	1	47.1
R25K03EFST	39,549	96,348	25,000	3893	12, 466	1785	35.0	**99.2157207**	1	45.8
R25K04EFST	39,555	96,260	25,000	84	274	37	38.0	**98.9431392**	1	47.6
R25K05EFST	39,153	93,806	25,000	49	146	26	28.9	**99.4912321**	1	39.1
R25K06EFST	39,438	95,690	25,000	5990	19, 160	2804	12.7	**99.3728768**	1	29.6
R25K07EFST	39,900	98,180	25,000	47	140	24	38.4	**99.5646105**	1	51.6
R25K08EFST	39,529	95,920	25,000	65	200	32	39.6	**99.2662017**	1	48.5
R25K09EFST	39,732	97,060	25,000	3807	12, 238	1773	38.1	**99.0968636**	1	44.7
R25K10EFST	39,248	94,668	25,000	48	136	23	28.1	**99.1104801**	1	35.7
R25K11EFST	39,425	95,470	25,000	2661	8418	1239	42.1	**99.1216345**	1	47.5
R25K12EFST	39,293	94,888	25,000	3434	10, 960	1593	37.3	**99.1134447**	1	45.5
R25K13EFST	39,284	94,770	25,000	3328	10, 524	1566	26.4	**99.4005526**	1	33.0
R25K14EFST	40,063	98,534	25,000	3957	12, 746	1795	38.9	**99.2046414**	1	46.1
R25K15EFST	39,498	95,704	25,000	44	130	21	43.7	**99.2521324**	1	54.6

Bold numbers signify a superior performance

Table 10 Detailed computational results for ESMT, test-set ESMT-R50

Instance	Original			Presolved				Optimum	N	t [s]
	$\|V\|$	$\|A\|$	$\|T\|$	$\|V\|$	$\|A\|$	$\|T\|$	t [s]			
R50K01EFST	79,505	194,746	50,000	9521	30, 346	4427	120.2	**140.398764**	1	139.8
R50K02EFST	78,754	190,726	50,000	254	798	119	116.2	**139.955781**	1	145.3
R50K03EFST	78,964	191,358	50,000	7198	23, 066	3325	126.2	**140.006412**	1	140.3
R50K04EFST	789,83	191,484	50,000	56	174	28	142.5	**140.093852**	1	154.3
R50K05EFST	79,200	193,418	50,000	43	126	23	142.7	**139.995235**	1	196.5
R50K06EFST	79,480	194,744	50,000	9705	31, 120	4495	133.9	**140.348542**	1	164.9
R50K07EFST	79,046	192,228	50,000	13, 631	43, 506	6350	44.2	**140.249582**	1	86.0
R50K08EFST	79,175	192,822	50,000	108	378	41	111.7	**140.351147**	1	126.8
R50K09EFST	78,825	190,952	50,000	54	156	26	107.0	**140.363481**	1	120.6
R50K10EFST	78,948	191,740	50,000	73	236	33	40.8	**140.321093**	1	80.0
R50K11EFST	79,121	192,608	50,000	8136	25, 928	3797	122.1	**140.169756**	1	139.6
R50K12EFST	79,133	192,768	50,000	51	156	24	29.3	**140.201234**	1	57.7
R50K13EFST	78,972	191,348	50,000	7654	24, 410	3562	116.3	**140.03999**	1	131.6
R50K14EFST	79,326	193,440	50,000	51	156	24	135.7	**140.209795**	1	151.8
R50K15EFST	79,483	194,414	50,000	66	224	25	156.4	**140.447926**	1	170.4

Bold numbers signify a superior performance

Table 11 Detailed computational results for ESMT, test-set ESMT-R100

Instance	Original			Presolved				Optimum	N	t [s]
	$\|V\|$	$\|A\|$	$\|T\|$	$\|V\|$	$\|A\|$	$\|T\|$	t [s]			
R100K01EFST	157,869	383,172	100,000	50	156	24	554.3	**198.306705**	1	631.7
R100K02EFST	158,031	383,994	100,000	71	244	26	156.6	**197.970178**	1	370.1
R100K03EFST	158,290	384,990	100,000	18, 337	58, 020	8630	477.9	**198.022313**	1	640.7
R100K04EFST	158,205	385,292	100,000	64	204	29	550.8	**198.189607**	1	612.3
R100K05EFST	158,587	386,646	100,000	47	146	22	123.6	**198.153237**	1	372.1
R100K06EFST	158,514	386,086	100,000	73	276	25	117.7	**198.174481**	1	328.3
R100K07EFST	157,947	383,296	100,000	24, 847	79, 452	11, 545	112.4	**197.878416**	1	369.6
R100K08EFST	157,839	382,404	100,000	17, 086	54, 446	7977	471.3	**197.994433**	1	541.8
R100K09EFST	158,069	383,470	100,000	26, 909	85, 924	12, 571	152.3	**198.135832**	1	391.4
R100K10EFST	158,575	386,344	100,000	18, 012	57, 070	8424	448.3	**198.039905**	1	546.1
R100K11EFST	158,265	385,490	100,000	25, 924	83, 376	11, 924	124.7	**198.136332**	1	368.6
R100K12EFST	157,806	382,352	100,000	50	158	22	472.6	**198.384696**	1	570.0
R100K13EFST	157,660	381,462	100,000	27, 619	87, 894	12, 879	151.4	**198.053544**	1	457.4
R100K14EFST	158,516	386,662	100,000	50	162	22	510.5	**198.226993**	1	729.7
R100K15EFST	158,033	384,044	100,000	27, 235	87, 188	12, 652	151.3	**198.274048**	1	460.0

Bold numbers signify a superior performance

Table 12 Detailed computational results for SPG, test-set LIN

Instance	Original			Presolved				Dual	Primal	Gap %	N	t [s]
	$\|V\|$	$\|A\|$	$\|T\|$	$\|V\|$	$\|A\|$	$\|T\|$	t [s]					
lin01	53	160	4	0	0	0	0.0	**503**			1	0.0
lin02	55	164	6	0	0	0	0.0	**557**			1	0.0
lin03	57	168	8	0	0	0	0.0	**926**			1	0.0
lin04	157	532	6	0	0	0	0.0	**1239**			1	0.0
lin05	160	538	9	0	0	0	0.0	**1703**			1	0.0
lin06	165	548	14	0	0	0	0.0	**1348**			1	0.0
lin07	307	1052	6	0	0	0	0.0	**1885**			1	0.0
lin08	311	1060	10	0	0	0	0.0	**2248**			1	0.0
lin09	313	1064	12	0	0	0	0.0	**2752**			1	0.0
lin10	321	1080	20	0	0	0	0.0	**4132**			1	0.0
lin11	816	2920	10	0	0	0	0.0	**4280**			1	0.0
lin12	818	2924	12	47	144	8	0.0	**5250**			1	0.0
lin13	822	2932	16	0	0	0	0.0	**4609**			1	0.0
lin14	828	2944	22	0	0	0	0.0	**5824**			1	0.0
lin15	840	2968	34	0	0	0	0.0	**7145**			1	0.0
lin16	1981	7266	12	0	0	0	0.1	**6618**			1	0.1
lin17	1989	7282	20	0	0	0	0.1	**8405**			1	0.1
lin18	1994	7292	25	13	34	6	0.5	**9714**			1	0.5
lin19	2010	7324	41	105	350	11	0.1	**132,68**			1	0.1
lin20	3675	13, 418	11	0	0	0	0.1	**6673**			1	0.1

Table 12 continued

Instance	Original			Presolved				Dual	Primal	Gap %	N	t [s]
	$\|V\|$	$\|A\|$	$\|T\|$	$\|V\|$	$\|A\|$	$\|T\|$	t [s]					
lin21	3683	13, 434	20	129	422	9	0.2	**9143**			1	0.2
lin22	3692	13, 452	28	0	0	0	0.2	**10,519**			1	0.2
lin23	3716	13, 500	52	84	278	17	0.8	**17,560**			1	0.8
lin24	7998	29, 468	16	2970	10, 706	16	1.3	**15,076**			1	1.5
lin25	8007	29, 486	24	931	3254	20	4.4	**17,803**			1	4.4
lin26	8013	29, 498	30	0	0	0	3.4	**21,757**			1	3.4
lin27	8017	29, 506	36	94	302	18	3.3	**20,678**			1	3.3
lin28	8062	29, 596	81	354	1200	44	11.8	**32,584**			1	12.0
lin29	19, 083	71, 272	24	1022	3714	19	11.9	**23,765**			1	11.9
lin30	19, 091	71, 288	31	190	650	19	15.9	**27,684**			1	15.9
lin31	19, 100	71, 306	40	6577	24, 148	37	39.9	**31,696**			1	44.6
lin32	19, 112	71, 330	53	6902	25, 308	52	49.2	**39,832**			1	67.0
lin33	19, 177	71, 460	117	5711	20, 688	97	52.2	**56,061**			1	1436.7
lin34	38, 282	143, 042	34	11, 461	42, 424	33	157.1	**45,018**			1	158.2
lin35	38, 294	143, 066	45	14, 095	51, 962	45	99.1	**50,559**			1	120.5
lin36	38, 307	143, 092	58	17, 931	66, 096	57	102.5	**55,608**			1	263.1
lin37	38, 418	143, 314	172	23, 971	88, 916	169	128.7	97,130.5673	99, 737	2.7	1	> 54,340

Bold numbers signify a superior performance

Table 13 Detailed computational results for SPG, test-set PUC

Instance	Original			Presolved				Dual	Primal	Gap%	N	t [s]
	$\|V\|$	$\|A\|$	$\|T\|$	$\|V\|$	$\|A\|$	$\|T\|$	t [s]					
bip42p	1200	7964	200	990	7234	200	0.2	**24,657**			16, 9521	20,228.7
bip42u	1200	7964	200	989	7216	200	0.2	**236**			115, 338	11,107.5
bip52p	2200	15, 994	200	1817	14, 650	200	0.4	24,320.8464	24,595	1.1	30, 4769	> 54,340
bip52u	2200	15, 994	200	1818	14, 646	200	0.6	230.581893	234	1.5	302, 048	> 54,340
bip62p	1200	20, 004	200	1199	20, 000	200	0.3	22,586.7321	22,885	1.3	160, 340	> 54,340
bip62u	1200	20, 004	200	1199	20, 000	200	0.5	214.995514	220	2.3	175, 484	> 54,340
bipa2p	3300	36, 146	300	3139	35, 588	300	1.1	34,784.9831	35,333	1.6	80, 653	> 54,340
bipa2u	3300	36, 146	300	3138	35, 590	300	1.2	330.625186	340	2.8	113, 369	> 54,340
bipe2p	550	10, 026	50	550	10, 026	50	0.1	**5616**			1591	135.3
bipe2u	550	10, 026	50	550	10, 026	50	0.2	**54**			97	69.4
cc10-2p	1024	10, 240	135	1024	10, 240	135	0.4	34,533.385	35,342	2.3	4204	> 54,340
cc10-2u	1024	10, 240	135	1024	10, 240	135	0.8	334.807731	343	2.4	3412	> 54,340
cc11-2p	2048	22, 526	244	2048	22, 526	244	1.3	62,133.2953	63,640	2.4	2043	> 54,340
cc11-2u	2048	22, 526	244	2048	22, 526	244	1.9	602.532956	615	2.1	3182	> 54,340
cc12-2p	4096	49, 148	473	4096	49, 148	473	4.3	118,619.962	121,523	2.4	151	> 54,340
cc12-2u	4096	49, 148	473	4096	49, 148	473	4.3	1150.17905	1184	2.9	228	> 54,340
cc3-10p	1000	27, 000	50	1000	27, 000	50	0.6	12,704.9214	12,783	0.6	7733	> 54,340
cc3-10u	1000	27, 000	50	1000	27, 000	50	1.0	120.884585	126	4.2	419	> 54,340
cc3-11p	1331	39, 930	61	1331	39, 930	61	1.0	15,464.6087	15,596	0.8	5409	> 54,340
cc3-11u	1331	39, 930	61	1331	39, 930	61	1.1	144.648808	154	6.5	1	> 54,340

Table 13 continued

Instance	Original			Presolved				Dual	Primal	Gap %	N	t [s]
	$\|V\|$	$\|A\|$	$\|T\|$	$\|V\|$	$\|A\|$	$\|T\|$	t [s]					
cc3-12p	1728	57, 024	74	1728	57, 024	74	1.5	18,737.1085	18,853	0.6	4468	> 54,340
cc3-12u	1728	57, 024	74	1728	57, 024	74	1.7	174.291838	186	6.7	24	> 54,340
cc3-4p	64	576	8	64	576	8	0.0	**2338**			1	0.0
cc3-4u	64	576	8	64	576	8	0.0	**23**			1	0.0
cc3-5p	125	1500	13	125	1500	13	0.0	**3661**			1	0.8
cc3-5u	125	1500	13	125	1500	13	0.0	**36**			1	0.8
cc5-3p	243	2430	27	243	2430	27	0.1	**7299**			1901	4362.0
cc5-3u	243	2430	27	243	2430	27	0.1	**71**			207	669.4
cc6-2p	64	384	12	64	384	12	0.0	**3271**			1	0.1
cc6-2u	64	384	12	64	384	12	0.0	**32**			1	0.2
cc6-3p	729	8736	76	729	8736	76	0.3	20,193.3481	20,286	0.5	22, 679	> 54,340
cc6-3u	729	8736	76	729	8736	76	0.6	**197**			56	10,003.2
cc7-3p	2187	30, 616	222	2187	30, 616	222	1.7	55,404.3712	57,072	3.0	863	> 54,340
cc7-3u	2187	30, 616	222	2187	30, 616	222	1.8	536.675303	552	2.9	738	> 54,340
cc9-2p	512	4608	64	512	4608	64	0.2	16,916.6425	17,284	2.2	7586	> 54,340
cc9-2u	512	4608	64	512	4608	64	0.3	163.855778	168	2.5	6994	> 54,340
hc10p	1024	10, 240	512	1024	10, 240	512	0.4	59,287.7329	59,777	0.8	41, 290	> 54,340
hc10u	1024	10, 240	512	1024	10, 240	512	0.8	567.888889	575	1.3	53, 052	> 54,340
hc11p	2048	22, 528	1024	2048	22, 528	1024	1.2	11,7451.749	11,9854	2.0	14, 242	> 54,340
hc11u	2048	22, 528	1024	2048	22, 528	1024	2.1	1125.4	1164	3.4	13, 107	> 54,340
hc12p	4096	49, 152	2048	4096	49, 152	2048	4.8	232,919.332	236,573	1.6	1533	> 54,340
hc12u	4096	49, 152	2048	4096	49, 152	2048	7.6	2233.09091	2321	3.9	1	> 54,340

Table 13 continued

Instance	Original			Presolved				Dual	Primal	Gap %	N	t [s]
	$\|V\|$	$\|A\|$	$\|T\|$	$\|V\|$	$\|A\|$	$\|T\|$	t [s]					
hc6p	64	384	32	64	384	32	0.0	**4003**			1589	17.5
hc6u	64	384	32	64	384	32	0.0	**39**			693	5.0
hc7p	128	896	64	128	896	64	0.0	**7905**			251, 863	2797.7
hc7u	128	896	64	128	896	64	0.1	**77**			599, 283	4081.1
hc8p	256	2048	128	256	2048	128	0.1	**15,322**			304, 601	21,378.1
hc8u	256	2048	128	256	2048	128	0.1	145.714286	148	1.6	1, 166, 252	> 54,340
hc9p	512	4608	256	512	4608	256	0.2	29,983.7876	30,247	0.9	158, 503	> 54,340
hc9u	512	4608	256	512	4608	256	0.3	287.125	292	1.7	395, 816	> 54,340

Bold numbers signify a superior performance

Table 14 Detailed computational results for SPG, test-set SP

Instance	Original			Presolved				Optimum	N	t [s]
	$\|V\|$	$\|A\|$	$\|T\|$	$\|V\|$	$\|A\|$	$\|T\|$	t [s]			
antiwheel5	10	30	5	0	0	0	0.0	**7**	1	0.0
design432	8	40	4	0	0	0	0.0	**9**	1	0.0
oddcycle3	6	18	3	0	0	0	0.0	**4**	1	0.0
oddwheel3	7	18	4	0	0	0	0.0	**5**	1	0.0
se03	13	42	4	0	0	0	0.0	**12**	1	0.0
w13c29	783	4524	406	783	4524	406	0.4	**507**	6	37.3
w23c23	1081	6348	552	1081	6348	552	0.7	**689**	132	1190.0
w3c571	3997	20, 556	2284	3997	20, 556	2284	3.1	**2854**	1	321.6

Bold numbers signify a superior performance

Table 15 Detailed computational results for SPG, test-set TSPFST

Instance	Original			Presolved				Optimum	N	t [s]
	$\|V\|$	$\|A\|$	$\|T\|$	$\|V\|$	$\|A\|$	$\|T\|$	t [s]			
a280fst	313	656	279	0	0	0	0.0	**2502**	1	0.0
att48fst	139	404	48	58	186	23	0.0	**30, 236**	1	0.0
att532fst	1468	4304	532	270	854	103	0.4	**84, 009**	1	0.5
berlin52fst	89	208	52	0	0	0	0.0	**6760**	1	0.0
bier127fst	258	714	127	0	0	0	0.0	**104, 284**	1	0.0
d1291fst	1365	2912	1291	0	0	0	0.0	**481, 421**	1	0.0
d1655fst	1906	4166	1655	33	100	16	0.0	**584, 948**	1	0.0
d198fst	232	512	198	0	0	0	0.0	**129, 175**	1	0.0
d2103fst	2206	4544	2103	0	0	0	0.0	**769, 797**	1	0.0
d493fst	1055	2946	493	92	280	46	0.1	**320, 137**	1	0.1
d657fst	1416	3956	657	131	418	56	0.3	**471, 589**	1	0.7
dsj1000fst	2562	7310	1000	69	220	28	0.2	**17, 564, 659**	1	0.3
eil101fst	330	1076	101	166	550	56	0.1	**605**	1	0.2
eil51fst	181	578	51	114	376	39	0.0	**409**	1	0.1
eil76fst	237	756	76	92	302	35	0.1	**513**	1	0.1
fl1400fst	2694	9092	1400	441	1648	221	0.5	**17, 980, 523**	1	7.8
fl1577fst	2413	6824	1577	94	320	48	0.1	**19, 825, 626**	1	0.3
fl3795fst	4859	13, 078	3795	444	1656	222	3.0	**25, 529, 856**	1	8.6
fl417fst	732	2168	417	124	438	52	0.1	**10, 883, 190**	1	0.2
fnl4461fst	17, 127	54, 704	4461	7720	25, 748	2484	22.3	**182, 361**	135	205.6
gil262fst	537	1446	262	34	104	22	0.0	**2306**	1	0.0
kroA100fst	197	500	100	0	0	0	0.0	**20, 401**	1	0.0
kroA150fst	389	1124	150	126	404	49	0.1	**25, 700**	1	0.1

Table 15 continued

Instance	Original			Presolved				Optimum	N	t [s]
	$\|V\|$	$\|A\|$	$\|T\|$	$\|V\|$	$\|A\|$	$\|T\|$	t [s]			
kroA200fst	500	1428	200	0	0	0	0.0	**28, 652**	1	0.0
kroB100fst	230	626	100	0	0	0	0.0	**21, 211**	1	0.0
kroB150fst	420	1238	150	62	204	25	0.1	**25, 217**	1	0.1
kroB200fst	480	1340	200	102	330	45	0.1	**28, 803**	1	0.1
kroC100fst	244	674	100	45	146	18	0.0	**20, 492**	1	0.0
kroD100fst	216	576	100	12	34	7	0.0	**20, 437**	1	0.0
kroE100fst	226	612	100	46	140	21	0.0	**21, 245**	1	0.0
lin105fst	216	646	105	43	134	20	0.0	**13, 429**	1	0.0
lin318fst	678	2060	318	78	252	38	0.1	**39, 335**	1	0.1
linhp318fst	678	2060	318	78	252	38	0.1	**39, 335**	1	0.1
nrw1379fst	5096	16, 210	1379	2370	7936	751	3.8	**56, 207**	1	15.4
p654fst	777	1734	654	0	0	0	0.0	**314, 925**	1	0.0
pcb1173fst	1912	4446	1173	17	48	10	0.0	**53, 301**	1	0.0
pcb3038fst	5829	15, 104	3038	77	232	40	0.4	**131, 895**	1	0.4
pcb442fst	503	1062	442	0	0	0	0.0	**47, 675**	1	0.0
pla7397fst	8790	19, 630	7397	97	296	51	0.1	**22, 481, 625**	1	0.1
pr1002fst	1473	3430	1002	0	0	0	0.0	**243, 176**	1	0.0
pr107fst	111	220	107	0	0	0	0.0	**34, 850**	1	0.0
pr124fst	154	330	124	0	0	0	0.0	**52, 759**	1	0.0
pr136fst	196	500	136	0	0	0	0.0	**86, 811**	1	0.0
pr144fst	221	570	144	0	0	0	0.0	**52, 925**	1	0.0
pr152fst	308	862	152	0	0	0	0.0	**64, 323**	1	0.0
pr226fst	255	538	226	0	0	0	0.0	**70, 700**	1	0.0

Table 15 continued

Instance	Original			Presolved				Optimum	N	t [s]
	$\|V\|$	$\|A\|$	$\|T\|$	$\|V\|$	$\|A\|$	$\|T\|$	t [s]			
pr2392fst	3398	7932	2392	0	0	0	0.0	**358, 989**	1	0.0
pr264fst	280	574	264	0	0	0	0.0	**41, 400**	1	0.0
pr299fst	420	1000	299	0	0	0	0.0	**44, 671**	1	0.0
pr439fst	572	1324	439	0	0	0	0.0	**97, 400**	1	0.0
pr76fst	168	494	76	33	94	16	0.0	**95, 908**	1	0.0
rat195fst	560	1740	195	182	594	70	0.2	**2386**	1	0.3
rat575fst	1986	6352	575	892	2940	319	1.6	**6808**	1	2.3
rat783fst	2397	7430	783	900	2976	338	1.7	**8883**	1	2.7
rat99fst	269	798	99	0	0	0	0.0	**1225**	1	0.0
rd100fst	201	506	100	0	0	0	0.0	**764, 269, 099**	1	0.0
rd400fst	1001	2838	400	161	514	65	0.1	**1, 490, 972, 010**	1	0.2
rl11849fst	13, 963	30, 630	11, 849	88	266	50	0.1	**8, 779, 590**	1	0.1
rl1304fst	1562	3388	1304	18	54	10	0.0	**236, 649**	1	0.0
rl1323fst	1598	3500	1323	0	0	0	0.0	**253, 620**	1	0.0
rl1889fst	2382	5348	1889	67	216	27	0.0	**295, 208**	1	0.0
rl5915fst	6569	13, 960	5915	34	102	19	0.0	**533, 226**	1	0.0
rl5934fst	6827	14, 730	5934	31	100	19	0.0	**529, 890**	1	0.0
st70fst	133	338	70	0	0	0	0.0	**626**	1	0.0
ts225fst	225	448	225	0	0	0	0.0	**1120**	1	0.0
tsp225fst	242	504	225	0	0	0	0.0	**356, 850**	1	0.0
u1060fst	1835	4858	1060	63	220	32	0.1	**21, 265, 372**	1	1.3
u1432fst	1432	2862	1432	0	0	0	0.0	**1465**	1	0.0
u159fst	184	372	159	0	0	0	0.0	**390**	1	0.0

Table 15 continued

Instance	Original			Presolved				Optimum	N	t [s]
	$\|V\|$	$\|A\|$	$\|T\|$	$\|V\|$	$\|A\|$	$\|T\|$	t [s]			
u1817fst	1831	3692	1817	0	0	0	0.0	**5, 513, 053**	1	0.0
u2152fst	2167	4368	2152	0	0	0	0.0	**6, 253, 305**	1	0.0
u2319fst	2319	4636	2319	0	0	0	0.0	**2322**	1	0.0
u574fst	990	2516	574	0	0	0	0.1	**3, 509, 275**	1	0.1
u724fst	1180	3074	724	11	32	7	0.0	**4, 069, 628**	1	0.1
vm1084fst	1679	4116	1084	26	80	15	0.1	**2, 248, 390**	1	0.1
vm1748fst	2856	7282	1748	191	612	85	0.4	**3, 194, 670**	1	0.5

Bold numbers signify a superior performance

Table 16 Detailed computational results for SPG, test-set vienna-geo-original

Instance	Original			Presolved				Optimum	N	t [s]
	$\|V\|$	$\|A\|$	$\|T\|$	$\|V\|$	$\|A\|$	$\|T\|$	t [s]			
G101	67, 966	164,970	100	669	2544	66	4.7	**3, 492, 405**	1	5.2
G102	111, 707	321,008	2052	9285	30, 744	1592	20.0	**15, 187, 538**	1	67.4
G103	135, 543	403,606	3033	12, 804	42, 348	2277	26.9	**19, 938, 744**	1	120.7
G104	158, 212	480,044	3914	16, 492	54, 266	2929	36.7	**26, 165, 528**	1	211.7
G105	79, 244	202,378	550	3103	10, 712	409	8.6	**12, 507, 877**	1	25.0
G106	204, 621	636,272	5556	1379	4618	233	70.1	**44, 547, 208**	1	469.7
G107	85, 568	228,226	938	1177	3824	247	7.5	**7, 325, 530**	1	9.8
G201	44, 624	112,410	190	775	2588	123	3.4	**3, 484, 028**	1	3.8
G202	62, 174	175,124	1015	1773	5766	357	7.5	**6, 849, 423**	1	9.3
G203	88, 728	267,250	2041	1817	5962	323	19.9	**13, 155, 210**	1	43.1
G204	50, 002	130,406	386	905	2916	181	4.9	**5, 313, 548**	1	5.2
G205	120, 866	374,624	3224	3873	12, 768	639	31.7	**24, 819, 583**	1	179.4
G206	60, 446	165,880	803	254	834	56	7.8	**9, 175, 622**	1	10.7
G207	42, 481	105,104	97	0	0	0	1.8	**2, 265, 834**	1	1.8
G301	80, 736	197,500	191	1313	4932	152	6.7	**4, 797, 441**	1	8.0
G302	117, 756	330,306	1879	354	1112	91	13.8	**13, 300, 990**	1	23.4
G303	147, 718	428,352	2992	10, 545	33, 992	1853	35.0	**27, 941, 456**	1	68.0
G304	86, 413	217,744	419	77	242	24	6.7	**6, 721, 180**	1	6.8
G305	172, 687	511,650	3902	2540	8404	421	42.6	**40, 632, 152**	1	126.7
G306	196, 404	600,072	4937	2329	7616	425	49.4	**33, 949, 874**	1	381.3
G307	235, 686	732,186	6313	3647	12, 044	610	79.3	**51, 219, 090**	1	524.8
G308	78, 834	191,464	88	1120	4464	72	6.5	**4, 699, 474**	3	11.0
G309	97, 928	257,264	902	576	1872	127	12.1	**11, 256, 303**	1	14.8

Bold numbers signify a superior performance

Table 17 Detailed computational results for SPG, test-set vienna-geo-advanced

Instance	Original			Presolved				Optimum	N	t [s]
	$\|V\|$	$\|A\|$	$\|T\|$	$\|V\|$	$\|A\|$	$\|T\|$	t [s]			
G101a	10, 734	32, 690	96	286	1050	44	4.3	**3, 492, 405**	1	4.5
G102a	27, 896	87, 850	2003	8975	29, 614	1543	22.2	**15, 187, 538**	1	64.1
G103a	36, 270	114, 740	2930	5448	17, 926	970	26.7	**19, 938, 744**	1	102.2
G104a	44, 251	140, 058	3776	16, 048	52, 674	2830	37.8	**26, 165, 528**	1	223.0
G105a	14, 586	44, 900	525	3029	10, 410	402	7.6	**12, 507, 877**	1	22.5
G106a	62, 618	200, 134	5373	1380	4618	233	62.8	**44, 547, 208**	1	506.1
G107a	15, 536	47, 716	893	1181	3850	247	6.2	**7, 325, 530**	1	8.8
G201a	8286	25, 234	188	772	2580	124	3.2	**3, 484, 028**	1	3.6
G202a	14, 028	43, 220	985	1771	5752	360	6.6	**6, 849, 423**	1	8.7
G203a	25, 651	81, 220	1999	1803	5910	320	18.2	**13, 155, 210**	1	40.9
G204a	9939	30, 498	376	868	2806	176	2.8	**5, 313, 548**	1	3.2
G205a	37, 398	118, 646	3146	3815	12, 590	624	28.2	**24, 819, 583**	1	152.0
G206a	13, 688	42, 394	789	294	974	60	6.8	**9, 175, 622**	1	9.7
G207a	7565	23, 042	98	2	2	1	1.7	**2, 265, 834**	1	1.7
G301a	13, 291	40, 522	181	952	3484	125	6.3	**4, 797, 441**	1	6.8
G302a	24, 951	77, 294	1797	403	1274	101	12.5	**13, 300, 990**	1	21.7
G303a	37, 085	115, 422	2915	1308	4250	231	29.5	**27, 941, 456**	1	77.3
G304a	15, 213	46, 658	403	117	380	30	5.7	**6, 721, 180**	1	5.7
G305a	47, 016	147, 722	3809	14, 024	45, 076	2425	46.7	**40, 632, 152**	1	127.6
G306a	55, 423	175, 558	4766	2329	7614	425	50.4	**33, 949, 874**	1	369.2
G307a	71, 184	227, 232	6107	3648	12, 048	610	68.8	**51, 219, 090**	1	536.1
G308a	13, 298	40, 702	86	702	2740	67	6.3	**4, 699, 474**	1	7.3
G309a	18, 704	57, 702	868	2259	7478	402	11.3	**11, 256, 303**	1	13.4

Bold numbers signify a superior performance

Table 18 Detailed computational results for SPG, test-set vienna-i-simple

Instance	Original			Presolved				Optimum	N	t [s]
	$\|V\|$	$\|A\|$	$\|T\|$	$\|V\|$	$\|A\|$	$\|T\|$	t [s]			
I001	30, 190	95, 496	1184	211	646	75	2.0	**253, 921, 201**	1	2.1
I002	49, 920	155, 742	1665	642	1940	186	5.9	**399, 809, 303**	1	6.7
I003	44, 482	146, 838	3222	443	1342	126	9.1	**788, 774, 494**	1	12.3
I004	5556	17, 104	570	0	0	0	0.1	**279, 512, 692**	1	0.1
I005	10, 284	31, 960	1017	0	0	0	0.2	**390, 876, 350**	1	0.2
I006	31, 754	105, 750	2202	544	1640	175	6.8	**504, 526, 035**	1	10.3
I007	15, 122	48, 742	737	136	400	42	0.9	**177, 909, 660**	1	1.0
I008	15, 714	51, 134	871	136	404	46	1.8	**201, 788, 202**	1	1.9
I009	33, 188	104, 014	1262	297	902	100	2.3	**275, 558, 727**	1	2.5
I010	29, 905	94, 914	943	151	450	58	1.3	**207, 889, 674**	1	1.3
I011	25, 195	82, 596	1428	734	2258	207	2.3	**317, 589, 880**	1	3.1
I012	12, 355	39, 924	503	32	98	16	0.3	**118, 893, 243**	1	0.3
I013	18, 242	57, 952	891	66	188	32	1.3	**193, 190, 339**	1	1.3
I014	12, 715	41, 264	475	10	26	5	0.3	**105, 173, 465**	1	0.3
I015	48, 833	159, 974	2493	424	1330	123	7.0	**592, 240, 832**	1	8.1
I016	72, 038	230, 110	4391	742	2290	207	16.3	**1, 110, 914, 620**	1	18.1
I017	15, 095	48, 182	478	76	234	21	0.5	**109, 739, 695**	1	0.5
I018	31, 121	102, 226	1898	982	2914	274	4.5	**463, 887, 832**	1	6.1
I019	25, 946	83, 290	866	320	970	90	1.7	**217, 647, 693**	1	1.9
I020	21, 808	69, 842	594	98	308	35	0.9	**146, 515, 460**	1	1.0
I021	16, 013	50, 538	392	17	46	7	0.5	**106, 470, 644**	1	0.5
I022	16, 224	51, 382	437	54	156	19	0.7	**106, 799, 980**	1	0.7
I023	22, 805	70, 614	582	92	294	31	0.7	**131, 044, 872**	1	0.7
I024	68, 464	217, 464	3001	275	848	79	10.3	**758, 483, 415**	1	11.6
I025	23, 412	75, 904	945	474	1488	153	3.4	**232, 790, 758**	1	3.5
I026	47, 429	158, 614	3334	1420	4372	409	10.8	**928, 032, 223**	1	12.4
I027	85, 085	277, 776	3954	1166	3564	291	16.0	**976, 812, 226**	1	17.2
I028	72, 701	230, 860	1790	176	546	59	15.8	**384, 053, 191**	1	15.8
I029	69, 988	223, 608	2162	349	1100	93	12.4	**492, 193, 565**	1	12.7
I030	33, 188	107, 360	1263	148	450	39	3.2	**321, 646, 787**	1	3.3
I031	54, 351	176, 422	2182	155	482	42	5.2	**578, 284, 709**	1	5.2
I032	56, 023	182, 798	3017	800	2404	244	6.2	**773, 096, 651**	1	7.2
I033	18, 555	59, 460	636	59	174	25	1.5	**134, 461, 857**	1	1.5
I034	22, 311	71, 032	735	64	186	21	1.8	**165, 115, 148**	1	1.8
I035	30, 585	100, 908	1704	129	386	49	3.5	**414, 440, 370**	1	4.0
I036	37, 208	120, 712	1411	125	402	36	6.1	**375, 260, 864**	1	6.6
I037	13, 694	44, 252	427	13	36	7	1.2	**105, 720, 727**	1	1.2
I038	18, 747	61, 278	967	679	2106	169	1.8	**255, 767, 543**	1	2.5

Table 18 continued

Instance	Original			Presolved				Optimum	N	t [s]
	$\|V\|$	$\|A\|$	$\|T\|$	$\|V\|$	$\|A\|$	$\|T\|$	t [s]			
I039	8755	28, 898	347	88	258	38	0.6	**85, 566, 290**	1	0.6
I040	40, 389	131, 640	1762	398	1236	121	5.8	**431, 498, 867**	1	6.0
I041	47, 197	150, 614	1193	181	554	65	5.3	**301, 914, 840**	1	5.3
I042	51, 896	171, 100	2171	131	394	39	6.8	**532, 131, 412**	1	6.9
I043	10, 398	33, 574	367	108	328	41	0.9	**95, 722, 094**	1	0.9
I044	68, 905	227, 778	3358	352	1082	90	10.7	**804, 532, 332**	1	13.5
I045	14, 685	46, 932	421	80	234	26	0.5	**105, 944, 062**	1	0.6
I046	70, 843	234, 418	3598	172	516	50	12.1	**925, 470, 052**	1	13.4
I047	28, 524	92, 502	2354	2176	6606	622	5.5	**695, 163, 406**	1	8.4
I048	13, 189	42, 438	358	0	0	0	0.5	**91, 509, 264**	1	0.5
I049	30, 857	99, 182	990	159	468	51	2.6	**294, 811, 505**	1	2.6
I050	43, 073	142, 552	2868	3449	10, 540	920	10.4	**792, 599, 114**	1	18.9
I051	27, 028	90, 812	1524	137	406	42	4.7	**357, 230, 839**	1	5.5
I052	2363	7522	40	0	0	0	0.0	**13, 309, 487**	1	0.0
I053	3224	10, 570	126	19	52	8	0.1	**30, 854, 904**	1	0.1
I054	3803	12, 426	38	0	0	0	0.0	**15, 841, 596**	1	0.0
I055	13, 332	43, 160	570	112	338	46	0.7	**144, 164, 924**	1	0.8
I056	1991	6352	51	0	0	0	0.0	**14, 171, 206**	1	0.0
I057	33, 231	110, 298	1569	112	340	40	3.2	**412, 746, 415**	1	3.9
I058	23, 527	79, 256	1256	169	538	42	1.2	**305, 024, 188**	1	1.3
I059	9287	29, 950	363	49	134	22	0.2	**107, 617, 854**	1	0.2
I060	42, 008	13, 5144	1242	160	504	54	5.7	**337, 290, 460**	1	5.7
I061	39, 160	127, 318	1458	171	532	46	7.1	**363, 042, 722**	1	7.7
I062	66, 048	220, 982	3343	122	374	43	7.2	**792, 941, 137**	1	7.6
I063	26, 840	87, 322	1645	777	2366	214	3.7	**459, 801, 704**	1	4.4
I064	63, 158	214, 690	3458	6440	20, 058	1597	19.7	**863, 103, 567**	1	36.3
I065	3898	12, 712	144	12	36	9	0.2	**32, 965, 718**	1	0.2
I066	15, 038	49, 192	551	70	212	28	0.4	**174, 219, 813**	1	0.4
I067	20, 547	66, 460	627	403	1256	121	1.5	**175, 540, 750**	1	1.7
I068	33, 118	110, 254	1553	353	1066	100	2.7	**420, 730, 046**	1	2.9
I069	9574	32, 416	543	258	804	71	0.9	**135, 161, 583**	1	1.0
I070	15, 079	49, 216	550	123	364	48	1.7	**136, 700, 139**	1	1.8
I071	33, 203	108, 854	1494	233	684	70	2.9	**382, 539, 099**	1	3.1
I072	26, 948	88, 388	993	110	338	24	2.0	**289, 019, 226**	1	2.0
I073	21, 653	70, 342	1847	115	336	44	2.8	**663, 004, 987**	1	3.5
I074	13, 316	44, 066	653	17	50	9	0.7	**165, 573, 383**	1	0.7
I075	57, 551	190, 762	2973	110	336	33	8.0	**815, 404, 026**	1	8.5
I076	14, 023	45, 790	598	71	208	31	0.9	**166, 249, 692**	1	0.9
I077	20, 856	68, 474	1787	3514	10, 400	882	4.5	**472, 503, 150**	1	10.8

Table 18 continued

Instance	Original			Presolved				Optimum	N	t [s]
	$\|V\|$	$\|A\|$	$\|T\|$	$\|V\|$	$\|A\|$	$\|T\|$	t [s]			
I078	13, 294	43, 896	835	86	244	37	1.1	**185, 525, 490**	1	1.1
I079	19, 867	62, 542	565	757	2598	213	2.5	**150, 506, 933**	1	2.9
I080	18, 695	59, 416	548	313	966	92	1.7	**164, 299, 652**	1	1.9
I081	25, 081	81, 478	888	53	154	27	2.4	**247, 527, 679**	1	2.5
I082	15, 592	49, 576	515	0	0	0	0.9	**147, 407, 632**	1	1.0
I083	89, 596	297, 166	4991	65	202	21	11.8	**1, 405, 593, 860**	1	13.3
I084	44, 934	147, 454	2319	95	318	26	4.7	**627, 187, 559**	1	6.8
I085	9113	28, 982	301	98	340	29	0.4	**80, 628, 079**	1	0.4

Bold numbers signify a superior performance

Table 19 Detailed computational results for SPG, test-set vienna-i-advanced

Instance	Original			Presolved				Optimum	N	t [s]
	$\|V\|$	$\|A\|$	$\|T\|$	$\|V\|$	$\|A\|$	$\|T\|$	t [s]			
I001a	14, 675	44, 110	941	212	638	72	1.8	**253, 921, 201**	1	1.9
I002a	23, 800	71, 516	1282	635	1918	186	4.6	**399, 809, 303**	1	5.5
I003a	16, 270	47, 838	2336	440	1332	125	7.6	**788, 774, 494**	1	10.7
I004a	867	2476	263	19	48	11	0.1	**279, 512, 692**	1	0.1
I005a	1677	4860	491	0	0	0	0.1	**390, 876, 350**	1	0.1
I006a	13, 339	39, 064	1842	104	316	28	5.6	**504, 526, 035**	1	9.6
I007a	6873	20, 598	599	128	370	42	0.8	**177, 909, 660**	1	0.8
I008a	6522	19, 258	708	101	296	33	1.5	**201, 788, 202**	1	1.6
I009a	14, 977	44, 870	1053	306	924	101	1.8	**275, 558, 727**	1	2.0
I010a	13, 041	39, 090	782	156	470	59	0.9	**207, 889, 674**	1	0.9
I011a	9298	27, 370	1202	709	2172	200	2.2	**317, 589, 880**	1	2.9
I012a	3500	10, 428	387	0	0	0	0.1	**118, 893, 243**	1	0.1
I013a	7147	21, 216	670	67	192	33	1.0	**193, 190, 339**	1	1.0
I014a	3577	10, 622	364	0	0	0	0.1	**105, 173, 465**	1	0.1
I015a	20, 573	61, 082	2119	407	1270	120	5.9	**592, 240, 832**	1	7.2
I016a	27, 214	79, 648	3434	507	1548	154	11.6	**1, 110, 914, 620**	1	14.0
I017a	7571	23, 142	386	0	0	0	0.3	**109, 739, 695**	1	0.3
I018a	12, 258	36, 028	1549	992	2942	276	3.3	**463, 887, 832**	1	4.8
I019a	11, 693	35, 248	732	278	846	79	1.3	**217, 647, 693**	1	1.4
I020a	6405	19, 128	508	58	180	18	0.5	**146, 515, 460**	1	0.5
I021a	5195	15, 722	295	102	306	27	0.2	**106, 470, 644**	1	0.2
I022a	8869	27, 102	356	64	188	24	0.5	**106, 799, 980**	1	0.5
I023a	13, 724	41, 726	403	222	672	64	0.5	**131, 044, 872**	1	0.5
I024a	32, 357	96, 500	2511	73	214	28	8.8	**758, 483, 415**	1	9.5
I025a	10, 055	29, 922	833	73	228	28	2.8	**232, 790, 758**	1	3.0
I026a	18, 155	53, 136	2661	1687	5180	496	8.4	**928, 032, 223**	1	10.2

Table 19 continued

Instance	Original			Presolved				Optimum	N	t [s]
	$\|V\|$	$\|A\|$	$\|T\|$	$\|V\|$	$\|A\|$	$\|T\|$	t [s]			
I027a	40, 772	121, 110	3490	109	346	33	14.7	**976, 812, 226**	1	16.3
I028a	43, 690	132, 922	1597	255	790	85	14.8	**384, 053, 191**	1	14.9
I029a	32, 979	99, 254	1946	270	856	73	9.2	**492, 193, 565**	1	9.4
I030a	12, 941	38, 558	1093	151	460	39	2.2	**321, 646, 787**	1	2.3
I031a	21, 054	62, 820	1832	156	484	42	3.6	**578, 284, 709**	1	3.6
I032a	21, 345	62, 706	2454	344	1058	90	5.3	**773, 096, 651**	1	6.2
I033a	8500	25, 400	548	252	770	76	1.0	**134, 461, 857**	1	1.1
I034a	9128	27, 336	606	142	412	48	1.1	**165, 115, 148**	1	1.2
I035a	13, 129	38, 840	1428	118	352	47	2.8	**414, 440, 370**	1	3.1
I036a	17, 036	50, 964	1258	318	984	74	5.2	**375, 260, 864**	1	5.8
I037a	5886	17, 738	392	60	180	21	0.8	**105, 720, 727**	1	0.8
I038a	7733	22, 956	798	693	2152	180	1.3	**255, 767, 543**	1	1.9
I039a	3719	11, 066	306	34	104	10	0.4	**85, 566, 290**	1	0.4
I040a	18, 837	56, 312	1501	165	512	49	5.6	**431, 498, 867**	1	5.7
I041a	22, 466	67, 736	1014	92	272	36	2.9	**301, 914, 840**	1	2.9
I042a	23, 925	71, 612	1923	116	346	34	5.4	**532, 131, 412**	1	5.6
I043a	4511	13, 480	335	99	288	35	0.7	**95, 722, 094**	1	0.7
I044a	31, 500	93, 514	2954	1327	4108	296	9.1	**804, 532, 332**	1	11.5
I045a	6775	20, 454	378	83	244	26	0.4	**105, 944, 062**	1	0.4
I046a	32, 376	96, 108	3154	163	482	50	9.3	**925, 470, 052**	1	11.0
I047a	10, 622	30, 880	1791	1365	4126	392	7.7	**695, 163, 406**	1	8.6
I048a	4920	14, 712	320	0	0	0	0.3	**91, 509, 264**	1	0.3
I049a	15, 045	45, 426	821	157	460	51	2.2	**294, 811, 505**	1	2.3
I050a	17, 787	52, 352	2232	3357	10, 250	902	9.2	**792, 599, 114**	1	17.3
I051a	12, 130	35, 784	1337	146	440	43	3.9	**357, 230, 839**	1	4.8
I052a	160	474	23	0	0	0	0.0	**13, 309, 487**	1	0.0
I053a	693	2046	102	26	72	13	0.0	**30, 854, 904**	1	0.0
I054a	540	1634	25	0	0	0	0.0	**15, 841, 596**	1	0.0
I055a	4701	13, 958	483	100	284	45	0.5	**144, 164, 924**	1	0.5
I056a	290	878	34	0	0	0	0.0	**14, 171, 206**	1	0.0
I057a	13, 078	38, 736	1346	178	546	64	2.8	**412, 746, 415**	1	3.4
I058a	7877	23, 314	997	156	494	39	0.9	**305, 024, 188**	1	1.0
I059a	2800	8314	286	31	86	11	0.1	**107, 617, 854**	1	0.1
I060a	18, 991	57, 072	1158	191	582	70	4.6	**337, 290, 460**	1	4.6
I061a	20, 958	62, 930	1337	153	464	49	5.8	**363, 042, 722**	1	6.3
I062a	23, 714	70, 610	2812	94	280	30	6.2	**792, 941, 137**	1	6.5
I063a	9600	28, 084	1291	950	2898	255	3.1	**459, 801, 704**	1	3.9
I064a	31, 712	93, 422	3182	6460	20, 152	1609	18.3	**863, 103, 567**	1	35.9
I065a	1185	3512	119	62	194	26	0.1	**32, 965, 718**	1	0.1

Table 19 continued

Instance	Original			Presolved				Optimum	N	t [s]
	$\|V\|$	$\|A\|$	$\|T\|$	$\|V\|$	$\|A\|$	$\|T\|$	t [s]			
I066a	4551	13, 642	417	59	182	24	0.3	**174, 219, 813**	1	0.3
I067a	10, 318	31, 176	579	407	1272	123	1.3	**175, 540, 750**	1	1.5
I068a	12, 191	36, 046	1302	321	976	91	1.9	**420, 730, 046**	1	2.2
I069a	3508	10, 312	452	269	844	73	0.7	**135, 161, 583**	1	0.8
I070a	6739	20, 128	511	147	438	52	1.4	**136, 700, 139**	1	1.4
I071a	12, 772	37, 772	1281	117	362	36	2.2	**382, 539, 099**	1	2.4
I072a	11, 628	34, 822	851	92	268	38	1.1	**289, 019, 226**	1	1.1
I073a	7510	21, 746	1337	1069	3244	324	2.5	**663, 004, 987**	1	3.1
I074a	4441	13, 124	548	37	110	13	0.3	**165, 573, 383**	1	0.3
I075a	23, 195	68, 724	2498	102	300	33	6.3	**815, 404, 026**	1	6.7
I076a	4909	14, 536	498	20	54	11	0.6	**166, 249, 692**	1	0.6
I077a	9153	26, 726	1490	3509	10, 388	880	4.1	**472, 503, 150**	1	11.0
I078a	5864	17, 324	692	168	486	58	1.0	**185, 525, 490**	1	1.0
I079a	7933	23, 614	497	732	2516	205	2.0	**150, 506, 933**	1	2.5
I080a	7589	22, 512	499	307	950	92	1.0	**164, 299, 652**	1	1.1
I081a	10, 747	32, 058	751	85	246	45	1.9	**247, 527, 679**	1	1.9
I082a	5850	17, 386	435	29	82	14	0.7	**147, 407, 632**	1	0.7
I083a	34, 221	100, 602	4138	326	1010	86	8.9	**1, 405, 593, 860**	1	10.3
I084a	17, 050	50, 402	1918	1265	3922	306	4.0	**627, 187, 559**	1	5.8
I085a	2780	8246	243	0	0	0	0.2	**80, 628, 079**	1	0.2

Bold numbers signify a superior performance

Table 20 Detailed computational results for SPG, test-set VLSI

Instance	Original			Presolved				Optimum	N	t [s]
	$\|V\|$	$\|A\|$	$\|T\|$	$\|V\|$	$\|A\|$	$\|T\|$	t [s]			
alue2087	1244	3942	34	0	0	0	0.0	**1049**	1	0.0
alue2105	1220	3716	34	0	0	0	0.0	**1032**	1	0.0
alue3146	3626	11, 738	64	0	0	0	0.1	**2240**	1	0.1
alue5067	3524	11, 120	68	50	146	16	0.1	**2586**	1	0.1
alue5345	5179	16, 330	68	62	202	17	0.9	**3507**	1	0.9
alue5623	4472	13, 876	68	20	56	9	0.6	**3413**	1	0.6
alue5901	11, 543	36, 858	68	41	124	18	0.7	**3912**	1	0.7
alue6179	3372	10, 426	67	0	0	0	0.1	**2452**	1	0.1
alue6457	3932	12, 274	68	0	0	0	0.1	**3057**	1	0.1
alue6735	4119	13, 392	68	140	446	20	0.1	**2696**	1	0.1
alue6951	2818	8838	67	57	172	19	0.1	**2386**	1	0.1
alue7065	34, 046	10, 9682	544	41	122	17	19.5	**23, 881**	1	19.5
alue7066	6405	20, 908	16	2281	7962	9	0.7	**2256**	1	0.7

Table 20 continued

Instance	Original			Presolved				Optimum	N	t [s]
	$\|V\|$	$\|A\|$	$\|T\|$	$\|V\|$	$\|A\|$	$\|T\|$	t [s]			
alue7080	34, 479	110, 988	2344	862	2736	344	9.5	**62, 449**	1	9.9
alue7229	940	2948	34	0	0	0	0.0	**824**	1	0.0
alut0787	1160	4178	34	0	0	0	0.0	**982**	1	0.0
alut0805	966	3332	34	0	0	0	0.0	**958**	1	0.0
alut1181	3041	11, 386	64	0	0	0	0.2	**2353**	1	0.2
alut2010	6104	22, 022	68	52	162	13	0.3	**3307**	1	0.3
alut2288	9070	33, 190	68	0	0	0	0.9	**3843**	1	0.9
alut2566	5021	18, 110	68	32	96	14	0.7	**3073**	1	0.7
alut2610	33, 901	125, 632	204	119	408	9	41.4	**12, 239**	1	41.4
alut2625	36, 711	136, 234	879	2875	10, 224	426	44.6	**35, 459**	1	51.5
alut2764	387	1252	34	0	0	0	0.0	**640**	1	0.0
diw0234	5349	20, 172	25	0	0	0	0.1	**1996**	1	0.1
diw0250	353	1216	11	0	0	0	0.0	**350**	1	0.0
diw0260	539	1970	12	0	0	0	0.0	**468**	1	0.0
diw0313	468	1644	14	0	0	0	0.0	**397**	1	0.0
diw0393	212	762	11	0	0	0	0.0	**302**	1	0.0
diw0445	1804	6622	33	21	60	12	0.1	**1363**	1	0.1
diw0459	3636	13, 578	25	14	40	5	0.1	**1362**	1	0.1
diw0460	339	1158	13	0	0	0	0.0	**345**	1	0.0
diw0473	2213	8270	25	0	0	0	0.1	**1098**	1	0.1
diw0487	2414	8772	25	0	0	0	0.0	**1424**	1	0.0
diw0495	938	3310	10	0	0	0	0.0	**616**	1	0.0
diw0513	918	3368	10	0	0	0	0.0	**604**	1	0.0
diw0523	1080	4030	10	0	0	0	0.0	**561**	1	0.0
diw0540	286	930	10	0	0	0	0.0	**374**	1	0.0
diw0559	3738	14, 026	18	171	608	12	0.2	**1570**	1	0.2
diw0778	7231	27, 454	24	0	0	0	0.6	**2173**	1	0.6
diw0779	11, 821	45, 032	50	32	100	8	2.7	**4440**	1	2.7
diw0795	3221	11, 876	10	0	0	0	0.1	**1550**	1	0.1
diw0801	3023	11, 150	10	0	0	0	0.1	**1587**	1	0.1
diw0819	10, 553	40, 132	32	0	0	0	0.2	**3399**	1	0.2
diw0820	11, 749	44, 768	37	88	310	12	3.8	**4167**	1	3.8
dmxa0296	233	772	12	0	0	0	0.0	**344**	1	0.0
dmxa0368	2050	7352	18	16	40	10	0.1	**1017**	1	0.1
dmxa0454	1848	6572	16	0	0	0	0.0	**914**	1	0.0
dmxa0628	169	560	10	0	0	0	0.0	**275**	1	0.0
dmxa0734	663	2308	11	0	0	0	0.0	**506**	1	0.0

Table 20 continued

Instance	Original			Presolved				Optimum	N	t [s]
	$\|V\|$	$\|A\|$	$\|T\|$	$\|V\|$	$\|A\|$	$\|T\|$	t [s]			
dmxa0848	499	1722	16	0	0	0	0.0	**594**	1	0.0
dmxa0903	632	2174	10	0	0	0	0.0	**580**	1	0.0
dmxa1010	3983	14, 216	23	0	0	0	0.1	**1488**	1	0.1
dmxa1109	343	1118	17	0	0	0	0.0	**454**	1	0.0
dmxa1200	770	2766	21	33	94	13	0.0	**750**	1	0.0
dmxa1304	298	1006	10	0	0	0	0.0	**311**	1	0.0
dmxa1516	720	2538	11	0	0	0	0.0	**508**	1	0.0
dmxa1721	1005	3462	18	0	0	0	0.0	**780**	1	0.0
dmxa1801	2333	8274	17	209	716	16	0.1	**1365**	1	0.1
gap1307	342	1104	17	0	0	0	0.0	**549**	1	0.0
gap1413	541	1812	10	0	0	0	0.0	**457**	1	0.0
gap1500	220	748	17	0	0	0	0.0	**254**	1	0.0
gap1810	429	1404	17	0	0	0	0.0	**482**	1	0.0
gap1904	735	2512	21	0	0	0	0.0	**763**	1	0.0
gap2007	2039	7096	17	0	0	0	0.0	**1104**	1	0.0
gap2119	1724	5950	29	0	0	0	0.0	**1244**	1	0.0
gap2740	1196	4168	14	0	0	0	0.0	**745**	1	0.0
gap2800	386	1306	12	0	0	0	0.0	**386**	1	0.0
gap2975	179	586	10	0	0	0	0.0	**245**	1	0.0
gap3036	346	1166	13	0	0	0	0.0	**457**	1	0.0
gap3100	921	3116	11	0	0	0	0.0	**640**	1	0.0
gap3128	10, 393	36, 086	104	0	0	0	0.2	**4292**	1	0.2
msm0580	338	1082	11	0	0	0	0.0	**467**	1	0.0
msm0654	1290	4540	10	0	0	0	0.0	**823**	1	0.0
msm0709	1442	4806	16	0	0	0	0.0	**884**	1	0.0
msm0920	752	2528	26	0	0	0	0.0	**806**	1	0.0
msm1008	402	1390	11	0	0	0	0.0	**494**	1	0.0
msm1234	933	3264	13	0	0	0	0.0	**550**	1	0.0
msm1477	1199	4156	31	0	0	0	0.0	**1068**	1	0.0
msm1707	278	956	11	0	0	0	0.0	**564**	1	0.0
msm1844	90	270	10	0	0	0	0.0	**188**	1	0.0
msm1931	875	3044	10	0	0	0	0.0	**604**	1	0.0
msm2000	898	3124	10	0	0	0	0.0	**594**	1	0.0
msm2152	2132	7404	37	0	0	0	0.1	**1590**	1	0.1
msm2326	418	1446	14	0	0	0	0.0	**399**	1	0.0
msm2492	4045	14, 188	12	0	0	0	0.1	**1459**	1	0.1
msm2525	3031	10, 478	12	0	0	0	0.1	**1290**	1	0.1

Table 20 continued

Instance	Original			Presolved				Optimum	N	t [s]
	$\|V\|$	$\|A\|$	$\|T\|$	$\|V\|$	$\|A\|$	$\|T\|$	t [s]			
msm2601	2961	10, 200	16	0	0	0	0.1	**1440**	1	0.1
msm2705	1359	4916	13	0	0	0	0.0	**714**	1	0.0
msm2802	1709	5926	18	0	0	0	0.0	**926**	1	0.0
msm2846	3263	11, 566	89	52	162	22	0.3	**3135**	1	0.3
msm3277	1704	5982	12	0	0	0	0.0	**869**	1	0.0
msm3676	957	3108	10	0	0	0	0.0	**607**	1	0.0
msm3727	4640	16, 510	21	0	0	0	0.1	**1376**	1	0.1
msm3829	4221	14, 510	12	0	0	0	0.3	**1571**	1	0.3
msm4038	237	780	11	0	0	0	0.0	**353**	1	0.0
msm4114	402	1380	16	0	0	0	0.0	**393**	1	0.0
msm4190	391	1332	16	0	0	0	0.0	**381**	1	0.0
msm4224	191	604	11	0	0	0	0.0	**311**	1	0.0
msm4312	5181	17, 786	10	672	2332	10	0.5	**2016**	1	0.5
msm4414	317	952	11	0	0	0	0.0	**408**	1	0.0
msm4515	777	2716	13	0	0	0	0.0	**630**	1	0.0
taq0014	6466	22, 092	128	0	0	0	0.5	**5326**	1	0.5
taq0023	572	1926	11	0	0	0	0.0	**621**	1	0.0
taq0365	4186	14, 148	22	61	198	9	0.1	**1914**	1	0.1
taq0377	6836	23, 430	136	58	160	34	1.6	**6393**	1	1.7
taq0431	1128	3810	13	0	0	0	0.0	**897**	1	0.0
taq0631	609	1864	10	0	0	0	0.0	**581**	1	0.0
taq0739	837	2876	16	0	0	0	0.0	**848**	1	0.0
taq0741	712	2434	16	53	170	9	0.0	**847**	1	0.0
taq0751	1051	3582	16	0	0	0	0.0	**939**	1	0.0
taq0891	331	1120	10	0	0	0	0.0	**319**	1	0.0
taq0903	6163	20, 980	130	0	0	0	1.4	**5099**	1	1.5
taq0910	310	1028	17	0	0	0	0.0	**370**	1	0.0
taq0920	122	388	17	0	0	0	0.0	**210**	1	0.0
taq0978	777	2478	10	0	0	0	0.0	**566**	1	0.0

Bold numbers signify a superior performance

Table 21 Detailed computational results for SPG, test-set WRP3

Instance	Original			Presolved				Optimum	N	t [s]
	$\|V\|$	$\|A\|$	$\|T\|$	$\|V\|$	$\|A\|$	$\|T\|$	t [s]			
wrp3-11	128	454	11	0	0	0	0.0	**1,100,361**	1	0.0
wrp3-12	84	298	12	0	0	0	0.0	**1,200,237**	1	0.0
wrp3-13	311	1226	13	131	492	13	0.1	**1,300,497**	1	0.1
wrp3-14	128	494	14	108	422	13	0.0	**1,400,250**	1	0.0
wrp3-15	138	514	15	0	0	0	0.0	**1,500,422**	1	0.0
wrp3-16	204	748	16	0	0	0	0.0	**1,600,208**	1	0.0
wrp3-17	177	708	17	151	622	13	0.0	**1,700,442**	1	0.0
wrp3-19	189	706	19	0	0	0	0.0	**1,900,439**	1	0.0
wrp3-20	245	908	20	0	0	0	0.0	**2,000,271**	1	0.0
wrp3-21	237	888	21	0	0	0	0.0	**2,100,522**	1	0.0
wrp3-22	233	862	22	186	686	20	0.0	**2,200,557**	1	0.1
wrp3-23	132	460	23	0	0	0	0.0	**2,300,245**	1	0.0
wrp3-24	262	974	24	120	416	17	0.0	**2,400,623**	1	0.0
wrp3-25	246	936	25	0	0	0	0.0	**2,500,540**	1	0.0
wrp3-26	402	1560	26	0	0	0	0.0	**2,600,484**	1	0.0
wrp3-27	370	1442	27	58	202	14	0.2	**2,700,502**	1	0.2
wrp3-28	307	1118	28	2	2	1	0.0	**2,800,379**	1	0.0
wrp3-29	245	872	29	0	0	0	0.0	**2,900,479**	1	0.0
wrp3-30	467	1792	30	73	248	14	0.1	**3,000,569**	1	0.1
wrp3-31	323	1184	31	55	188	16	0.1	**3,100,635**	1	0.1
wrp3-33	437	1676	33	100	382	13	0.0	**3,300,513**	1	0.0
wrp3-34	1244	4948	34	1057	4206	32	2.4	**3,400,646**	1	3.6
wrp3-36	435	1636	36	99	332	15	0.4	**3,600,610**	1	0.4
wrp3-37	1011	4020	37	847	3356	37	3.2	**3,700,485**	1	4.7
wrp3-38	603	2414	38	437	1780	37	0.9	**3,800,656**	1	2.1
wrp3-39	703	3232	39	609	2822	38	2.3	**3,900,450**	1	4.2
wrp3-41	178	614	41	129	448	36	0.2	**4,100,466**	1	0.2
wrp3-42	705	2746	42	572	2214	41	0.8	**4,200,598**	1	1.3
wrp3-43	173	596	43	0	0	0	0.1	**4,300,457**	1	0.1
wrp3-45	1414	5626	45	1204	4786	45	2.9	**4,500,860**	1	3.3
wrp3-48	925	3476	48	491	1816	45	1.1	**4,800,552**	1	2.0
wrp3-49	886	3600	49	693	2798	46	1.8	**4,900,882**	1	9.3
wrp3-50	1119	4502	50	915	3716	49	2.5	**5,000,673**	1	4.3
wrp3-52	701	2704	52	581	2250	49	1.6	**5,200,825**	1	5.2
wrp3-53	775	2942	53	148	534	12	0.3	**5,300,847**	1	0.3
wrp3-55	1645	6372	55	1487	5844	55	2.1	**5,500,888**	1	68.8

Table 21 continued

Instance	Original			Presolved				Optimum	N	t [s]
	$\|V\|$	$\|A\|$	$\|T\|$	$\|V\|$	$\|A\|$	$\|T\|$	t [s]			
wrp3-56	853	3180	56	590	2238	52	0.9	**5, 600, 872**	1	3.1
wrp3-60	838	3526	60	785	3300	60	2.2	**6, 001, 164**	1	29.9
wrp3-62	670	2632	62	586	2278	62	1.1	**6, 201, 016**	1	6.1
wrp3-64	1822	7220	64	1592	6402	59	3.4	**6, 400, 931**	1	9.7
wrp3-66	2521	9716	66	2269	8946	62	3.0	**6, 600, 922**	1	363.9
wrp3-67	987	3846	67	467	1848	36	1.8	**6, 700, 776**	1	3.7
wrp3-69	856	3242	69	447	1674	61	1.6	**6, 900, 841**	1	1.9
wrp3-70	1468	5862	70	964	3810	56	2.6	**7, 000, 890**	1	11.2
wrp3-71	1221	4828	71	947	3754	62	2.7	**7, 101, 028**	1	17.9
wrp3-73	1890	7226	73	1679	6534	63	2.2	**7, 301, 207**	1	36.9
wrp3-74	1019	3882	74	861	3326	65	1.1	**7, 400, 759**	1	13.1
wrp3-75	729	2790	75	551	2054	75	1.6	**7, 501, 020**	1	2.6
wrp3-76	1761	6740	76	1049	4066	46	3.1	**7, 601, 028**	1	4.6
wrp3-78	2346	9312	78	1993	7980	71	3.6	**7, 801, 094**	1	224.6
wrp3-79	833	3190	79	0	0	0	1.1	**7, 900, 444**	1	1.1
wrp3-80	1491	5662	80	1214	4650	75	3.6	**8, 000, 849**	1	34.3
wrp3-83	3168	12, 440	83	2961	11, 852	80	3.2	**8, 300, 906**	1	2873.0
wrp3-84	2356	9094	84	1915	7600	73	3.4	**8, 401, 094**	1	18.5
wrp3-85	528	2034	85	509	1958	85	0.5	**8, 500, 739**	1	5.6
wrp3-86	1360	5214	86	1157	4444	86	2.8	**86, 000, 746**	1	43.9
wrp3-88	743	2818	88	390	1470	58	1.6	**88, 001, 175**	1	2.3
wrp3-91	1343	5188	91	873	3356	78	3.1	**91, 000, 866**	1	5.3
wrp3-92	1765	7226	92	1265	5254	70	3.5	**92, 000, 764**	1	35.6
wrp3-94	1976	7672	94	1504	6002	79	3.9	**94, 001, 181**	5	54.5
wrp3-96	2518	9970	96	2193	8800	87	3.8	**96, 001, 172**	1	204.1
wrp3-98	2265	9090	98	1893	7712	83	3.9	**98, 001, 224**	1	303.9
wrp3-99	2076	8144	99	1689	6612	94	2.0	**99, 001, 097**	1	121.1

Bold numbers signify a superior performance

Table 22 Detailed computational results for SPG, test-set WRP4

Instance	Original			Presolved				Optimum	N	t [s]
	$\|V\|$	$\|A\|$	$\|T\|$	$\|V\|$	$\|A\|$	$\|T\|$	t [s]			
wrp4-11	123	466	11	0	0	0	0.0	**1,100,179**	1	0.0
wrp4-13	110	376	13	0	0	0	0.0	**1,300,798**	1	0.0
wrp4-14	145	566	14	0	0	0	0.0	**1,400,290**	1	0.0
wrp4-15	193	738	15	0	0	0	0.0	**1,500,405**	1	0.0
wrp4-16	311	1158	16	0	0	0	0.0	**1,601,190**	1	0.0
wrp4-17	223	808	17	138	486	13	0.0	**1,700,525**	1	0.0
wrp4-18	211	760	18	0	0	0	0.0	**1,801,464**	1	0.0
wrp4-19	119	412	19	0	0	0	0.0	**1,901,446**	1	0.0
wrp4-21	529	2064	21	167	644	15	0.1	**2,103,283**	1	0.1
wrp4-22	294	1136	22	108	392	15	0.1	**2,200,394**	1	0.1
wrp4-23	257	1030	23	131	478	18	0.0	**2,300,376**	1	0.0
wrp4-24	493	1926	24	0	0	0	0.1	**2,403,332**	1	0.1
wrp4-25	422	1616	25	92	332	9	0.1	**2,500,828**	1	0.1
wrp4-26	396	1562	26	310	1224	26	0.5	**2,600,443**	1	1.7
wrp4-27	243	994	27	71	260	16	0.1	**2,700,441**	1	0.1
wrp4-28	272	1090	28	190	756	28	0.2	**2,800,466**	1	0.5
wrp4-29	247	1010	29	105	394	22	0.2	**2,900,484**	1	0.2
wrp4-30	361	1448	30	296	1190	29	0.1	**3,000,526**	1	1.8
wrp4-31	390	1572	31	318	1280	30	0.3	**3,100,526**	1	2.2
wrp4-32	311	1264	32	246	998	29	0.1	**3,200,554**	1	1.3
wrp4-33	304	1142	33	103	372	19	0.0	**3,300,655**	1	0.0
wrp4-34	314	1300	34	45	154	9	0.1	**3,400,525**	1	0.1
wrp4-35	471	1908	35	320	1240	35	0.3	**3,500,601**	1	1.1
wrp4-36	363	1500	36	310	1276	36	0.2	**3,600,596**	1	1.1
wrp4-37	522	2108	37	438	1726	37	0.4	**3,700,647**	1	3.5
wrp4-38	294	1236	38	0	0	0	0.1	**3,800,606**	1	0.1
wrp4-39	802	3106	39	163	600	14	0.1	**3,903,734**	1	0.1
wrp4-40	538	2176	40	440	1774	39	0.3	**4,000,758**	1	6.2
wrp4-41	465	1910	41	377	1540	41	0.4	**4,100,695**	1	3.4
wrp4-42	552	2262	42	502	2038	42	0.4	**4,200,701**	1	9.3
wrp4-43	596	2296	43	277	1054	33	0.1	**4,301,508**	1	0.2
wrp4-44	398	1576	44	153	576	27	0.3	**4,401,504**	39	0.6
wrp4-45	388	1630	45	0	0	0	0.3	**4,500,728**	1	0.3
wrp4-46	632	2574	46	583	2356	46	0.4	**4,600,756**	1	8.6
wrp4-47	555	2196	47	0	0	0	0.9	**4,701,318**	1	0.9
wrp4-48	451	1650	48	0	0	0	0.1	**4,802,220**	1	0.1
wrp4-49	557	2160	49	158	582	22	0.5	**4,901,968**	1	0.6
wrp4-50	564	2224	50	223	860	24	0.4	**5,001,625**	1	0.6

Table 22 continued

Instance	Original			Presolved				Optimum	N	t [s]
	$\|V\|$	$\|A\|$	$\|T\|$	$\|V\|$	$\|A\|$	$\|T\|$	t [s]			
wrp4-51	668	2612	51	407	1592	45	1.3	**5,101,616**	1	1.6
wrp4-52	547	2230	52	70	240	20	0.4	**5,201,081**	1	0.4
wrp4-53	615	2464	53	351	1370	46	0.7	**5,301,351**	1	1.4
wrp4-54	688	2776	54	356	1398	40	0.6	**5,401,534**	1	1.4
wrp4-55	610	2402	55	403	1562	51	0.7	**5,501,952**	1	1.0
wrp4-56	839	3234	56	489	1902	47	0.8	**5,602,299**	1	1.5
wrp4-58	757	2986	58	367	1446	41	0.6	**5,801,466**	1	1.5
wrp4-59	904	3612	59	154	506	29	0.2	**5,901,592**	1	0.2
wrp4-60	693	2740	60	103	346	24	0.4	**6,001,782**	1	0.4
wrp4-61	775	3076	61	138	500	19	0.2	**6,102,210**	1	0.2
wrp4-62	1283	4986	62	313	1184	29	2.6	**6,202,100**	1	2.7
wrp4-63	1121	4454	63	943	3752	60	0.9	**6,301,479**	1	59.9
wrp4-64	632	2562	64	0	0	0	0.3	**6,401,996**	1	0.3
wrp4-66	844	3382	66	229	834	24	1.0	**6,602,931**	1	1.0
wrp4-67	1518	6120	67	208	770	28	2.5	**6,702,800**	1	2.6
wrp4-68	917	3700	68	793	3182	67	0.8	**6,801,753**	1	3.7
wrp4-69	574	2330	69	0	0	0	0.7	**6,902,328**	1	0.7
wrp4-70	637	2538	70	0	0	0	0.1	**7,003,022**	1	0.1
wrp4-71	802	3218	71	0	0	0	0.1	**7,102,320**	1	0.1
wrp4-72	1151	4548	72	538	2132	48	1.1	**7,202,807**	1	4.2
wrp4-73	1898	7232	73	1290	5112	73	1.9	**7,302,643**	1	27.7
wrp4-74	802	3240	74	610	2422	72	0.8	**7,402,046**	1	1.9
wrp4-75	938	3738	75	702	2784	75	1.1	**7,501,712**	1	2.0
wrp4-76	766	3070	76	140	504	30	0.5	**7,602,040**	1	0.6

Bold numbers signify a superior performance

References

1. Achterberg, T.: Conflict analysis in mixed integer programming. Discrete Optim. **4**(1), 4–20 (2007). https://doi.org/10.1016/j.disopt.2006.10.006
2. Achterberg, T.: Constraint Integer Programming. Ph.D. thesis, Technische Universität Berlin (2007)
3. de Aragão, M.P., Werneck, R.F.: 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, pp. 1–15. Springer (2002). https://doi.org/10.1007/3-540-45643-0_1
4. Bonnet, É., Sikora, F.: The PACE 2018 parameterized algorithms and computational experiments challenge: the third iteration. In: Paul, C., Pilipczuk, M. (eds.) 13th International Symposium on Parameterized and Exact Computation (IPEC 2018), Leibniz International Proceedings in Informatics (LIPIcs), vol. 115, pp. 26:1–26:15. Schloss Dagstuhl–Leibniz–Zentrum fuer Informatik, Dagstuhl, Germany (2019). https://doi.org/10.4230/LIPIcs.IPEC.2018.26
5. Byrka, J., Grandoni, F., Rothvoß, T., Sanità, L.: Steiner tree approximation via iterative randomized rounding. J. ACM **60**(1), 6 (2013). https://doi.org/10.1145/2432622.2432628
6. Cheng, X., Du, D.Z.: Steiner Trees in Industry, vol. 11. Springer, Berlin (2004). https://doi.org/10.1007/0-387-23830-1_4
7. DIMACS: 11th DIMACS Challenge. http://dimacs11.zib.de/ (2015). Accessed 10 Jan 2020
8. Dreyfus, S.E., Wagner, R.A.: The Steiner problem in graphs. Networks **1**(3), 195–207 (1971). https://doi.org/10.1002/net.3230010302
9. Duin, C.: Steiner Problems in Graphs. Ph.D. thesis, University of Amsterdam (1993)
10. Duin, C.: Preprocessing the Steiner Problem in Graphs. Springer US, Boston (2000). https://doi.org/10.1007/978-1-4757-3171-2_10
11. Duin, C., Volgenant, A.: An edge elimination test for the Steiner problem in graphs. Oper. Res. Lett. **8**(2), 79–83 (1989). https://doi.org/10.1016/0167-6377(89)90005-9
12. Duin, C.W., Volgenant, A.: Reduction tests for the Steiner problem in graphs. Networks **19**(5), 549–567 (1989). https://doi.org/10.1002/net.3230190506
13. Fischetti, M., Leitner, M., Ljubić, I., Luipersbeck, M., Monaci, M., Resch, M., Salvagnin, D., Sinnl, M.: Thinning out Steiner trees: a node-based model for uniform edge costs. Math. Program. Comput. **9**(2), 203–229 (2017). https://doi.org/10.1007/s12532-016-0111-0
14. Gamrath, G., Koch, T., Maher, S., Rehfeldt, D., Shinano, Y.: SCIP-Jack—a solver for STP and variants with parallelization extensions. Math. Program. Comput. **9**(2), 231–296 (2017). https://doi.org/10.1007/s12532-016-0114-x
15. Goemans, M.X., Olver, N., Rothvoß, T., Zenklusen, R.: Matroids and integrality gaps for hypergraphic Steiner tree relaxations. In: Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing, STOC '12, pp. 1161–1176. Association for Computing Machinery, New York, NY, USA (2012). https://doi.org/10.1145/2213977.2214081
16. Hegde, C., Indyk, P., Schmidt, L.: A fast, adaptive variant of the Goemans–Williamson scheme for the prize-collecting Steiner tree problem. In: Workshop of the 11th DIMACS Implementation Challenge. Workshop of the 11th DIMACS Implementation Challenge (2014)
17. Hougardy, S., Silvanus, J., Vygen, J.: Dijkstra meets Steiner: a fast exact goal-oriented Steiner tree algorithm. Math. Program. Comput. **9**(2), 135–202 (2017). https://doi.org/10.1007/s12532-016-0110-1
18. Hušek, R., Knop, D., Masařík, T.: Approximation Algorithms for Steiner Tree Based on Star Contractions: A Unified View. arXiv preprint arXiv:2002.03583 (2020)
19. Hwang, F., Richards, D., Winter, P.: The Steiner Tree Problem. Annals of Discrete Mathematics. Elsevier Science, Amsterdam (1992)
20. IBM: Cplex. https://www.ibm.com/analytics/cplex-optimizer (2020)
21. Iwata, Y., Shigemura, T.: Separator-based pruned dynamic programming for Steiner tree. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, pp. 1520–1527 (2019). https://doi.org/10.1609/aaai.v33i01.33011520
22. Juhl, D., Warme, D.M., Winter, P., Zachariasen, M.: The GeoSteiner software package for computing Steiner trees in the plane: an updated computational study. Math. Program. Comput. **10**(4), 487–532 (2018). https://doi.org/10.1007/s12532-018-0135-8
23. Karp, R.: Reducibility among combinatorial problems. In: Miller, R., Thatcher, J. (eds.) Complexity of Computer Computations, pp. 85–103. Plenum Press, New York (1972). https://doi.org/10.1007/978-1-4684-2001-2_9

24. Kisfaludi-Bak, S., Nederlof, J., Leeuwen, E.J.V.: Nearly ETH-tight algorithms for planar Steiner tree with terminals on few faces. ACM Trans. Algorithms (TALG) **16**(3), 1–30 (2020). https://doi.org/10.1145/3371389
25. Koch, T., Martin, A.: Solving Steiner tree problems in graphs to optimality. Networks **32**, 207–232 (1998). https://doi.org/10.1002/(SICI)1097-0037(199810)32:3%3C207::AID-NET5%3E3.0.CO;2-O
26. Koch, T., Martin, A., Voß, S.: SteinLib: An updated library on Steiner tree problems in graphs. In: Du, D.Z., Cheng, X. (eds.) Steiner Trees in Industries, pp. 285–325. Kluwer, Alphen aan den Rijn (2001)
27. Leitner, M., Ljubic, I., Luipersbeck, M., Prossegger, M., Resch, M.: New Real-World Instances for the Steiner Tree Problem in Graphs. Tech. rep, ISOR, Uni Wien (2014)
28. Müller, B., Serrano, F., Gleixner, A.: Using two-dimensional projections for stronger separation and propagation of bilinear terms. SIAM J. Optim. **30**(2), 1339–1365 (2020). https://doi.org/10.1137/19M1249825
29. Nederlof, J.: Fast polynomial-space algorithms using Möbius inversion: improving on Steiner tree and related problems. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) International Colloquium on Automata, Languages, and Programming, pp. 713–725. Springer (2009). https://doi.org/10.1007/978-3-642-02927-1_59
30. Pajor, T., Uchoa, E., Werneck, R.F.: A robust and scalable algorithm for the Steiner problem in graphs. Math. Program. Comput. (2017). https://doi.org/10.1007/s12532-017-0123-4
31. Polzin, T.: Algorithms for the Steiner Problem in Networks. Ph.D. thesis, Saarland University (2003)
32. Polzin, T., Daneshmand, S.V.: A comparison of Steiner tree relaxations. Discrete Appl. Math. **112**(1–3), 241–261 (2001). https://doi.org/10.1016/s0166-218x(00)00318-8
33. Polzin, T., Daneshmand, S.V.: Improved algorithms for the Steiner problem in networks. Discrete Appl. Math. **112**(1–3), 263–300 (2001). https://doi.org/10.1016/S0166-218X(00)00319-X
34. Polzin, T., Daneshmand, S.V.: Extending Reduction Techniques for the Steiner Tree Problem, pp. 795–807. Springer, Berlin (2002). https://doi.org/10.1007/3-540-45749-6_69
35. Polzin, T., Daneshmand, S.V.: On Steiner trees and minimum spanning trees in hypergraphs. Oper. Res. Lett. **31**(1), 12–20 (2003). https://doi.org/10.1016/S0167-6377(02)00185-2
36. Polzin, T., Daneshmand, S.V.: Practical partitioning-based methods for the Steiner problem. In: Àlvarez, C.., Serna, M.. (eds.) Experimental Algorithms, pp. 247–252. Springer, Berlin (2006). https://doi.org/10.1007/11764298_22
37. Polzin, T., Vahdati-Daneshmand, S.: The Steiner Tree Challenge: An updated Study (2014). Unpublished manuscript at http://dimacs11.cs.princeton.edu/downloads.html
38. Rehfeldt, D., Koch, T.: Implications, conflicts, and reductions for Steiner trees. Tech. Rep. 20-28, ZIB, Takustr. 7, 14195 Berlin (2020)
39. Rehfeldt, D., Koch, T.: On the exact solution of prize-collecting Steiner tree problems. Tech. Rep. 20-11, ZIB, Takustr. 7, 14195 Berlin (2020)
40. Reinelt, G.: TSPLIB—a traveling salesman problem library. ORSA J. Comput. **3**(4), 376–384 (1991). https://doi.org/10.1287/ijoc.3.4.376
41. Rosseti, I., de Aragão, M.P., Ribeiro, C.C., Uchoa, E., Werneck, R.F.: New Benchmark Instances for The Steiner Problem in Graphs. Springer US, Boston (2004). https://doi.org/10.1007/978-1-4757-4137-7_28
42. Savelsbergh, M.W.P.: Preprocessing and probing techniques for mixed integer programming problems. ORSA J. Comput. **6**(4), 445–454 (1994). https://doi.org/10.1287/ijoc.6.4.445
43. Takahashi, H., Matsuyama, A.: An approximate solution for the Steiner problem in graphs. Mathematica Japonicae **24**, 573–577 (1980)
44. Uchoa, E.: Reduction tests for the prize-collecting Steiner problem. Oper. Res. Lett. **34**(4), 437–444 (2006). https://doi.org/10.1016/j.orl.2005.02.007
45. Uchoa, E., Poggi de Aragão, M., Ribeiro, C.C.: Preprocessing Steiner problems from VLSI layout. Networks **40**(1), 38–50 (2002). https://doi.org/10.1002/net.10035
46. Vahdati Daneshmand, S.: Algorithmic Approaches to the Steiner Problem in Networks. Ph.D. thesis, Universität Mannheim (2004)
47. Vygen, J.: Faster algorithm for optimum Steiner trees. Inf. Process. Lett. **111**(21), 1075–1079 (2011). https://doi.org/10.1016/j.ipl.2011.08.005
48. Winter, P.: Reductions for the rectilinear Steiner tree problem. Networks **26**(4), 187–198 (1995). https://doi.org/10.1002/net.3230260404
49. Witzig, J., Gleixner, A.: Conflict-driven heuristics for mixed integer programming. INFORMS J. Comput. (2020). https://doi.org/10.1287/ijoc.2020.0973. (**Epub ahead of print**)

50. Wong, R.: A dual ascent approach for Steiner tree problems on a directed graph. Math. Program. **28**, 271–287 (1984). https://doi.org/10.1007/BF02612335