Zuse Institute Berlin

Takustr. 7
14195 Berlin
Germany

DANIEL REHFELDT, THORSTEN KOCH

Reduction-based exact solution of prize-collecting Steiner tree problems

ZIB Report 18-55 (November 2018)

Zuse Institute Berlin
Takustr. 7
D-14195 Berlin

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

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

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

Reduction-based exact solution of prize-collecting Steiner tree problems

Daniel Rehfeldt*† · Thorsten Koch

Abstract

The prize-collecting Steiner tree problem (PCSTP) is a well-known generalization of the classical Steiner tree problem in graphs, with a large number of practical applications. It attracted particular interest during the latest (11th) DIMACS Challenge and since then a number of PCSTP solvers have been introduced in the literature, some of which drastically improved on the best results achieved at the Challenge. The following article aims to further advance the state of the art. It introduces new techniques and algorithms for PCSTP, involving various forms of reductions of PCSTP instances to equivalent problems—which for example allows to decrease the problem size or to obtain a better IP formulation. Several of the new techniques and algorithms provably dominate previous approaches. Further theoretical properties of the new components, such as their complexity, are discussed, and their profound interaction is described. Finally, the new developments also translate into a strong computational performance: the resulting exact solver outperforms all previous approaches—both in terms of run-time and solvability—and can solve formerly intractable benchmark instances from the 11th DIMACS Challenge to optimality.

1 Introduction

The Steiner tree problem in graphs (SPG) is one of the fundamental ($\mathcal{NP}$-hard) combinatorial optimization problems [19]. A well-known generalization is the prize-collecting Steiner tree problem (PCSTP), stated as follows: Given an undirected graph $G = (V, E)$, edge-weights $c : E \to \mathbb{Q}_{>0}$, and node-weights (or *prizes*) $p : V \to \mathbb{Q}_{\geq 0}$, a tree $S = (V(S), E(S)) \subseteq G$ is required such that

$$C(S) := \sum_{e \in E(S)} c(e) + \sum_{v \in V \setminus V(S)} p(v) \tag{1}$$

is minimized. By setting sufficiently high node weights for its terminals, each SPG instance can be reduced to a PCSTP. However, while the number of real-world applications of the classical Steiner tree problem in graphs is limited [13], the PCSTP entails many practical applications, which can be found in various areas, for instance in the design of telecommunication networks [24], electricity planning [6], computational biology [17], or geophysics [31].

The PCSTP has been extensively discussed in the literature, see e.g. [5, 7, 18, 23]. Moreover, many exact and heuristic solving approaches have been suggested. The problem attracted particular interest in the wake of the 11th DIMACS Challenge [1] in December 2014—dedicated to Steiner tree problems—where the PCSTP categories could boast the most participants by far. Furthermore, in the last two years a number of solvers for the PCSTP have been described

*TU Berlin, 10623 Berlin, Germany
†Zuse Institute Berlin, 14195 Berlin, Germany, koch@zib.de, rehfeldt@zib.de

in the literature [3, 11, 12, 13, 22, 25, 32], some of which, in particular [22], could drastically improve on the best results achieved at the DIMACS Challenge—being able to not only solve many instances orders of magnitude faster, but also to solve a number of instances for the first time to optimality. Exact approaches for PCSTP are usually based on branch-and-bound or branch-and-cut [11, 13], include specialized (primal and sometimes dual) heuristics [20, 22], and make use of various preprocessing methods to reduce the problem size [23, 29].

This article introduces new techniques and algorithms for solving PCSTP, most of which are based on, or result in reductions of the PCSTP to equivalent problems—these problems can be PCSTPs itself, but can also be from different problem classes. The reductions can for example decrease the problem size or allow to obtain a stronger IP formulation. Moreover, several of the new methods provably dominate previous approaches. While some of the techniques require to solve NP-hard subproblems (not yet described in the literature), the underlying concepts allow to design empirically efficient heuristics. Practically also the integration of the new methods into an exact solver will be described and computational experiments on a large number of benchmark instances will be presented—along with a comparison with a state-of-the-art solver. What sets the new techniques apart from other approaches for combinatorial optimization problems is the deep and intricate interaction of the individual components combined with their wide applicability within a branch-and-cut framework—from preprocessing and probing to IP formulation and separation to heuristics, domain propagation and branching. While interaction between solution techniques and multiple usability has been described in state-of-the-art solvers for problems such as the traveling salesman problem [4] or (even more pronounced) the SPG [26], we are not aware of any other approach for which such a broad impact on the overall solving procedure and profound interrelation can be achieved by such a, comparatively, small number of techniques. We would also like to point out that the newly developed software has been integrated into the academic Steiner tree framework SCIP-Jack [13] and will be made publicly available as part of its next major release.

Finally, while it will be shown that one set of methods is also directly applicable for the SPG, one can furthermore extend several of the presented techniques and algorithms to related combinatorial optimization problems such as the node-weighted Steiner tree, or the maximum-weight connected subgraph problem.

1.1 Preliminaries and notation

Throughout this article it will be presupposed that a PCSTP instance $I_{PC} = (V, E, c, p)$ is given such that (V, E) is connected; otherwise one can optimize each connected component separately. For a graph G we denote its vertices by $V(G)$ and its edges by $E(G)$; similarly, for a finite walk W we denote the set of vertices and the set of edges it contains by $V(W)$ and $E(W)$. We call $T_p := \{t_1, t_2, ..., t_s\} := \{v \in V \mid p(v) > 0\}$ the set of *potential terminals* [22].

By $d(v_i, v_j)$ we denote the distance of a shortest path (with respect to c) between vertices $v_i, v_j \in V$. Similarly, $\underline{d}(v_i, v_j)$ is defined as the distance between v_i and v_j in the graph induced by $V \setminus (T_p \setminus \{v_i, v_j\})$ [29]. To each vertex $v_i \in V \setminus T$, the k $\underline{d}$-nearest potential terminals will be denoted by $\underline{v}_{i,1}, \underline{v}_{i,2}, ..., \underline{v}_{i,k}$ (ties are broken arbitrarily). Moreover, for any function $x : M \mapsto \mathbb{Q}$ with M finite, and any $M' \subseteq M$ define $x(M') := \sum_{i \in M'} x(i)$. For $U \subseteq V$ define $\delta(U) := \{\{u, v\} \in E \mid u \in U, v \in V \setminus U\}$; for a directed graph $D = (V, A)$ define $\delta^+(U) := \{(u, v) \in A \mid u \in U, v \in V \setminus U\}$ and $\delta^-(U) := \delta^+(V \setminus U)$. We also write δ_G or δ_D^+, δ_D^- to distinguish the underlying graph. For an IP formulation F we denote the optimal objective value and the set of feasible points of its LP relaxation by $v_{LP}(F)$ and $\mathcal{P}_{LP}(F)$, respectively.

2 Reductions within the problem class

The reductions described in the following aim to reduce a given instance to a smaller one of the same problem class. Several articles have addressed such techniques for the PCSTP, e.g. [22, 23, 29, 33], but most are dominated by the methods described in the following. The new methods will not only be employed for classical preprocessing, but also throughout the entire solving process, e.g. for domain propagation or within heuristics.

2.1 Taking short walks

The following approach uses a new, walk-based, distance function. It generalizes the bottleneck distance concept that was the central theme of [33]. Let $v_i, v_j \in V$. A finite walk $W = (v_{i_1}, e_{i_1}, v_{i_2}, e_{i_2}, ..., e_{i_r}, v_{i_r})$ with $v_{i_1} = v_i$ and $v_{i_r} = v_j$ will be called *prize-constrained* (v_i, v_j)*-walk* if no $v \in T_p \cup \{v_i, v_j\}$ is contained more than once in W. For any $k, l \in \mathbb{N}$ with $1 \leq k \leq l \leq r$ define the subwalk $W(v_{i_k}, v_{i_l}) := (v_{i_k}, e_{i_k}, v_{i_{k+1}}, e_{i_{k+1}}, ..., e_{i_l}, v_{i_l})$; note that $W(v_{i_k}, v_{i_l})$ is again a prize-constrained walk. Furthermore, define the *prize-collecting cost* of W as

$$c_{pc}(W) := \sum_{e \in E(W)} c(e) - \sum_{v \in V(W) \setminus \{v_i, v_j\}} p(v). \tag{2}$$

Thereupon, define the *prize-constrained length* of W as

$$l_{pc}(W) := \max\{c_{pc}(W(v_{i_k}, v_{i_l})) \mid 1 \leq k \leq l \leq r,\ v_{i_k}, v_{i_l} \in T_p \cup \{v_i, v_j\}\}. \tag{3}$$

Intuitively, $l_{pc}(W)$ provides the cost of the least profitable subwalk of W. This measure will in the following be useful to bound the cost of connecting two disjoint trees that contain the first and the last vertex of W, respectively. Finally, we denote the set of all prize-constrained (v_i, v_j)-walks by $\mathcal{W}_{pc}(v_i, v_j)$ and define the *prize-constrained distance* between v_i and v_j as

$$d_{pc}(v_i, v_j) := \min\{l_{pc}(W') \mid W' \in \mathcal{W}_{pc}(v_i, v_j)\}. \tag{4}$$

Note that $d_{pc}(v_i, v_j) = d_{pc}(v_j, v_i)$ for any $v_i, v_j \in V$. By using the prize-constrained distance one can formulate a reduction criterion that dominates the *special distance* test from [33]—it identifies (and allows to delete) all edges found by [33] and can (and usually does) identify further ones:

Proposition 1. *Let* $\{v_i, v_j\} \in E$. *If*

$$c(\{v_i, v_j\}) > d_{pc}(v_i, v_j) \tag{5}$$

is satisfied, then $\{v_i, v_j\}$ *cannot be contained in any optimal solution.*

Proof. Let S be a tree with $\{v_i, v_j\} \in E(S)$. Further, let $W = (v_{i_1}, e_{i_1}, ..., e_{i_r}, v_{i_r})$ be a prize-constrained (v_i, v_j)-walk with $l_{pc}(W) = d_{pc}(v_i, v_j)$. Remove $\{v_i, v_j\}$ from S to obtain two new trees. Of these two trees denote the one that contains v_i by S_i and the other (containing v_j) by S_j. Define $b := \min\{k \in \{1, ..., r\} \mid v_{i_k} \in V(S_j)\}$ and $a := \max\{k \in \{1, ..., b\} \mid v_{i_k} \in V(S_i)\}$. Further, define $x := \max\{k \in \{1, ..., a\} \mid v_{i_k} \in T_p \cup \{v_i\}\}$ and $y := \min\{k \in \{b, ..., r\} \mid v_{i_k} \in T_p \cup \{v_j\}\}$. By definition, $x \leq a < b \leq y$ and furthermore:

$$c_{pc}(W(v_{i_a}, v_{i_b})) \leq c_{pc}(W(v_{i_x}, v_{i_y})). \tag{6}$$

Reconnect S_i and S_j by $W(v_{i_a}, v_{i_b})$), which yields a new connected subgraph S'. If S' is not a tree, make it one by removing redundant edges, without removing any node (which can only decrease $C(S')$). For this tree it holds that:

$$\begin{aligned} C(S') &\leq C(S) + c_{pc}(W(v_{i_a}, v_{i_b})) - c(\{v_i, v_j\}) \\ &\overset{(6)}{\leq} C(S) + c_{pc}(W(v_{i_x}, v_{i_y})) - c(\{v_i, v_j\}) \\ &\leq C(S) + l_{pc}(W) - c(\{v_i, v_j\}) \\ &= C(S) + d_{pc}(v_i, v_j) - c(\{v_i, v_j\}) \\ &\overset{(5)}{<} C(S). \end{aligned}$$

Because of $C(S') < C(S)$ no optimal solution can contain $\{v_i, v_j\}$. □

Since computing the Steiner bottleneck distance is already $\mathcal{NP}$-hard [33], it does not come as a surprise that the same holds for d_{pc} (which can be shown in the same way). However, the definition of d_{pc} allows to design a simple algorithm for finding upper bounds that yields empirically strong results. The method is an extension of Dijkstra's algorithm [8] (with priority queue), and is run from both endpoints of an edge $e = \{v_i, v_j\}$ that one attempts to delete. We sketch the procedure for endpoint v_i: Let $dist(v)$ be the distance value of Dijkstra's algorithm for each $v \in V$, initialized with $dist(v) := \infty$ for all $v \in V \setminus \{v_i\}$ and $dist(v_i) := 0$. Start Dijkstra's algorithm from v_i, but apply the following modifications: First, do not update vertex v_l from vertex v_k if $dist(v_k) + c(\{v_k, v_l\}) \geq c(e)$. Second, for $t \in T_p \setminus \{v_i\}$ set $dist(t)$ (as computed by Dijkstra) to $\max\{dist(t) - p(t), 0\}$ before inserting t into the priority queue or when updating t (if it already is in the priority queue). Third, all $v \in V \setminus (T_p \cup \{v_i, v_j\})$ can be reinserted into the priority queue after they have been removed. Fourth, exit if v_j is visited. If $dist(v_j) < c(e)$, we can remove e. Bounds on the number of visited edges are set to limit the run time. Note that both Proposition 1 and the heuristic can be extended to the case of equality.

By using the prize-constrained distance, further reduction tests from the literature can be strengthened, such as *Non-Terminal of Degree k* [33]; a related walk-based concept introduced in Section 3 allows to strengthen additional methods such as *Nearest Vertex* [29], or *Short Links* [29].

2.2 Using bounds

Bound-based reductions techniques identify edges and vertices for elimination by examining whether they induce a lower bound that exceeds a given upper bound [10, 26]. For the subsequent reduction techniques of this type we introduce the following concept: a *terminal-regions decomposition* of I_{PC} is a partition $H = \{H_t \subseteq V \mid T_p \cap H_t = \{t\}\}$ of V such that for each $t \in T_p$ the subgraph induced by H_t is connected. Define for all $t \in T_p$

$$r_H^{pc}(t) := \min\{p(t), \min\{d(t, v) \mid v \notin H_t\}\}. \tag{7}$$

The terminal-regions decomposition concept generalizes the Voronoi decomposition for the PC-STP introduced in [29] and can easily be extended to SPG by using $\min\{d(t, v) \mid v \notin H_t\}$ instead of $r_H^{pc}(t)$—which generalizes the SPG Voronoi concept [26]. In [28] we introduced a related concept for the maximum-weight connected subgraph problem. For ease of presentation assume $r_H^{pc}(t_i) \leq r_H^{pc}(t_j)$ for $1 \leq i < j \leq s$.

Proposition 2. *Let $v_i \in V \setminus T_p$. If there is an optimal solution S such that $v_i \in V(S)$, then a lower bound on $C(S)$ is defined by*

$$\underline{d}(v_i, \underline{v}_{i,1}) + \underline{d}(\underline{v}_i, \underline{v}_{i,2}) + \sum_{k=1}^{s-2} r_H^{pc}(t_k). \tag{8}$$

The proposition can be proven similarly to the Voronoi decomposition results [29], see Appendix B.1. Each vertex $v_i \in V \setminus T_p$ with the property that the affiliated lower bound (8) exceeds a known upper bound can be eliminated. Moreover, if a solution S corresponding to the upper bound is given and $v_i \notin V(S)$, one can also eliminate v_i if the lower bound stated in Proposition 2 is equal to $C(S)$. A result similar to Proposition 2 can be formulated for edges of an optimal solution. Finally, the following proposition allows to *pseudo-eliminate* [10] vertices, i.e., to delete a vertex and connect all its adjacent vertices by new edges.

Proposition 3. *Let $v_i \in V \setminus T_p$. If there is an optimal solution S such that $\delta_S(v_i) \geq 3$, then a lower bound on $C(S)$ is defined by*

$$\underline{d}(v_i, \underline{v}_{i,1}) + \underline{d}(v_i, \underline{v}_{i,2}) + \underline{d}(v_i, \underline{v}_{i,3}) + \sum_{k=1}^{s-3} r_H^{pc}(t_k). \tag{9}$$

To efficiently apply Proposition 2, one would like to maximize (8)—and for Proposition 3 to maximize (9). However, as shown in Appendix B.2, one obtains

Proposition 4. *Given a $v_i \in V \setminus T_p$, finding a terminal-regions decomposition that maximizes* (8) *is $\mathcal{NP}$-hard. The same holds for* (9)*.*

Thus, to compute a terminal-regions decomposition a heuristic approach will be used; still, the flexibility of the terminal regions concept allows to design a heuristic that computes bounds that are at least as good as those provided by the Voronoi decomposition method [29], and empirically often better—allowing for significantly stronger graph reductions.

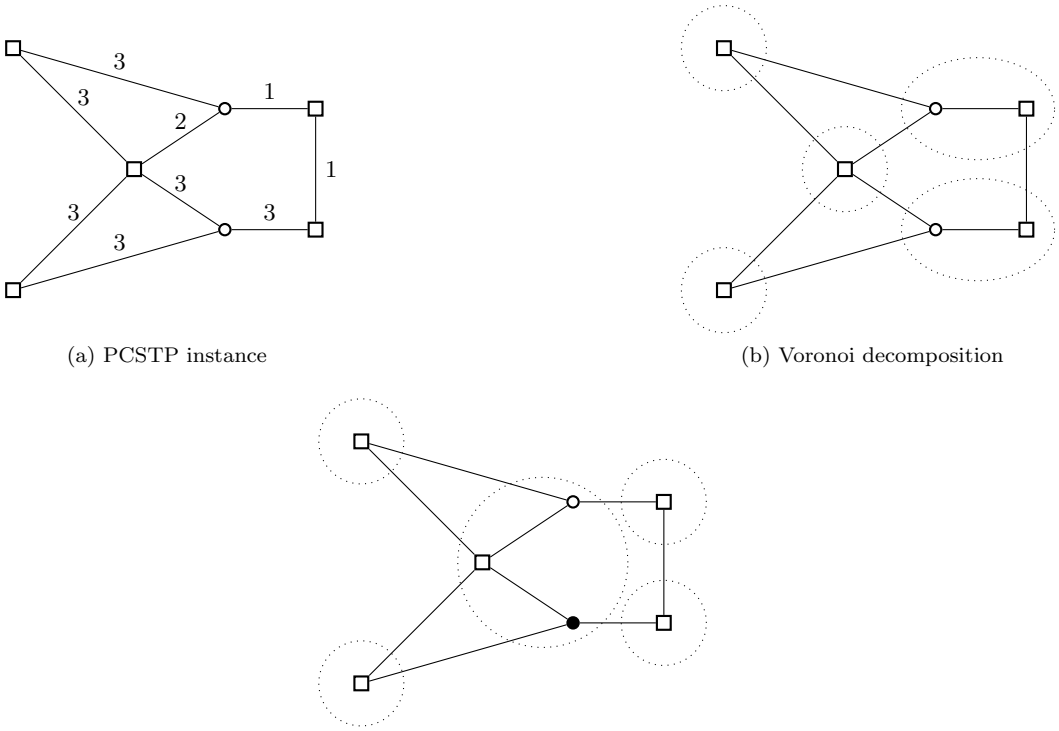
Figure 1 depicts a PCSTP, a corresponding Voronoi decomposition as described in [29]—which is in fact a particular terminal-regions decomposition—and an alternative terminal-regions decomposition, found by our heuristic. The second terminal-regions decomposition yields a stronger lower bound than the Voronoi decomposition. For the filled vertex the lower bounds are 10 and 11, respectively (10 is also the optimal solution value for the instance).

3 Changing the problem class

A cornerstone of the reduction approach described in this section is the *Steiner arborescence problem* (*SAP*), which is defined as follows: Given a directed graph $D = (V, A)$, costs $c : A \to \mathbb{Q}_{\geq 0}$, a set $T \subseteq V$ of *terminals* and a root $r \in T$, a directed tree (arborescence) $S \subseteq D$ of minimum cost $\sum_{a \in A(S)} c(a)$ is required such that for all $t \in T$ the tree S contains a directed path from r to t. Associating with each $a \in A$ a variable $x(a)$ that indicates whether a is contained in a solution ($x(a) = 1$) or not ($x(a) = 0$), one can state the well-known *directed cut* (*DCut*) formulation [34] for an SAP (V, A, T, c, r):

Formulation 1. *Directed cut (DCut)*

$$\begin{aligned}
&\min \quad c^T x && && (10)\\
&x(\delta^-(U)) \;\geq\; 1 && \text{for all } U \subset V,\ r \notin U, U \cap T \neq \emptyset, && (11)\\
&x(a) \;\in\; \{0,1\} && \text{for all } a \in A. && (12)
\end{aligned}$$



(a) PCSTP instance

(b) Voronoi decomposition

(c) terminal-regions decomposition

Figure 1: Illustration of a PCSTP instance (a), a Voronoi decomposition (b), and a second terminal-regions decomposition (c). Potential terminals are drawn as squares. All potential terminals have a prize of 5. If an upper bound less than 11 is known, the filled vertex in (c) can be deleted by means of the terminal-regions decomposition depicted in (c), but not by means of the Voronoi decomposition, unless also an optimal solution tree is given.

In [34] also a dual-ascent algorithm for *DCut* was introduced that allows to quickly compute empirically strong lower bounds. Dual-ascent is one of the reasons it has proven advantageous to transform undirected problems such as the SPG to SAP [21, 26] and use *DCut*, but furthermore such transformations can provide stronger LP relaxations, both theoretically and practically [10, 16]. For the PCSTP such a transformation can be stated as follows [27]:

Transformation 1 (PCSTP to SAP)**.**
Input: *PCSTP* (V, E, c, p)
Output: *SAP* (V', A', T', c', r')

1. *Set* $V' := V$, $A' := \{(v, w) \in V' \times V' \mid \{v, w\} \in E\}$, *and* $c' : A' \to \mathbb{Q}_{\geq 0}$ *with* $c'(a) := c(\{v, w\})$ *for* $a = (v, w) \in A'$; *define* $M := \sum_{t \in T_p} p(t)$.
2. *Add vertices* r' *and* v'_0 *to* V'.
3. *For each* $i \in \{1, ..., s\}$:
   (a) *add arc* (r', t_i) *of weight* M *to* A';
   (b) *add node* t'_i *to* V';

(c) add arcs (t_i, v'_0) *and* (t_i, t'_i) *to* A'*, both being of weight* 0*;*

(d) add arc (v'_0, t'_i) *of weight* $p(t_i)$ *to* A'*.*

4. *Define set of terminals* $T' := \{t'_1, ..., t'_s\} \cup \{r'\}$.

5. ***Return*** (V', A', T', c', r').

The underlying idea of the transformation is to add a new terminal t'_i for each original potential terminal t_i and provide additional arcs that allow to connect t'_i from any original potential terminal t_j with cost $p(t_j)$. Note that one can use a, similar, simplified transformation if one adds an additional constraint to the resulting SAP [13]. However, besides providing a "pure" SAP, Transformation 1 allows to directly apply dual-ascent. The following results still hold if Transformation 1 is replaced by the simpler version.

Each optimal solution to the SAP obtained from Transformation 1 can be transformed to an optimal solution to the original PCSTP. For $I_{PC} = (V, E, c, p)$ one can therefore define the following formulation, which uses the SAP (V', A', T', c', r') obtained from applying Transformation 1 on I_{PC}:

Formulation 2. *Transformed prize-collecting cut (PrizeCut)*

$$\begin{aligned}
&min \quad {c'}^T x - M && && (13)\\
&\quad x \text{ satisfies } (11), (12) && && (14)\\
&y(\{v_i, v_j\}) = x((v_i, v_j) + x((v_j, v_i)) && \text{for all } \{v_i, v_j\} \in E && (15)\\
&\quad y(e) \in \{0, 1\} && \text{for all } e \in E. && (16)
\end{aligned}$$

The y variables correspond to the solution to I_{PC}; note that removing them does not change the optimal solution value, neither that of the LP relaxation. To avoid adding an artificial root (which entails *big M* constants and symmetry) in the transformation to SAP, one can attempt to identify vertices that are part of all optimal solutions. To this end, let $v_i, v_j \in V$, and let W be a prize-constrained (v_i, v_j)-walk (as defined in Section 2). Define the *left-rooted prize-constrained length* of W as:

$$l^-_{pc}(W) := \max\{c_{pc}(W(v_i, v_{i_k})) \mid v_{i_k} \in V(W) \cap (T \cup \{v_j\})\}. \tag{17}$$

Furthermore, define the *left-rooted prize-constrained* (v_i, v_j)*-distance* as:

$$d^-_{pc}(v_i, v_j) := \min\{l^-_{pc}(W') \mid W' \in \mathcal{W}_{pc}(v_i, v_j)\}. \tag{18}$$

Note that in general d^-_{pc} is not symmetric. Definition (18) gives rise to

Proposition 5. *Let* $v_i, v_j \in V$*. If*

$$p(v_i) > d^-_{pc}(v_i, v_j) \tag{19}$$

is satisfied, then every optimal solution that contains v_j *also contains* v_i*.*

Proof. Let S be a tree with $v_j \in V(S)$ and $v_i \notin V(S)$. Further, let $W = (v_{i_1}, e_{i_1}, ..., e_{i_r}, v_{i_r})$ be a prize-constrained (v_i, v_j)-walk with $l^-_{pc}(W) = d^-_{pc}(v_i, v_j)$ and define $a := \min\{k \in \{1, ..., r\} \mid v_{i_k} \in V(S)\}$ and $b := \min\{k \in \{a, a+1, ..., r\} \mid v_{i_k} \in T_p \cup \{v_j\}\}$. Note that

$$c_{pc}(W(v_i, v_{i_a})) \leq c_{pc}(W(v_i, v_{i_b})). \tag{20}$$

Add the subgraph corresponding to $W(v_i, v_{i_a})$ to S, which yields a new connected subgraph S'. If S' is not a tree, make it one by removing redundant edges, without removing any node (which can only decrease $C(S')$). It holds that:

$$\begin{aligned} C(S') &\leq C(S) + c_{pc}(W(v_i, v_{i_a})) - p(v_i) \\ &\overset{(20)}{\leq} C(S) + c_{pc}(W(v_i, v_{i_b})) - p(v_i) \\ &\leq C(S) + l^-_{pc}(W) - p(v_i) \\ &= C(S) + d^-_{pc}(v_i, v_j) - p(v_i) \\ &\overset{(5)}{<} C(S). \end{aligned}$$

The relation $C(S') < C(S)$ implicates that an optimal solution that contains v_j also contains v_i. □

As for d_{pc}, computing d^-_{pc} is $\mathcal{NP}$-hard. However, one can devise a simple algorithm for finding upper bounds similar to the one for d_{pc}: Let $t_0 \in T_p$. The following adaptation of Dijkstra's algorithm provides a set of vertices $\bar{T}_{t_0}$ such that $d^-_{pc}(t_0, v) < p(t_0)$ for all $v \in \bar{T}_{t_0}$. Initialize $dist(v) := \infty$ for all $v \in V \setminus \{t_0\}$, and $dist(t_0) := 0$. Start Dijkstra's algorithm from t_0, but apply the following modifications: First, do not update vertex v_j from vertex v_i if $dist(v_i) + c(\{v_i, v_j\}) \geq p(t_0)$. Second, for $t \in T_p \setminus \{t_0\}$ set $dist(t)$ (as computed by Dijkstra) to $dist(t) - p(t)$ before inserting t into the priority queue or when updating t (if it already is in the priority queue). Third, all $v \in V \setminus T_p$ can be reinserted into the priority queue after they have been removed. Finally, define $\bar{T}_{t_0} := \{v \in V \mid dist(v) < p(t_0)\}$.

By using LP information, this heuristic can be combined with Transformation 1 to obtain a criterion for potential terminals to be part of all optimal solutions. First, note that if a separation algorithm or dual-ascent is applied, one obtains reduced costs for an LP relaxation of $DCut$ that contains only a subset of constraints (11). Second, observe that given an SAP I' obtained from I_{PC} with corresponding optimal solutions S' and S, for $t_i \in T_p$ it holds that $t_i \in V(S)$ if and only if $(v'_0, t'_i) \notin A'(S')$. As a consequence one obtains

Proposition 6. *Consider (V', A', T', c', r') obtained by applying Transformation 1 on I_{PC}. Let $\tilde{\mathcal{U}} \subseteq \{U \subset V' \mid r' \notin U, U \cap T' \neq \emptyset\}$ and let $\tilde{L}$ be the objective value and $\tilde{c}$ the reduced costs of an optimal solution to the LP:*

$$\begin{aligned} min \quad & {c'}^T x - M & & & (21) \\ & x(\delta^-(U)) \geq 1 & & \text{for all } U \in \tilde{\mathcal{U}}, & (22) \\ & x(a) \in [0,1] & & \text{for all } a \in A'. & (23) \end{aligned}$$

Moreover, let U be an upper bound on the cost of an optimal solution to I_{PC}. Finally, let $t_i \in T_p$ and let $\bar{T}_i \subseteq T_p$ such that $V(S) \cap \bar{T}_i \neq \emptyset \Rightarrow t_i \in V(S)$ for each optimal solution S to I_{PC}. If

$$\sum_{j \mid t_j \in \bar{T}_i} \tilde{c}((v'_0, t'_j)) + \tilde{L} > U \tag{24}$$

holds, then t_i is part of all optimal solutions to I_{PC}.

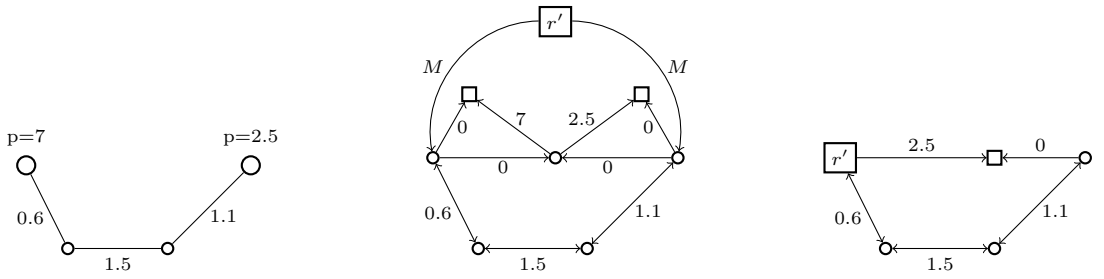
If a $t_i \in T_p$ has been shown to be part of all optimal solutions, by building $\bar{T}_i$ with Proposition 5 and using (24), Proposition 5 can again be applied—to directly identify further $t_j \in T_p$ that are part of all optimal solutions by using the condition $p(t_j) > d^-_{pc}(t_j, t_i)$. Identifying such

fixed terminals can considerably improve the strength of the techniques described in Section 2, which usually leads to further graph reductions and the fixing of additional terminals. Moreover, in this case the PCSTP can be reduced to a *rooted prize-collecting Steiner tree problem* (RPCSTP)[1], which incorporates the additional condition that a non-empty set $T_f \subseteq V$ of *fixed terminals* needs to be part of all feasible solutions. Assuming $T_p \setminus T_f = \{t_1, ..., t_z\}$, we introduce the following simple transformation:

Transformation 2 (RPCSTP to SAP)**.**
Input: *RPCSTP* (V, E, T_f, c, p) *and* $t_p, t_q \in T_f$
Output: *SAP* (V', A', T', c', r')

1. *Set* $V' := V$, $A' := \{(v, w) \in V' \times V' \mid \{v, w\} \in E\}$, $c' := c$, $r' := t_q$.
2. *For each* $i \in \{1, ..., z\}$*:*
   (a) *add node* t'_i *to* V',
   (b) *add arc* (t_i, t'_i) *of weight* 0 *to* A',
   (c) *add arc* (t_p, t'_i) *of weight* $p(t_i)$ *to* A'.
3. *Define set of terminals* $T' := \{t'_1, ..., t'_z\} \cup T_f$.
4. ***Return*** (V', A', T', c', r').

A comparison of Transformation 1 and Transformation 2 is illustrated in Figure 2.



(a) PCSTP instance

(b) SAP instance obtained from Transformation 1

(c) SAP instance obtained from Transformation 2

Figure 2: Illustration of a PCSTP instance (left) and the equivalent SAP obtained by Transformation 1 (middle). Given the information that the potential terminal with weight $p = 7$ is part of at least one optimal solution, Transformation 2 yields the SAP depicted on the right. The terminals of the SAPs are drawn as squares and the (two) potential terminals for the PCSTP are enlarged.

For an RPCSTP (V, E, T_f, c, p, r) we define the *transformed rooted prize-collecting cut* ($PrizeRCut$) formulation, similar to $PrizeCut$, based on the SAP instance (V', A', T', c', r') obtained from Transformation 2:

Formulation 3. *Transformed rooted prize-collecting cut (PrizeRCut)*

$$\min\{{c'}^T x \mid x \text{ satisfies } (11), (12),\ (x, y) \text{ satisfies } (15),\ y \text{ satisfies } (16)\}. \tag{25}$$

[1]Note that in the literature it is more common to denote only problems with exactly one fixed terminal as rooted prize-collecting Steiner tree problem.

By $PrizeRCut(I_{RPC}, t_p, t_q)$ we denote the $PrizeRCut$ formulation for an RPCSTP I_{RPC} when using (fixed) terminals t_p, t_q in Transformation 2. One may wonder whether the choice of t_p and t_q affects $v_{LP}(PrizeRCut(I_{RPC}, t_p, t_q))$; in fact, it does not, and even more:

Proposition 7. *Let I_{RPC} be an RPCSTP and let $t_p, t_q, t_{\tilde{p}}, t_{\tilde{q}}$ be any of its fixed terminals. Define $R(t_i, t_j) := \mathcal{P}_{LP}(PrizeRCut(I_{RPC}, t_i, t_j))$. It holds that:*

$$proj_y(R(t_p, t_q)) = proj_y(R(t_{\tilde{p}}, t_{\tilde{q}})). \tag{26}$$

Proof. Let (V, E, T_f, c, p) be the RPCSTP I_{RPC} and denote the SAP resulting from applying Transformation 2 on (I_{RPC}, t_p, t_q) by (V', A', T', c', t_q). Set $D = (V', A')$. Furthermore, let x, y be a feasible solution to the LP relaxation of $PrizeRCut(I_{RPC}, t_p, t_q)$—so $(x, y) \in R(t_p, t_q)$. For ease of presentation, we will use the notation x_{ij} instead of $x((v_i, v_j))$ for an arc (v_i, v_j). The proposition will be proved in two steps: first by fixing t_q and changing t_p, and second by fixing t_p and changing t_q. Note that due to symmetry reasons in both cases it is sufficient to show that one projection is contained in the other.

1) $proj_y(R(t_p, t_q)) = proj_y(R(t_{\tilde{p}}, t_q))$ Let $\tilde{I}_{\tilde{p}} = (\tilde{V}, \tilde{A}, \tilde{T}, \tilde{c}, t_q)$ be the SAP resulting from applying Transformation 2 on $(I_{RPC}, t_{\tilde{p}}, t_q)$, and set $\tilde{D} := (\tilde{V}, \tilde{A})$; note that $\tilde{V} = V'$ and $\tilde{T} = T'$. Define $\tilde{x} \in [0, 1]^{\tilde{A}}$ by $\tilde{x}((t_{\tilde{p}}, t'_i)) := x((t_p, t'_i))$ for $i = 1, ..., z$ (with the notation from Transformation 2) and by $\tilde{x}_{ij} := x_{ij}$ for all remaining arcs. Suppose that there is a $U \subset \tilde{V}$ with $t_q \notin U$ and $U \cap \tilde{T} \neq \emptyset$ such that $\tilde{x}(\delta^-_{\tilde{D}}(U)) < 1$. From $x(\delta^-_D(U)) \geq 1$ and the construction of $\tilde{x}$ it follows that $t_{\tilde{p}} \in U$—otherwise $\tilde{x}(\delta^-_{\tilde{D}}(U)) \geq x(\delta^-_D(U))$. For $U^z := U \setminus \{t'_1, ..., t'_z\}$ one obtains

$$x(\delta^-_D(U^z)) = \tilde{x}(\delta^-_{\tilde{D}}(U^z)) \leq \tilde{x}(\delta^-_{\tilde{D}}(U)) < 1. \tag{27}$$

Because of $t_q \notin U^z$ and $U^z \cap \tilde{T} \supseteq \{t_{\tilde{p}}\} \neq \emptyset$, one obtains a contradiction from (27). Therefore, $\tilde{x}$ satisfies (11) for the SAP $\tilde{I}_{\tilde{p}}$. Furthermore, $\tilde{y}$ defined by $\tilde{y}(\{v_i, v_j\}) := \tilde{x}_{ij} + \tilde{x}_{ji}$ for all $\{v_i, v_j\} \in E$ satisfies $\tilde{y} = y$.

2) $proj_y(R(t_p, t_q)) = proj_y(R(t_p, t_{\tilde{q}}))$ Define the SAP $\tilde{I}_{\tilde{q}} := (V', A', T', c', t_{\tilde{q}})$ (the result of transforming $(I_{RPC}, t_p, t_{\tilde{q}})$). As there is only one underlying directed graph (namely D), in the following we write δ^- instead of δ^-_D. Let f be a 1-unit flow from t_q to $t_{\tilde{q}}$ such that $f_{ij} \leq x_{ij}$ for all $(v_i, v_j) \in A'$. Define $\tilde{x}$ by $\tilde{x}_{ij} := x_{ij} + f_{ji} - f_{ij}$ for all $(v_i, v_j) \in A'$. Let $U \subset V'$ such that $t_{\tilde{q}} \notin U$ and $U \cap T' \neq \emptyset$. If $t_q \notin U$, then $f(\delta^-(U)) = f(\delta^+(U))$ and so $\tilde{x}(\delta^-(U)) = x(\delta^-(U)) \geq 1$. On the other hand, if $t_q \in U$, then $f(\delta^+(U)) = f(\delta^-(U)) + 1$, so

$$\tilde{x}(\delta^-(U)) \geq x(\delta^-(U)) + 1 \geq 1. \tag{28}$$

Consequently, $\tilde{x}$ satisfies (11) for the SAP $\tilde{I}_{\tilde{q}}$. From $x_{ij} + x_{ji} \leq 1$ for all $(v_i, v_j) \in A'$, it follows that $\tilde{x} \in [0, 1]^{A'}$, and for the corresponding $\tilde{y}$ one verifies $\tilde{y} = y$. □

Consequently, if only the y variables are of interest, we write $PrizeRCut(I_{RPC})$ instead of $PrizeRCut(I_{RPC}, t_p, t_q)$. For the (heuristic) dual-ascent algorithm the choice of t_p and t_q still matters, as it can change both lower bound and reduced costs. Therefore, we repeat the dual-ascent reduction techniques [29] on several SAPs resulting from different choices of t_p and t_q.

From the definitions of Transformation 1 and 2 one can acknowledge that switching from $PrizeCut$ to $PrizeRCut$ (if possible) does not deteriorate (and can improve) the tightness of the LP relaxation; due to its importance we formally state this observation:

Lemma 1. *For $I_{PC} = (V, E, c, p)$ let $T_0 \subseteq T_p$ such that $T_0 \subseteq V(S)$ for at least one optimal solution S to I_{PC}. Let $I_{T_0} := (V, E, T_0, c, p)$ be an RPCSTP. With $R_{T_0} := \mathcal{P}_{LP}(PrizeRCut(I_{T_0}))$, $R := \mathcal{P}_{LP}(PrizeCut(I_{PC}))$ it holds that*

$$proj_y(R_{T_0}) \subseteq proj_y(R). \tag{29}$$

Proposition 8. *With T_0 and I_{T_0} defined as in Lemma 1 the inequality*

$$v_{LP}(PrizeCut(I_{PC})) \leq v_{LP}(PrizeRCut(I_{T_0})) \tag{30}$$

holds and can be strict.

A proof of Proposition 8 is given in Appendix B.3.

Finally, by combining the reductions to RPCSTP and SAP with the reductions techniques described in Section 2, it is sometimes possible to either eliminate or fix each potential terminal. Hence the instance becomes an SPG, which allows to apply a number of further algorithmic techniques [13, 26].

4 Reduction-based exact solving

This section describes how the new algorithms are integrated within a branch-and-cut framework. Also, the performance of the resulting solver is discussed.

4.1 Interleaving the components within branch-and-cut

The exact solver described in this article is realized within the branch-and-cut based Steiner tree framework SCIP-Jack [13]. SCIP-Jack already includes reduction techniques for PCSTP [29] (in the sense of Section 2), but almost all of them have been replaced by new methods introduced in this article. Furthermore, we use the reduction techniques for domain propagation, translating the deletion of edges and the fixing of potential terminals into variable fixings in the IP. Additionally, we employ a technique similar to the probing [30] approach for general MIPs: Instead of setting binary variables to 0 or 1, we fix or delete potential terminals. By using the left-rooted prize-constrained distance, in each case it is often possible to either fix or delete additional potential terminals—which usually allows for further graph reductions.

SCIP-Jack also includes several (generic) primal heuristics that can be applied for PCSTP. Most compute new solutions on newly built subgraphs (e.g. by merging feasible solutions). For these heuristics the new reduction techniques can often increase the solution quality. In turn, an improved upper bound can allow for further graph reductions (e.g. by the terminal-regions composition) or to fix additional terminals (by means of Proposition 6). Additionally, we have implemented a new primal heuristic that starts with a single (potential or fixed) terminal and connects other terminals t_i to the current subtree S if $\min_{v \in V(S)} d^-_{pc}(t_i, v) \leq p(t_i)$ (only upper bounds on d^-_{pc} are used). A similar approach has been implemented as a local search heuristic.

Also the LP kernel interacts with the remaining components: By means of the prize-constrained distances and upper bounds provided by the heuristics it is usually possible to switch to the $PrizeRCut$ formulation. In turn, the reduced costs and lower bound provided by an improved LP solution can be used to reduce the problem size [29]—which can even enable further prize-constrained walk based reductions. Moreover, besides the separation of (11), already implemented in SCIP-Jack, we also separate constraints for $TransRCut$ of the form

$$x(\delta^-(v_j)) + x((t_p, t'_i)) \leq 1 \qquad t_i \in T_p \setminus T_f, v_j \in \{v \in V \mid d^-_{pc}(t_i, v) < p(t_i)\}.$$

The constraints represent the implication that $v \in V(S) \Rightarrow t \in V(S)$ for any optimal solution S if $d^-_{pc}(t, v) < p(t)$. Corresponding constraints are separated for $TransCut$.

Finally, branching is performed on vertices—by rendering vertex v_i to branch on a fixed terminal (and transforming the problem to RPCSTP if not already done) in one branch-and-bound child node and removing it in the other. As in probing, the implications from the left-rooted prize-constrained distance often allow further graph changes. Throughout the solving process, we switch to the SPG solver of SCIP-Jack [13] if all potential terminals could be fixed.

4.2 Computational results

To the best of our knowledge, the three strongest exact algorithms for PCSTP are from [11, 13, 22]. While no solver dominates on all benchmark test sets, the branch-and-bound solver from [22] is competitive on almost all, and on several ones orders of magnitude faster—it is even faster than state-of-the-art heuristic methods [12]. Thus it will in the following be used for comparison.

Table 1: Details on PCSTP tests sets.

Name	Instances	$\|V\|$	$\|E\|$	Status	Description
JMP	34	100 - 400	315 - 1576	solved	Sparse instances of varying structure, introduced in [18].
Cologne1	14	741 - 751	6332 - 6343	solved	Instances derived from the design of fiber optic networks for German cities [24].
Cologne2	15	1801 - 1810	16719 - 16794	solved	
CRR	80	500 - 1000	25000	solved	Mostly sparse instances, based on C and D test sets of the SteinLib [24].
E	40	2500	3125 - 62500	solved	Mostly sparse instances originally for SPG, introduced in [24].
ACTMOD	8	2034 - 5226	3335 - 93394	solved	Real-world instances derived from integrative biological network analysis [9].
HANDBI	14	158400	315808	unsolved	Images of hand-written text derived from a signal processing problem [1].
HANDBD	14	169800	338551	unsolved	
PUCNU	18	64 - 4096	192 - 28512	unsolved	Hard instances derived from the PUC set for SPG. From 11th DIMACS Challenge.
H	14	64 - 4096	192 - 24576	unsolved	Hard instances based on hypercubes. From 11th DIMACS Challenge.

The computational experiments were performed on Intel Xeon CPUs E3-1245 with 3.40 GHz and 32 GB RAM. For our approach CPLEX 12.7.1[2] is employed as underlying LP solver—[22] does not use an LP solver. Furthermore, only single-thread mode was used (as [22] does not support multiple threads. For the following experiments 11 benchmark test sets from the literature and the 11th DIMACS Challenge are used, as detailed in Table 1.

Table 2 provides aggregated results of the experiments with a time limit of one hour. The first column shows the test set considered in the current row. Columns two and three show the shifted geometric mean [2] (with shift 1) of the run time taken by the respective solvers. The next two columns provide the maximum run time, the last two columns the number of solved instances.

The new solver is on each test set faster than or as fast as [22], both in terms of the maximum and average run time. While both solvers can solve all instances from JMP, CRR, Cologne1, Cologne2, and ACTMOD to optimality, on all remaining test sets except HANDBD the new approach solves more instances than [22]. Moreover, it solves all instances solved by [22], and the primal-dual gap on each instance that cannot be solved by either solver is smaller than

[2]http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

Table 2: Computational comparison of the solver described in [22], denoted by *[22]*, and the solver described in this article, denoted by *new*.

		mean time [s]		max. time [s]		# solved	
Test set	#	[22]	new	[22]	new	[22]	new
JMP	34	0.0	0.0	0.0	0.0	34	34
Cologne1	14	0.0	0.0	0.1	**0.0**	14	14
Cologne2	15	0.1	0.1	0.2	**0.1**	15	15
CRR	80	0.1	0.1	5.7	**1.1**	80	80
ACTMOD	8	0.9	**0.3**	3.5	**1.5**	8	8
E	40	1.8	**0.2**	>3600	**34.5**	37	**40**
HANDBI	14	36.5	**14.9**	>3600	>3600	12	**13**
HANDBD	14	34.1	**17.9**	>3600	>3600	13	13
I640	100	8.7	**6.1**	>3600	>3600	90	**91**
PUCNU	18	278.9	**80.2**	>3600	>3600	7	**11**
H	14	488.7	**477.4**	>3600	>3600	4	**5**

that of [22]—usually by a factor of more than 2. Furthermore, the new solver improves the best known upper bounds for more than a third of the previously unsolved instances from the DIMACS Challenge, with two being solved to optimality, as detailed in A.

5 Acknowledgements

This work was supported by the BMWi project *Realisierung von Beschleunigungsstrategien der anwendungsorientierten Mathematik und Informatik für optimierende Energiesystemmodelle - BEAM-ME* (fund number 03ET4023DE). 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).

References

[1] 11th DIMACS Challenge. `http://dimacs11.zib.de/`. Accessed: November 10. 2018.

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

[3] Murodzhon Akhmedov, Ivo Kwee, and Roberto Montemanni. A divide and conquer matheuristic algorithm for the prize-collecting steiner tree problem. *Computers & Operations Research*, 70:18 – 25, 2016.

[4] David L. Applegate, Robert E. Bixby, Vaek Chvatl, and William J. Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, 2006.

[5] Daniel Bienstock, Michel X. Goemans, David Simchi-Levi, and David P. Williamson. A note on the prize collecting traveling salesman problem. *Math. Program.*, 59:413–420, 1993.

[6] Gizem Bolukbasi and Ayse Selin Kocaman. A prize collecting steiner tree approach to least cost evaluation of grid and off-grid electrification systems. *Energy*, 160:536 – 543, 2018.

[7] S. A. Canuto, M. G. C. Resende, and C. C. Ribeiro. Local search with perturbations for the prize-collecting Steiner tree problem in graphs. *Networks*, 2001.

[8] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, 1(1):269–271, December 1959.

[9] Marcus T. Dittrich, Gunnar W. Klau, Andreas Rosenwald, Thomas Dandekar, and Tobias Müller. Identifying functional modules in protein-protein interaction networks: an integrated exact approach. In *ISMB*, pages 223–231, 2008.

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

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

[12] Zhang-Hua Fu and Jin-Kao Hao. Knowledge-guided local search for the prize-collecting steiner tree problem in graphs. *Knowl.-Based Syst.*, 128:78–92, 2017.

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

[14] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[15] Ambros Gleixner, Michael Bastubbe, Leon Eifler, Tristan Gally, Gerald Gamrath, Robert Lion Gottwald, Gregor Hendel, Christopher Hojny, Thorsten Koch, Marco E. Lübbecke, Stephen J. Maher, Matthias Miltenberger, Benjamin Müller, Marc E. Pfetsch, Christian Puchert, Daniel Rehfeldt, Franziska Schlösser, Christoph Schubert, Felipe Serrano, Yuji Shinano, Jan Merlin Viernickel, Matthias Walter, Fabian Wegscheider, Jonas T. Witt, and Jakob Witzig. The scip optimization suite 6.0. Technical Report 18-26, ZIB, Takustr. 7, 14195 Berlin, 2018.

[16] Michel X. Goemans and Young-Soo Myung. A catalog of Steiner tree formulations. *Networks*, 23(1):19–28, 1993.

[17] Trey Ideker, Owen Ozier, Benno Schwikowski, and Andrew F. Siegel. Discovering regulatory and signalling circuits in molecular interaction networks. In *ISMB*, pages 233–240, 2002.

[18] David S. Johnson, Maria Minkoff, and Steven Phillips. The Prize Collecting Steiner Tree Problem: Theory and Practice. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '00, pages 760–769, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics.

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

[20] Gunnar W. Klau, Ivana Ljubić, Andreas Moser, Petra Mutzel, Philipp Neuner, Ulrich Pferschy, Günther Raidl, and René Weiskircher. Combining a memetic algorithm with integer programming to solve the prize-collecting steiner tree problem. In Kalyanmoy Deb, editor, *Genetic and Evolutionary Computation – GECCO 2004*, pages 1304–1315, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

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

[22] Markus Leitner, Ivana Ljubic, Martin Luipersbeck, and Markus Sinnl. A dual ascent-based branch-and-bound framework for the prize-collecting steiner tree and related problems. *INFORMS Journal on Computing*, 30(2):402–420, 2018.

[23] Ivana Ljubic, Ren Weiskircher, Ulrich Pferschy, Gunnar W. Klau, Petra Mutzel, and Matteo Fischetti. An Algorithmic Framework for the Exact Solution of the Prize-Collecting Steiner Tree Problem. *Mathematical Programming*, 105(2-3):427–449, 2006.

[24] Ivana Ljubi. *Exact and Memetic Algorithms for Two Network Design Problems*. PhD thesis, Vienna University of Technology, 2004.

[25] Yi-Fei Ming, Si-Bo Chen, Yong-Quan Chen, and Zhang-Hua Fu. A fast vertex-swap operator for the prize-collecting steiner tree problem. In Yong Shi, Haohuan Fu, Yingjie Tian, Valeria V. Krzhizhanovskaya, Michael Harold Lees, Jack Dongarra, and Peter M. A. Sloot, editors, *Computational Science – ICCS 2018*, pages 553–560, Cham, 2018. Springer International Publishing.

[26] Tobias Polzin and Siavash Vahdati Daneshmand. Improved algorithms for the steiner problem in networks. *Discrete Appl. Math.*, 112(1-3):263–300, September 2001.

[27] Daniel Rehfeldt and Thorsten Koch. Transformations for the Prize-Collecting Steiner Tree Problem and the Maximum-Weight Connected Subgraph Problem to SAP. *Journal of Computational Mathematics*, 36(3):459 – 468, 2018.

[28] Daniel Rehfeldt and Thorsten Koch. Combining NP-Hard Reduction Techniques and Strong Heuristics in an Exact Algorithm for the Maximum-Weight Connected Subgraph Problem. *SIAM Journal on Optimization*, in press.

[29] Daniel Rehfeldt, Thorsten Koch, and Stephen Maher. Reduction Techniques for the Prize Collecting Steiner Tree Problem and the Maximum-Weight Connected Subgraph Problem. *Networks*, in press.

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

[31] L. Schmidt, C. Hegde, P. Indyk, L. Lu, X. Chi, and D. Hohl. Seismic feature extraction using steiner tree methods. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1647–1651, April 2015.

[32] Yahui Sun. *Classical, prize-collecting and node-weighted Steiner tree problems in graphs*. PhD thesis, 2018.

[33] Eduardo Uchoa. Reduction Tests for the Prize-collecting Steiner Problem. *Oper. Res. Lett.*, 34(4):437–444, July 2006.

[34] R.T. Wong. A dual ascent approach for Steiner tree problems on a directed graph. *Mathematical Programming*, 28:271287, 1984.

Table 3: Improvements on unsolved DIMACS instances.

Name	gap [%]	new UB	previous UB
cc7-3nu	**opt**	**270**	271
cc10-2nu	**opt**	**167**	168
cc11-2nu	1.2	304	305
cc12-2nu	1.0	565	568
hc9p2	1.4	30230	30242
hc10p	1.4	59778	59866
hc10p2	1.5	59807	59930
hc11p	1.6	118729	119191
hc11p2	1.8	118979	119236
i640-341	0.5	29691	29700
i640-344	0.6	29910	29921

A Results for unsolved DIMACS instances

Using an extended time limit of four hours, we could improve or even solve more than one third of the (previously) unsolved PCSTP instances from the 11th DIMACS Challenge. All improved instances are listed in Table 3, with the first column giving the name of the instance, the second its primal-dual gap, the third the improved found bound, and the fourth the previously best known one.[3]

B Further proofs

B.1 Proof of Proposition 2

Proof. Initially, define $b : V \to T_p$ such that $v \in H_{b(v)}$ for all $v \in V$. Assume that there exists an optimal solution S such that $v_i \in V(S)$. Denote the (unique) path in S between v_i and any $t_j \in V(S) \cap T_p$ by Q_j and the set of all such paths by $\mathcal{Q}$. First, note that $|\mathcal{Q}| \geq 2$, because if $\mathcal{Q}$ just contained one path, say Q_l, the single-vertex tree $\{t_l\}$ would be of smaller cost than S. Second, if a vertex v_k is contained in two distinct paths in $\mathcal{Q}$, the subpaths of these two paths between v_i and v_k coincide. Otherwise there would need to be a cycle in S. Additionally, there are at least two paths in $\mathcal{Q}$ having only the vertex v_i in common. Otherwise, due to the precedent observation, all paths would have one edge $\{v_i, v_i'\}$ in common, which could be discarded to yield a tree of smaller cost than $C(S)$.

Let $Q_k \in \mathcal{Q}$ and $Q_l \in \mathcal{Q}$ be two distinct paths with $V(Q_k) \cap V(Q_l) = \{v_i\}$ such that

$$|\{\{v_x, v_y\} \in E(Q_k) \cup E(Q_l) \mid b(v_x) \neq b(v_y)\}| \tag{31}$$

is minimized. Define $\mathcal{Q}^- := \mathcal{Q} \setminus \{Q_k, Q_l\}$. For all $Q_r \in \mathcal{Q}^-$, denote by Q_r' the subpath of Q_r between t_r and the first vertex not in H_{t_r}. Suppose that Q_k has an edge $e \in E(S)$ in common with a Q_r': Consequently, Q_l cannot have any edge in common with Q_r, because this would require a cycle in S. Furthermore, Q_k and Q_r have to contain a joint subpath including v_i and e. But this would imply that Q_k contained at least one additional edge $\{v_x, v_y\}$ with $b(v_x) \neq b(v_y)$. Thus, Q_r would have initially been selected instead of Q_k.

Following the same line of argumentation, one validates that Q_l has no edge in common with

[3]The solution of the instance *cc7-3nu* is already noted in the report [15], but is based on techniques described in this article (which have not been published before).

any Q'_r. Conclusively, the paths Q_k, Q_l and all Q'_r are edge-disjoint. Thus one obtains:

$$\begin{aligned} C(S) &= \sum_{e \in E(S)} c(e) + \sum_{v \in V \setminus V(S)} p(v) \\ &\geq \Bigg(\sum_{Q_r \in \mathcal{Q}^-} c(E(Q'_r)) \Bigg) + c(E(Q_k)) + c(E(Q_l)) + \sum_{v \in V \setminus V(S)} p(v) \\ &\geq \sum_{q=1}^{s-2} r_H^{pc}(t_q) + c(E(Q_k)) + c(E(Q_l)) \\ &\geq \sum_{q=1}^{s-2} r_H^{pc}(t_q) + \underline{d}(\underline{v}_i, \underline{v}_{i,1}) + \underline{d}(\underline{v}_i, \underline{v}_{i,2}), \end{aligned}$$

which proves the proposition. □

B.2 Proof of Proposition 4

We will show the $\mathcal{NP}$-hardness already for the SPG variant of the terminal-regions decomposition (which implies the $\mathcal{NP}$-hardness for PCSTP). For an SPG (V, E, T, c) define the terminal-regions decomposition as a partition $H = \{H_t \subseteq V \mid T \cap H_t = \{t\}\}$ of V such that for each $t \in T$ the subgraph induced by H_t is connected. Define for all $t \in T$

$$r_H(t) := \min\{d(t, v) \mid v \notin H_t\}. \tag{32}$$

Note that this definition is just a special case of the PCSTP version (for PCSTP instances with sufficiently high vertex weights). First, the decision variant of the terminal-regions decomposition problem is stated. Let $\alpha \in \mathbb{N}_0$ and let $G_0 = (V_0, E_0)$ be an undirected, connected graph with costs $c : E \to \mathbb{N}$. Furthermore, set $T_0 := \{v \in V_0 \mid p(v) > 0\}$, and assume that $\alpha < |T_0|$. For each terminal-regions decomposition H_0 of G_0 define $T'_0 \subsetneq T_0$ such that $|T'_0| = \alpha$ and $r_{H_0}(t') \geq r_{H_0}(t)$ for all $t' \in T'_0$ and $t \in T_0 \setminus T'_0$. Let $C_{H_0} := \sum_{t \in T_0 \setminus T'_0} r_{H_0}(t)$. We now define the α terminal-regions decomposition problem as follows: Given a $k \in \mathbb{N}$, is there a terminal-regions decomposition H_0 such that $C_{H_0} \geq k$? The next lemma forthwith establishes the $\mathcal{NP}$-hardness of finding a terminal-regions decomposition that maximizes (8), or (9)—which corresponds to $\alpha = 2$ and $\alpha = 3$, respectively.

Lemma 2. *For each $\alpha \in \mathbb{N}_0$ the α terminal-regions decomposition problem is $\mathcal{NP}$-complete.*

Proof. Given a terminal-regions decomposition H_0 it can be tested in polynomial time whether $C_{H_0} \geq k$. Consequently, the terminal-regions decomposition problem is in $\mathcal{NP}$.

In the remainder it will be shown that the ($\mathcal{NP}$-complete [14]) independent set problem can be reduced to the terminal-regions decomposition problem. To this end, let $G_{ind} = (V_{ind}, E_{ind})$ be an undirected, connected graph and $k \in \mathbb{N}$. The problem is to determine whether an independent set in G_{ind} of cardinality at least k exists. To establish the reduction, construct a graph G_0 from G_{ind} as follows. Initially, set $G_0 = (V_0, E_0) := G_{ind}$. Next, extend G_0 by replacing each edge $e_l = \{v_i, v_j\} \in E_0$ with a vertex v'_l and the two edges $\{v_i, v'_l\}$ and $\{v_j, v'_l\}$. Define edge weights $c_0(e) = 1$ for all $e \in E_0$ (which includes the newly added edges). If $\alpha > 0$, choose an arbitrary $v_i \in V_0 \cap V_{ind}$ and add for $j = 1, ..., \alpha$ vertices $t_i^{(j)}$ to both V_0 and T_0. Finally, add for $j = 1, ..., \alpha$ edges $\{v_i, t_i^{(j)}\}$ with $c_0(\{v_i, t_i^{(j)}\}) = 2$ to E_0.

First, one observes that the size $|V_0| + |E_0|$ of the new graph G_0 is a polynomial in the size $|V_{ind}| + |E_{ind}|$ of G_{ind}. Next, $r_{H_0}(v_i) = 2$ holds for a vertex $v_i \in G_0 \cap G_{ind}$ if and only

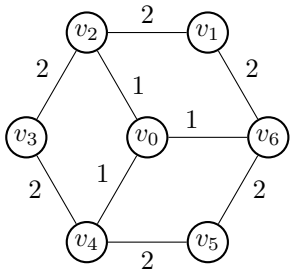
if H_{v_i} contains all (newly inserted) adjacent vertices of v_i in G_0. Moreover, in any terminal-regions decomposition H_0 for (G_0, c_0), it holds that $r_{H_0}(t_i^{(j)}) = 2$ for $j = 1, ..., \alpha$. Hence, there is an independent set in G_{ind} of cardinality at least k if and only if there is a terminal-regions decomposition H_0 for (V_0, E_0, T_0, c_0) such that

$$C_{H_0} \geq |V_{ind}| + k$$

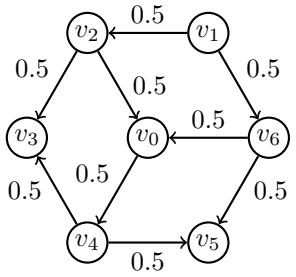
This proves the proposition. □

B.3 Proof of Proposition 8

Proof. First it follows from the construction of Transformation 1 and 2 that each optimal solution x^0, y^0 to the LP relaxation of $TransRCut(I_{T_0})$ can be transformed to a solution x, y to the LP relaxation of $TransCut(I_{PC})$ without changing the objective value: By setting $x((v_i, v_j)) := x^0((v_i, v_j))$ and $x((v_j, v_i)) := x^0((v_j, v_i))$ for all $\{v_i, v_j\} \in E$, $x((r', t_0)) := 1$ for any $t_0 \in T_0$, $x((t_i, t_i')) := 1$ for all $t_i \in T_0$, and by setting the remaining $x((v_i, v_j))$ accordingly. Thus $v_{LP}(PrizeCut(I_{PC})) \leq v_{LP}(PrizeRCut(I_{T_0}))$. To see that the inequality can be strict, consider the following wheel instance (which is well-known to have an integrality gap for DCut on SPG):



Set $p(v_0) = p(v_1) = p(v_3) = p(v_5) = 4$, $p(v_2) = p(v_6) = 0$, and $p(v_4) = \epsilon$ with $0 < \epsilon < 1$. Let $T_0 := \{v_0, v_1, v_3, v_4, v_5\}$. Let I be the PCSTP and I_{T_0} the corresponding RPCSTP. It holds that $v_{LP}(TransCut(I)) = 7.5 + \frac{\epsilon}{2} < 8 = v_{LP}(TransRCut(I_{T_0}))$. Part of the solution corresponding to $v_{LP}(TransCut(I))$ is shown below (with numbers next to the arcs denoting the x values), the remaining x and y are set accordingly (e.g., $x((r', v_1)) = 1$).



□