Daniel Rehfeldt, Thorsten Koch

On the exact solution of prize-collecting Steiner tree problems

ZIB Report 20-11 (April 2020)

Zuse Institute Berlin
Takustr. 7
14195 Berlin
Germany

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

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

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

On the exact solution of prize-collecting Steiner tree problems

Daniel Rehfeldt, Thorsten Koch

Zuse Institute Berlin,

Takustr. 7, 14195 Berlin, Germany

{rehfeldt,koch}@zib.de

TU Berlin, Chair of Software and Algorithms for Discrete Optimization,

Str. des 17. Juni 135, 10623 Berlin, Germany

October 12, 2020 (revision)

Abstract

The prize-collecting Steiner tree problem (PCSTP) is a well-known generalization of the classic Steiner tree problem in graphs, with a large number of practical applications. It attracted particular interest during the 11th DIMACS Challenge in 2014, and since then, several PCSTP solvers have been introduced in the literature. Although these new solvers further, and often drastically, improved on the results of the DIMACS Challenge, many PCSTP instances have remained unsolved. The following article describes further advances in the state of the art in exact PCSTP solving. It introduces new techniques and algorithms for PCSTP, involving various new transformations (or reductions) of PCSTP instances to equivalent problems; for example 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. Moreover, new complexity results for the exact solution of PCSTP and related problems are given, which form the base of the algorithmic developments. Finally, the new developments also translate into a strong computational performance: the resulting exact PCSTP solver outperforms all previous approaches, both in terms of run-time and solvability. In particular, it solves several formerly intractable benchmark instances from the 11th DIMACS Challenge to optimality. Moreover, several recently introduced large-scale instances with up to 10 million edges, previously considered to be too large for any exact approach, can now be solved to optimality in less than two hours.

1 Introduction

The Steiner tree problem in graphs (SPG) is one of the fundamental ($\mathcal{NP}$-hard) combinatorial optimization problems [32]. 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 transformed to a PCSTP. However, while the number of real-world applications of the classic Steiner tree problem in graphs is limited [22], the PCSTP entails many practical applications, which can be found in various areas, for instance in the design of telecommunication networks [38], electricity planning [8], computational biology [30], geophysics [48], and even machine learning [28].

The PCSTP has been extensively discussed in the literature, both from theoretical and practical perspectives. The first approximation algorithm was introduced by [7], and achieved a factor 3 approximation. This factor was later improved by [26], [31], and [19]. The latter achieve a $(2 - \frac{2}{|V|})$-approximation. Finally, [4] proposed a $(2 - \epsilon)$ approximation; with $0.03 < \epsilon < 0.04$. For approximation results on planar graphs see [5]. Moreover, a large number of heuristic algorithms for PCSTP have been suggested, see e.g. [11, 12, 21]. As to (practical) exact solving, the sophisticated branch-and-cut algorithm by [39] was an early milestone. Later, the PCSTP attracted considerable interest in the wake of the 11th DIMACS Challenge [14] in December 2014—dedicated to Steiner tree problems—where the PCSTP categories could boast the most participants by far. Furthermore, in the recent years a considerable number of additional solvers for the PCSTP have been introduced [2, 9, 20, 21, 22, 37, 40, 49]. Some of these solvers, in particular [37], 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 [20, 22], include specialized (primal and sometimes dual) heuristics [33, 37], and make use of various preprocessing methods to reduce the problem size [39, 45].

1.1 Contribution

This article introduces and analyses new techniques and algorithms for PCSTP that ultimately aim for efficient exact solution. Most of the techniques are based on, or result in reductions (or transformations) 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 us to obtain a stronger IP formulation. Moreover, several of the new methods provably dominate previous approaches. While some of the techniques require to solve $\mathcal{NP}$-hard subproblems (not yet described in the literature), the underlying concepts allow us to design empirically efficient heuristics. Furthermore, we provide complexity results for the exact solution of PCSTP (and related problems), which underpin the design of most algorithms in this article. Also these complexity results base on problem transformations.

This article extends existing work, especially [43, 44, 45]. However, it not only significantly improves known results, but also combines them with several new techniques. Indeed, a salient feature of this work is the intricate interaction of the individual algorithmic components and their wide applicability within a branch-and-cut framework—from preprocessing and probing, to IP formulation and separation methods, to heuristics, domain propagation, and branching. The integration of the new methods into an exact solver also brings significant computational advancements: The new solver is significantly faster than current state-of-the-art competitors, and furthermore solves 24 benchmark instances for the first time to optimality. The newly developed software has been integrated into the academic Steiner tree framework SCIP-Jack [22] and will be made publicly available as part of its next major release.

Finally, while it will be shown that a 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 (rooted and unrooted) maximum-weight connected subgraph problem. Moreover, one can also directly apply the PC-STP algorithms to these two problems—by using the transformations to PCSTP described in Section 2.2.

1.2 Notation and preliminaries

For a graph G we denote its vertices by $V(G)$ and its edges by $E(G)$. For a given graph G we use $n := |V(G)|$ and $m := |E(G)|$. For a walk W we likewise denote the set of vertices and the set of edges it contains by $V(W)$ and $E(W)$. By $d(v,w)$ we denote the distance of a shortest path (with respect to c) between vertices $v, w \in V$. For $U \subseteq V$ define the induced *edge cut* as $\delta(U) := \{\{u,v\} \in E \mid u \in U, v \in V \setminus U\}$; 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 a single vertex v we use the short-hand notation $\delta(v) := \delta(\{v\})$, and accordingly for directed graphs. 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 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.

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. We call $T_p := \{v \in V \mid p(v) > 0\}$ the set of *potential terminals* [37]. It will be assumed that $T_p \neq \emptyset$. For ease of presentation we use $\{t_1, t_2, ..., t_s\} := T_p$, so in particular $s := |T_p|$. A $t \in T_p$ will be called *proper potential terminal* if

$$p(t) > \min_{e \in \delta(t)} c(e). \tag{2}$$

The set of all proper potential terminals will be denoted by $T_p^+ = \{t_1^+, t_2^+, ..., t_{s^+}^+\}$, with $s^+ := |T_p^+|$. Accordingly, define $T_p^- := T_p \setminus T_p^+$. The distinction of proper and non-proper potential terminals was already made in [50], where it was noted that non-proper potential terminals allow for additional presolving methods. This distinction can also be found in [20, 37].

We will call any PCSTP solution that consists of just one vertex *trivial*. If $T_p^+ = \emptyset$, then there exists a trivial optimal solution. In general, there exists an optimal solution whose leaves are a subset of T_p^+, or there exists at least one trivial optimal solution.

Finally, we define a variation of the PCSTP, the *rooted prize-collecting Steiner tree problem* (RPCSTP)[1]. The RPCSTP incorporates the additional condition that a non-empty set $T_f \subseteq V$ of *fixed terminals* needs to be part of all feasible solutions. We assume w.l.o.g. that $p(t) = 0$ for all $t \in T_f$.

1.3 Structure

The remainder of this article is structured as follows.

- Section 2 shows that PCSTP is fixed-parameter tractable (FPT) in $|T_p^+|$. Furthermore, we discuss (known and new) transformations from the node-weighted Steiner tree and maximum-weight connected subgraph problem to PCSTP, which directly lead to FPT results. Also, we show that non-proper potential terminals naturally arise from these transformation. Overall, Section 2 provides a strong theoretical motivation for distinguishing between proper and non-proper potential terminals within PCSTP algorithms, which will be a dominating theme throughout this article.

[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, e.g. in [39]

- Section 3 introduces several new reduction techniques for PCSTP. Most importantly, a new distance function based on so-called *prize-constrained walks* is introduced. By using this distance function, we introduce for example a new edge elimination criterion. The new techniques are also compared with previous methods from the literature.
- Section 4 makes further use of the concept of prize-constrained walks. By a combination with the reduced-costs of a particular LP relaxation, prize-constrained walks allow us to find vertices that need to be part of any optimal solution. We further show how this information leads to a better IP formulation.
- Section 5 shows how to integrate the newly developed algorithms within a branch-and-cut framework. Furthermore, computational results are given, as well as comparisons with state-of-the-art PCSTP solvers.
- Finally, Section 6 offers a conclusion, and suggestions on possible future research.

2 Proper potential terminals and complexity

In a number of (real-world) PCSTP instances from the literature $|T_p^-|$ is considerably larger than $|T_p^+|$, so it seems well-worthwhile to algorithmically distinguish between proper and non-proper potential terminals. This section provides also a theoretical foundation for such a distinction. Namely, by showing how proper and non-proper potential terminals arise from problems related to PCSTP and by showing how the complexity of PCSTP depends on the number of proper potential terminals.

2.1 On the complexity of PCSTP

In the following we demonstrate that for the complexity of PCSTP the number $s^+ = |T_p^+|$ is the crucial parameter. One observes throughout this article that the complexity of several new PCSTP algorithms is likewise governed by s^+. We first show the following.

Theorem 1. *The PCSTP is fixed-parameter tractable for the parameter s^+. It can be solved in time $O(3^{s^+}n + 2^{s^+}n^2 + n^2 \log n + mn)$.*

A detailed proof is given in Appendix A.2. In the following we describe the main building blocks. Consider a RPCSTP $I_f = (G, T_f, c, p)$ with $T_p^+ = \emptyset$. By extending the well-known dynamic programming algorithm from [16], we obtain the following result.

Proposition 2. *An optimal solution to I_f can be found in time*
$O(3^{|T_f|}n + 2^{|T_f|}n^2 + n^2 \log n + mn)$.

Now we return to PCSTP. It will be assumed that no trivial solution exists for PCSTP (otherwise one needs to compare the solution found in the following with the best trivial solution). The following describes how to transform any PCSTP to an equivalent RPCSTP instance that has no proper potential terminals and satisfies $|T_f| = s^+ + 1$.

Transformation 1 (PCSTP to RPCSTP)**.**
Input: *PCSTP (V, E, c, p) with $T_p^+ \neq \emptyset$*
Output: *RPCSTP (V', E', T_f', c', p')*

1. *Initially, set $V' := V$, $E' := E$, $c' := c$; define $M := \sum_{t \in T_p^+} p(t)$.*

2. *Define $p' : V' \to \mathbb{Q}_{\geq 0}$ for all $v \in V'$ by*

$$p'(v) := \begin{cases} p(v) & \text{if } v \in T_p^-, \\ 0 & \text{otherwise.} \end{cases}$$

3. *Let $j \in \{1, ..., s^+\}$ such that $p(t_j) = \min_{t \in T_p^+} p(t)$.*
4. *Add vertex t'_0 to V'.*
5. *For each $i \in \{1, ..., s^+\}$:*
   (a) *add node t'_i with $p(t'_i) := 0$ to V';*
   (b) *add edges $\{t'_0, t_i\}$ and $\{t_i, t'_i\}$ to E', both of weight M.*
6. *For each $i \in \{1, ..., s^+\} \setminus \{j\}$:*
   (a) *add edge $\{t_i, t'_j\}$ of weight $M + p(t_j)$ to E';*
   (b) *add edge $\{t'_i, t'_j\}$ of weight $M + p(t_i)$ to E'.*
7. *Define fixed terminals $T'_f := \{t'_1, ..., t'_{s^+}\} \cup \{t'_0\}$.*
8. ***Return** (V', E', T'_f, c', p').*

Let I be a PCSTP. Let I' be the RPCSTP resulting from Transformation 1, and let S' be an optimal solution to I'. One observes that $S := S' \cap (V, E)$ is an optimal solution to I. Because of the choice of M, one further observes that for each fixed terminal of I' exactly one incident edge is in S'. Thus, for any optimal solution S to I one obtains the following relation

$$C(S) = C(S') - |T'_f|M.$$

By combining Proposition 2 and Transformation 1, one obtains Theorem 1.

Note that one can also extend Transformation 1 such that the result is an SPG with $s + 1$ terminals (and at most $2n + 1$ vertices). However, the structure of the resulting SPG does not lend itself well to an efficient practical solution by state-of-the-art SPG algorithms. Still, one can use this transformation to directly derive further complexity results for PCSTP from SPG. E.g., [51] shows that an SPG with k terminals can be solved in time $O(nk2^{k+\log_2 k \log_2 n})$, which translates to $O(ns2^{s+\log_2 s \log_2 n + \log_2 s})$ for PCSTP. One could also extend the result from [51] (by verifying them for I_f similarly to Proposition 2) to show that the same bound holds for s^+.

Having demonstrated that PCSTP is tractable if the number of proper potential terminals is bounded from above, we now turn to the opposite case. The SPG is well-known to be fixed-parameter tractable in $n - |T|$, which can be shown by enumeration of the non-terminal vertices [27]. For node-weighted Steiner tree and maximum-weight connected subgraph problems one can show similar results [10]. However, the situation for PCSTP with respect to $n - s^+$ is different, as the following proposition shows.

Proposition 3. *PCSTP is $\mathcal{NP}$-hard even if $s^+ = n$.*

Proof. We show that the ($\mathcal{NP}$-complete [23]) vertex cover problem can be reduced to the decision variant of PCSTP, such that the resulting instance satisfies $s^+ = n$. Let $G_{cov} = (V_{cov}, E_{cov})$ be an undirected graph and $k \in \mathbb{N}$. In the vertex cover problem one has to determine whether a subset of V_{cov} of cardinality at most k exists that is incident to all edges E_{cov}. Let $n := |V_{cov}|$ and $m := |E_{cov}|$. Assume that the vertices and edges of G_{cov} are given as $\{v_1, v_2, ..., v_n\}$ and

$\{e_1, e_2, ..., e_m\}$, respectively. Construct a PCSTP instance $I' = (V', E', c', p')$ with $2n + m + 1$ vertices and $2n + 2m$ edges as follows. Denote the vertices of V' by u'_i and v'_i for $i = 1, ..., n$, and w'_i for $i = 0, 1, ..., m$. For each original edge $e_i = \{v_j, v_k\} \in E_{cov}$ create the two edges $\{w'_i, v'_j\}$ and $\{w'_i, v'_k\}$ with cost $c'(\{w'_i, v'_j\}) = c'(\{w'_i, v'_k\}) = 8$. For each original vertex $v_i \in V_{cov}$ create the two edges $\{v'_i, u'_i\}$ and $\{u'_i, w'_0\}$ with cost $c(\{v'_i, u'_i\}) = 1$ and $c(\{u'_i, w'_0\}) = 4$. Finally, define the prizes of I' as follows. First, $p'(w'_i) = 10$ for $i = 0, 1, ..., m$. Second, $p'(v'_i) = p'(u'_i) = 2$ for $i = 1, ..., n$. Observe that all vertices in V' are proper potential terminals. We claim that an independent set for G_{cov} of cardinality at most k exists if and only if there is a tree $S' \subseteq (V', E')$ that satisfies

$$C(S') - 8m - 4n \leq k. \tag{3}$$

First, assume that a vertex cover with vertex index set J_{cov} exists such that $|J_{cov}| \leq k$. Build a tree $S' \subseteq (V', E')$ as follows. Initially, set

$$V'(S') := \{v'_j, u'_j \mid j \in J_{cov}\} \cup \{w_j \mid j \in \{0, ..., m\}\}.$$

For $E'(S')$ take all edges $\{v'_j, u'_j\}, \{u'_j, w'_0\}$ with $j \in J_{cov}$. Furthermore, for $i = 1, ..., m$ add exactly one edge $\{w'_i, v'_j\}$ with $j \in J_{cov}$. For S' one observes that

$$\sum_{e \in E'(S')} c'(e) = 8m + 5k$$

and

$$\sum_{v \in V' \setminus V'(S')} p'(v) = 4(n - k).$$

Thus, $C(S') - 8m - 4n = k$.

Conversely, assume that a tree S' exists that satisfies (3). Assume that S' is an optimal solution to I'. One verifies that S' contains all vertices w'_i (e.g. by using Theorem 16). Note that also $\delta_{S'}(w'_i) = 1$ for $i = 1, ..., m$. Let $J_{cov} \subseteq \{1, ..., n\}$ such that $v'_j \in S' \iff j \in J_{cov}$ and set $k' := |J_{cov}|$. From the optimality of S' one obtains

$$C(S') = 8m + 5k' + 4(n - k'). \tag{4}$$

From (3) it follows that $k' \leq k$. □

2.2 From PCSTP to related problems

The distinction of proper potential terminals also arises in relation with the *maximum-weight connect subgraph problem* (MWCSP), which is closely related to the PCSTP. Given an undirected graph $G = (V, E)$ with node weights $w : V \to \mathbb{Q}$, the MWCSP asks for a connected subgraph $S \subseteq G$ that maximizes

$$\sum_{v \in V(S)} w(v). \tag{5}$$

See e.g. [3] for more detail on MWCSP. Let $I = (V, E, w)$ be an MWCSP instance and assume that $w_0 := \min_{v \in V(S)} w(v)$ is negative (otherwise I is trivial to solve). I can be transformed to an equivalent PCSTP $I' = (V, E, c, p)$ by setting $c(e) := -w_0$ for all $e \in E$, and $p(v) := w(v) - w_0$ for all $v \in V$, as described in [15]. It should be noted, though, that due to the special form of I', algorithms tailored to MWCSP, such as [43], perform vastly better in practice on I than PCSTP algorithms on I'. As to proper potential terminals, one observes the following: For any $v \in V$ it holds that $w(v) > 0$ (in I) if and only if v is a proper potential terminal in I'. Thus, one also obtains the following corollary (which improves on a result from [10]).

Corollary 4. *MWCSP can be solved in time $O(3^q n + 2^q n^2 + n^2 \log n + mn)$, where q denotes the number of positive weight vertices.*

Another natural distinction between proper and non-proper potential terminals can be observed for the *node-weighted Steiner tree problem* (NWSTP), see e.g. [34]. Given an undirected, connected graph $G = (V, E)$ with vertex weights $w : V \to \mathbb{Q}_{\geq 0}$ and edge weights $c : E \to \mathbb{Q}_{\geq 0}$, and given a set of *terminals* $T \subseteq V$, the NWSTP asks for tree $S \subseteq G$ with $T \subseteq V(S)$ that minimizes

$$\sum_{e \in E(S)} c(v) + \sum_{v \in V(S)} w(v). \tag{6}$$

Let $I = (V, E, T, c, w)$ be an NWSTP instance and assume w.l.o.g. that $w(t) = 0$ for all $t \in T$. I can be reduced to an equivalent PCSTP $I' = (V, E, c', p')$ by the following, new, transformation. Let $z := \max_{v \in V} w(v)$. Define $c'(e) := c(e) + z$ for all $e \in E$. Define $p'(t) := k$ for all $t \in T$, with a sufficiently large $k \in \mathbb{Q}_{\geq 0}$, e.g. $k = \sum_{e \in E} c'(e)$. Finally, define $p'(v) = z - w(v)$ for all $v \in V \setminus T$. A tree S is an optimal solution to I if and only if it is an optimal solution to I'. Furthermore, in this case S satisfies

$$\sum_{v \in V(S)} w(v) = C(S) - (|T| - 1)z - \sum_{v \in V} p(v). \tag{7}$$

Note that $T_p^- \supseteq \{v \in V \setminus T \mid w(v) < z\}$. Likewise, the set of terminals T for I corresponds to the set of proper potential terminals for I'.

One can alternatively transform NWSTP to RPCSTP to avoid the use of the large constant k. Also, one immediately obtains the following corollary (which was already shown algorithmically in [10]) from Proposition 2.

Corollary 5. *NWSTP can be solved in time $O(3^k n + 2^k n^2 + n^2 \log n + mn)$, where k denotes the number of terminals.*

Finally, one notes that PCSTP can be seen as a generalization of both MWCSP and NWSTP, as these problems can be transformed to a PCSTP with the same number of edges and vertices.

3 Reductions within the problem class

The methods 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. [37, 39, 45, 50], but most are dominated by the methods described in the following. The new methods will not only be employed for classic preprocessing, but also throughout the entire solving process, e.g. for domain propagation, or within heuristics.

3.1 Taking short walks

The following approach uses a new, walk-based, distance function. It generalizes the bottleneck Steiner distance concept that was the central theme of [50]. Let $v, w \in V$. A finite walk $W = (v_1, e_1, v_2, e_2, ..., e_r, v_r)$ with $v_1 = v$ and $v_r = w$ will be called *prize-constrained (v, w)-walk* if no $v \in T_p^+ \cup \{v, w\}$ 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 $W(v_k, v_l) := (v_k, e_k, v_{k+1}, e_{k+1}, ..., e_l, v_l)$; note that $W(v_k, v_l)$ is again a prize-constrained walk. In the following, let W be a prize-constrained (v, w)-walk. Define the *prize-collecting cost* of W as

$$c_{pc}(W) := \sum_{e \in E(W)} c(e) - \sum_{u \in V(W) \setminus \{v, w\}} p(u). \tag{8}$$

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

$$l_{pc}(W) := \max\{c_{pc}(W(v_k, v_l)) \mid 1 \leq k \leq l \leq r,\ v_k, v_l \in T_p^+ \cup \{v, w\}\}. \tag{9}$$

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 any two disjoint trees that contain the first and the last vertex of W, respectively. Finally, we denote the set of all prize-constrained (v, w)-walks by $\mathcal{W}_{pc}(v, w)$ and define the *prize-constrained distance* between v and w as

$$d_{pc}(v, w) := \min\{l_{pc}(W) \mid W \in \mathcal{W}_{pc}(v, w)\}. \tag{10}$$

Note that $d_{pc}(v, w) = d_{pc}(w, v)$ for any $v, w \in V$. Also, it is important to note that for each subwalk the cost of an edge and the prize of an inner vertex is counted exactly once, even if an edge or vertex is contained multiple times in the subwalk. Using the same measuring concept, one could in fact also allow arbitrary finite walks instead of prize-constrained ones—and count for each subwalk the costs of its edges and the prizes of its inner vertices exactly once. However, the prize-constrained distance is already arbitrarily stronger than the bottleneck Steiner distance, and additionally more closely related to the algorithms described below.

By using the prize-constrained distance one can formulate a reduction criterion that dominates the *special distance* test from [50]. This criterion is expressed in the following theorem:

Theorem 6. *Let $\{v, w\} \in E$. If*

$$c(\{v, w\}) > d_{pc}(v, w) \tag{11}$$

is satisfied, then $\{v, w\}$ cannot be contained in any optimal solution to I_{PC}.

Proof. Let S be a tree with $\{v, w\} \in E(S)$. Further, let $W = (v_1, e_1, ..., e_r, v_r)$ be a prize-constrained (v, w)-walk with $l_{pc}(W) = d_{pc}(v, w)$. Remove $\{v, w\}$ from S to obtain two new trees. Of these two trees denote the one that contains v by S_v, and the other (containing w) by S_w. Define $b := \min\{k \in \{1, ..., r\} \mid v_k \in V(S_w)\}$ and $a := \max\{k \in \{1, ..., b\} \mid v_k \in V(S_v)\}$. Further, define $x := \max\{k \in \{1, ..., a\} \mid v_k \in T_p^+ \cup \{v\}\}$ and $y := \min\{k \in \{b, ..., r\} \mid v_k \in T_p^+ \cup \{w\}\}$. By definition, $x \leq a < b \leq y$ and furthermore:

$$c_{pc}(W(v_a, v_b)) \leq c_{pc}(W(v_x, v_y)). \tag{12}$$

Reconnect S_v and S_w by $W(v_a, v_b))$, which yields a new tree S'. For this tree it holds that:

$$\begin{aligned}
C(S') &\leq C(S) + c_{pc}(W(v_a, v_b)) - c(\{v, w\}) \\
&\overset{(12)}{\leq} C(S) + c_{pc}(W(v_x, v_y)) - c(\{v, w\}) \\
&\leq C(S) + l_{pc}(W) - c(\{v, w\}) \\
&= C(S) + d_{pc}(v, w) - c(\{v, w\}) \\
&\overset{(11)}{<} C(S).
\end{aligned}$$

Because of $C(S') < C(S)$ no optimal solution can contain $\{v, w\}$. $\square$

To obtain a criterion for the case of equality in (11), define $d_{pc}^-(v, w)$ as the prize-constrained distance with respect to the PCSTP $(V, E \setminus \{e\}, c, p)$, with $e := \{v, w\}$. If v and w are disconnected in $(V, E \setminus \{e\}, c, p)$, define $d_{pc}^-(v, w) := \infty$. With this definition at hand, one obtains the following corollary to Theorem 6.

Corollary 7. *Let $e = \{v, w\} \in E$. If*

$$c(e) \geq d^-_{pc}(v, w) \tag{13}$$

is satisfied, then e is not contained in at least one optimal solution to I_{PC}.

Figure 1 shows a PCSTP instance on which Theorem 6 allows one to eliminate an edge. Only vertex v_4 has a non-zero prize. Consider the (dashed) edge $\{v_1, v_2\}$. For the prize-constrained (v_1, v_2)-walk $W = (v_1, \{v_1, v_3\}, v_3, \{v_3, v_4\}, v_4, \{v_3, v_4\}, v_3, \{v_2, v_3\}, v_2)$ it holds that $d_{pc}(v_1, v_2) = l_{pc}(W) = 6$.

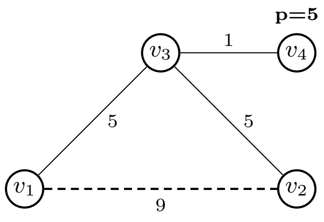


Figure 1: PCSTP instance. Edge $\{v_1, v_2\}$ (dashed) can be eliminated due to Theorem 6.

Algorithms for the prize-constrained distance

Since computing the bottleneck Steiner distance is already $\mathcal{NP}$-hard [50], 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 us to design a simple algorithm for finding upper bounds that yields empirically strong results—significantly better than those of the more involved bottleneck Steiner distance heuristics [45]. In particular, while the bottleneck Steiner distance heuristics in both [45] and [50] consider only paths with at most two intermediary potential terminals, the following algorithm can find walks where the number of intermediary potential terminals is only bounded by $|T_p|$. Besides the (more time-consuming) dual-ascent [45], the bottleneck Steiner distance (through the use of heuristics) has been the most important reduction concept for PCSTP [22, 45, 50]. Due to this importance, we will take a deeper look at the prize-constrained distance, as well as at associated algorithms for computing upper bounds. We will denote the bottleneck Steiner distance by B in the following, as in the original publication [50]; the definition of the bottleneck Steiner distance is given in Appendix A.1.

To check whether an edge $\{v, w\}$ can be deleted by means of criterion (13), we suggest the procedure detailed in Algorithm 1, which is based on Dijkstra's algorithm [13]. The algorithm is given the edge $e = \{v, w\}$ as well as one of its endpoint, say v, from which it computes prize-constrained walks of length not higher than $c(e)$. The algorithm starts with a priority queue that contains v. In contrast to Dijkstra's algorithm, all vertices except for potential terminals and v can be reinserted into the priority queue after they have been removed. We associate with each vertex u the distance $dist_{pc}[u]$—initially set to 0 for v and to ∞ otherwise. As with Dijkstra's algorithm, in each iteration one vertex u with minimum distance value is removed from the priority queue and neighboring vertices of u are updated. However, a different distance value than in Dijkstra's algorithm is used and a neighboring vertex q is only updated if $dist_{pc}[u] + c(\{u, q\}) \leq c(e)$. Throughout the computation the following invariant is satisfied for any $u \in V \setminus \{v\}$: either $dist_{pc}[u] = \infty$ or $dist_{pc}[u] + p(u) \leq c(e)$, and in the latter case there exists a prize-constrained (v, u)-walk W_u such that $l_{pc}(W_u) \leq c(e)$.

```
Data: PCSTP $(V, E, c, p)$, edge $\{v_{start}, v_{end}\} \in E$
Result: deletable if edge has shown to be redundant, unknown otherwise
$Q := \{v_{start}\}$;
$E_0 := E \setminus \{\{v_{start}, v_{end}\}\}$;
$c_0 := c(\{v_{start}, v_{end}\})$;
foreach $v \in V \setminus \{v_{start}\}$ do
    $dist_{pc}[v] := \infty$;
    $forbidden[v] := false$;
end
$dist_{pc}[v_{start}] := 0$;
$forbidden[v_{start}] := true$;
while $Q \neq \emptyset$ do
    $v := \arg\min_{u \in Q} dist_{pc}[u]$;
    $Q := Q \setminus \{v\}$;
    if $v \in T_p$ then
        $forbidden[v] := true$;
    end
    foreach $w \in V$ with $\{v, w\} \in E_0$ do
        if $forbidden[w] = false$ and $dist_{pc}[v] + c(\{v, w\}) \leq c_0$ and
         $dist_{pc}[v] + c(\{v, w\}) - p(w) < dist_{pc}[w]$ then
            if $w = v_{end}$ then
                return deleteable;
            end
            $dist_{pc}[w] := \max\{0, c(\{v, w\}) + dist_{pc}[v] - p(w)\}$;
            if $w \notin Q$ then
                $Q := Q \cup \{w\}$;
            end
        end
    end
end
return unknown;
```

Algorithm 1: Checking whether a given edge can be deleted.

One obtains the following results:

Proposition 8. *For any two vertices $v, w \in V$ it holds that*

$$d_{pc}(v, w) \leq B(v, w). \tag{14}$$

Furthermore, let $\mathcal{I}_{pc}$ be the set of all PCSTP instances. It holds that

$$\sup_{(V,E,p,c) \in \mathcal{I}_{pc}} \max_{v,w \in V} \frac{B(v, w)}{d_{pc}(v, w)} = \infty. \tag{15}$$

Proof. The relation (14) can be verified by the definitions of B and d_{pc}. For the second part consider the PCSTP depicted in Figure 2. Let $n \in \mathbb{N}, n \geq 3$. The prizes p and costs c are defined as follows: $p(v) = 0$ for $i = 0, ..., n$, and $p(w_i) = 3$ for $i = 1, ..., n-1$; further, $c \equiv 1$. Observing that

$$\lim_{n \to \infty} \frac{B(v_0, v_n)}{d_{pc}(v_0, v_n)} = \lim_{n \to \infty} \frac{n}{3} = \infty,$$

one can validate that (15) holds. □

Next, we show that (the heuristic) Algorithm 1 can yield better results than the (exact) bottleneck Steiner distance. To this end, first define B^- analogously to d^-_{pc}. The next corollary shows that the results from Algorithm 1 can be (in a relative sense) arbitrarily better than the bottleneck Steiner distance.

Corollary 9. *For any $K \in \mathbb{N}_0$ there is a PCSTP instance I_K with an edge $e = \{v, w\}$ such that*

$$\frac{B^-(v,w)}{c(e)} \geq K,$$

while Algorithm 1 returns deletable for $(I_K, \{v, w\})$.

Proof. If $K = 0$, condition (9) is trivially satisfied. Assume $K \geq 1$, and consider the PCSTP instance in Figure 2 for $n = 3K$. Add an arc $\{v_0, v_n\}$ of cost 3 to the instance. For this PCSTP together with the edge $\{v_0, v_n\}$ the PCD algorithm returns *deletable*. On the other hand, $B^-(v_0, v_n) = 3K$. □

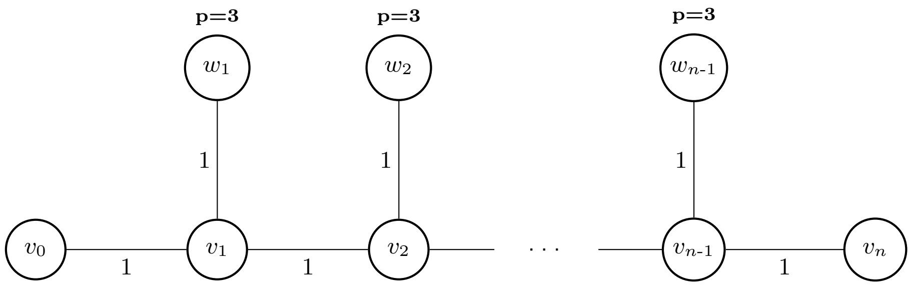


Figure 2: PCSTP instance such that the ratio of the bottleneck Steiner distance and the prize-constrained distance between v_0 and v_n becomes arbitrarily large.

Since Algorithm 1 runs in polynomial time and the decision variant of the prize-constrained distance is $\mathcal{NP}$-complete, Algorithm 1 cannot be in general exact—in the sense that it might return *unknown* even though $c(\{v_{start}, v_{end}\}) \geq d^-_{pc}(v_{start}, v_{end})$—unless $\mathcal{P} = \mathcal{NP}$. However, under certain conditions Algorithm 1 is exact, as detailed in the following proposition (see Appendix A.3 for a proof).

Proposition 10. *Let $\{v, w\} \in E$ and let W be a (v, w)-walk with $l_{pc}(W) = d^-_{pc}(v, w)$. If for all $t \in (V(W) \setminus \{v, w\}) \cap T^+_p$*

$$p(t) \leq \min_{e \in \delta(t) \cap E(W)} c(e) \tag{16}$$

holds, then Algorithm 1 returns deletable if and only if $c(\{v, w\}) \geq d^-_{pc}(v, w)$.

Corollary 11. *Let $v, w \in V$. If $T^+_p \subseteq \{v, w\}$ holds (which includes $T^+_p = \emptyset$), then both $d^-_{pc}(v, w)$ and $d_{pc}(v, w)$ can be computed in polynomial time (with respect to the encoding size of I_{PC}).*

To check whether an edge e can be eliminated, we run a restricted version of Algorithm 1 (which only checks at most a fixed number of edges during its execution) from both endpoints of e. If none of the two tests are successful, we check for each vertex that has been visited in both runs whether the corresponding walks can be combined to obtain a walk that allows to delete e. This procedure will be referred to as *prize-constrained distance* (*PCD*) *test.*

We have also implemented an extension of PCD, referred to as *extended prize-constrained distance* (*EPCD*) *test*, which will be sketched in the following. One downside of PCD, even in its unrestricted form, is that once a potential terminal has been removed from the priority queue, it cannot be used in any other walk. Thus, EPCD keeps for each vertex v a (bounded) list of potential terminal that are part of the current (v_{start}, v)-walk. Whenever a vertex w could be updated from a vertex v, but $forbidden[w] = false$, it is checked whether w is in the potential terminal list of v, and if not, w is still updated. Another problem of PCD is that the cost of an edge on a subwalk might be counted several times. Consider for example the instance described in the proof of Corollary 9 and change the prizes for all w_i to $p(w_i) := 2$. It still holds that $d^-_{pc}(v_0, v_n) = 3$, but PCD for edge $\{v_0, v_n\}$ will only return *deletable* if $c(\{v_0, v_n\}) \geq n$. Therefore, EPCD saves for each vertex (a limited number of) edges on the current subwalk. This list is cleared as soon as the distance value of a vertex is set to 0. EPCD also allows that non-proper potential terminals can be used several times on one walk, by keeping a similar list of non-proper potential terminals on the current subwalk. Finally, we note that one can also adapted other test from the SPG literature, such as *NSV* [17], and NTD_k [18] to PCSTP by using the prize-constrained bottleneck distance.

3.2 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 [17, 41]. In the following, we will introduce an approach that improves on previous results both for PCSTP and SPG. For any $U \subseteq V$ such that $T_p^+ \subseteq U$, and for any $v_i, v_j \in V$ let $\mathcal{Q}_U(v_i, v_j)$ be the set of all (v_i, v_j)-paths in the graph induced by $V \setminus (U \setminus \{v_i, v_j\})$. Define $\underline{d}_U : V \times V \mapsto \mathbb{Q}_{\geq 0} \cup \{\infty\}$ as

$$\underline{d}_U(v_i, v_j) := \inf_{Q \in \mathcal{Q}_U(v_i, v_j)} \sum_{e \in E(Q)} c(e) - \sum_{v \in V \setminus \{U \cup \{v_i\}\}} p(v),$$

with the common convention $\inf \emptyset := \infty$. Let $v_i \in V$. Define $\underline{v}^U_{i,0} := v_i$, and, recursively, for $k \in \mathbb{N}$

$$\underline{v}^U_{i,k} := \arg\min\{\underline{d}_U(v_i, v) \mid v \in U \setminus \cup_{j=0}^{k-1}\{\underline{v}^U_{i,j}\}\}, \tag{17}$$

assuming that such a vertex exists.

With these definitions at hand, we introduce the following concept: a *terminal-regions decomposition* of I_{PC} is a partition $(H_0, H_1, H_2, ..., H_{s^+})$ of V such that for $i = 1, ..., s^+$ it holds that $T_p^+ \cap H_i = \{t_i^+\}$ and that the subgraph induced by H_i is connected—recall that $T_p^+ = \{t_1^+, t_2^+, ..., t_{s^+}^+\}$. Note that H_0 does not need to be connected. Define $H^p := T_p^+ \cup (H_0 \cap T_p^-)$. Further, define $r_H^{pc} : T_p^+ \mapsto \mathbb{Q}_{\geq 0}$ by

$$r_H^{pc}(t_i^+) := \min\left\{p(t_i^+), \min\{\underline{d}_{H^p}(t_i^+, v) \mid v \notin H_i\}\right\} \tag{18}$$

for $t_i^+ \in T_p^+$. We will refer to this value as the *prize-collecting radius* of H_i. The terminal-regions decomposition concept generalizes the Voronoi decomposition for the PCSTP introduced in [45]. Furthermore, the decomposition can easily be extended to SPG by using $\min\{d(t_i, v) \mid v \notin H_i\}$ instead of $r_H^{pc}(t_i)$, which corresponds to setting sufficiently high prizes for each terminal of the SPG. Notably, this approach generalizes the SPG Voronoi concept from [41]. In [43] we introduced a related, but coarser, 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 $1 \leq i < j \leq s^+$. Also, assume that in the following a fixed terminal-regions decomposition H is given.

Proposition 12. *Let $v_i \in V \setminus T_p^+$. If $v_i \in V(S)$ for all optimal solutions S, then a lower bound on $C(S)$ is defined by*

$$\underline{d}_{H^p}(v_i, \underline{v}_{i,1}^{H^p}) + \underline{d}_{H^p}(v_i, \underline{v}_{i,2}^{H^p}) + \sum_{k=1}^{s^+-2} r_H^{pc}(t_k^+) + \sum_{t \in T_p^- \setminus \{v_i\}} p(t). \tag{19}$$

A proof of the proposition is given in Appendix A.4. Each vertex $v_i \in V \setminus T_p^+$ with the property that the affiliated lower bound (19) 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 (19) is equal to $C(S)$. A result similar to Proposition 12 can be formulated for edges of an optimal solution.

Figure 3 depicts a PCSTP, the corresponding Voronoi decomposition as described in [45], and another terminal-regions decomposition (found by the TRD test described below). The terminal-regions decomposition shown in Figure 3c yields a stronger lower bound (19) than the Voronoi decomposition in Figure 3b. For the filled vertex the lower bounds are 10 and 11, respectively (10 is also the optimal solution value for the instance). Thus, if a feasible solution of cost smaller than 11 is known (or one of cost 11 that does not contain the filled vertex), Proposition 12 implies that the filled vertex can be deleted.

Another application of the terminal-regions decomposition can be found for PCSTP instances with articulation points. While only a few instances from the literature contain articulation points in their original form, this situation changes once the reduction techniques described so far have been applied. Even more so once branch-and-bound has been initiated, see Section 5.1. Recall that throughout this article I_{PC} is assumed to be connected.

First, one observes that if a biconnected component $B \subseteq G$ satisfies $T_p^+ \subseteq V(B)$ and no trivial solution exists, then there is at least one optimal solution S with $S \subseteq B$. The opposite case, namely that $T_p^+ \nsubseteq V(B)$, is not as straightforward and requires some groundwork. In the remainder of this section it will be assumed that I_{PC} contains at least one biconnected component that does not contain all proper potential terminals. Let $B \subseteq G$ be a biconnected component and let $R \subseteq G$ be a connected subgraph such that $E(R) \cap E(B) = \emptyset$. A vertex $v \in B$ such that there is a path Q from v to R with $V(Q) \cap V(B) = \{v\}$ will be called *gate vertex* from B to R.

Lemma 13. *Let $B \subseteq G$ be a biconnected component and $R \subseteq G$ a nonempty, connected subgraph such that $E(R) \cap E(B) = \emptyset$. There exists exactly one gate vertex from B to R.*

Based on Lemma 13 and the terminal-regions decomposition, the following proposition gives an additional criterion to eliminate biconnected components (see Appendix A.5 for a proof).

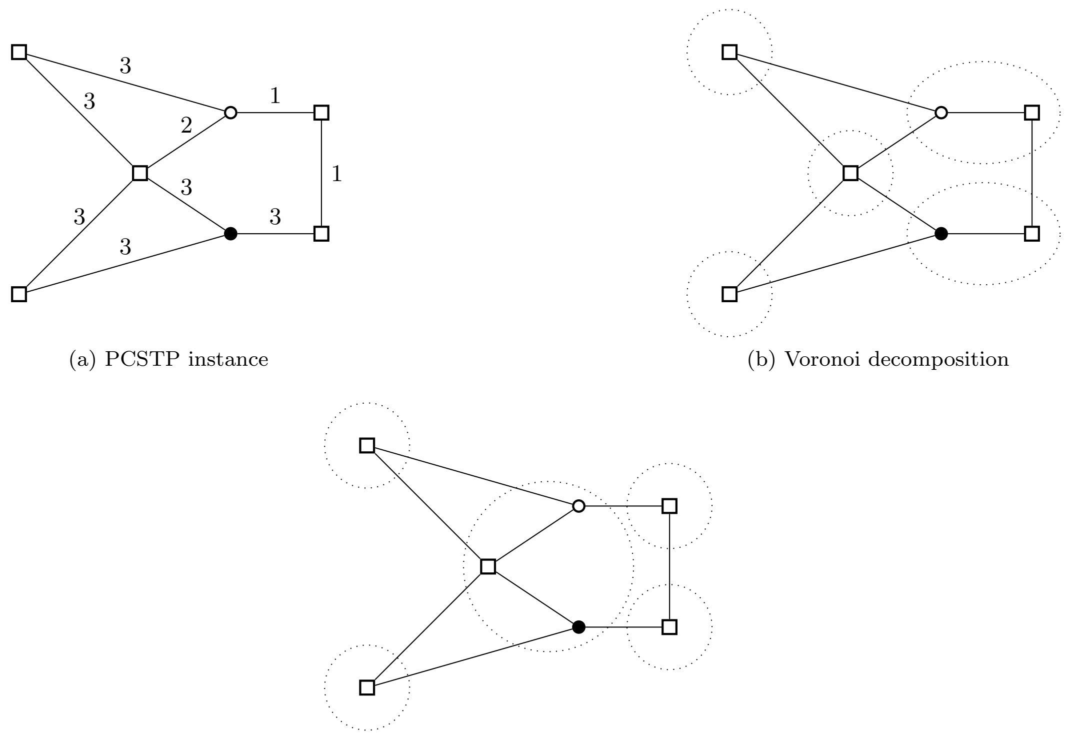
Proposition 14. *Let $B \subseteq G$ be a biconnected component with $T_p^+ \nsubseteq V(B)$. Let $R \subseteq G$ be a connected subgraph with $T_p^+ \setminus V(B) \subseteq V(R)$ and assume that $E(R) \cap E(B) = \emptyset$. If $T_p^+ \cap V(B) = \emptyset$, then there is an optimal solution that does not contain any edge of B. Otherwise, let $v_i \in V(B)$ be the gate vertex from B to R, let H be a terminal-regions decomposition of B, and define $\underline{d}_{H^p}$ and $\underline{v}_{i,1}^{H^p}$ with respect to B. Let $X := V(B) \setminus \{v_i\}$ and define*

$$L := \underline{d}_{H^p}(v_i, \underline{v}_{i,1}^{H^p}) + \sum_{t \in X \cap T_p^+} r_H^{pc}(t) - \max_{t \in X \cap T_p^+} r_H^{pc}(t) + \sum_{t \in X \cap T_p^-} p(t). \tag{20}$$

If

$$L \geq \sum_{v \in X} p(v) \tag{21}$$

holds, then at for at least one solution S to I_{PC} it holds that either $S \subseteq B$ or $E(S) \cap E(B) = \emptyset$.



(a) PCSTP instance (b) Voronoi decomposition

(c) terminal-regions decomposition

Figure 3: Illustration of a PCSTP instance (a), a Voronoi decomposition (b), and a 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.

The set R in Proposition 14 can for example be computed (or its non-existence can be shown) by a depth-first-search on the graph $(V, E \setminus E(B))$ starting from any $t \in T_p^+ \setminus V(B)$. If (21) holds, it might still be the case that $S \subseteq B$ for all optimal solutions S. This case can for example be ruled out if a feasible solution $S' \subsetneq B$ is known that satisfies $C(S') \leq \sum_{v \in V \setminus V(B)} p(v)$ (or by more sophisticated criteria involving the terminal-regions decomposition of B). If such a S' is known, one can eliminate all edges of B from I_{PC}.

To efficiently apply the previous two propositions, one would like to maximize the lower bounds (19), and (20) respectively. However, as shown in Appendix A.6 and similar to a result from [43], one obtains

Proposition 15. *Given a $v_i \in V \setminus T_p^+$, finding a terminal-regions decomposition that maximizes* (19) *is $\mathcal{NP}$-hard. The same holds for* (20).

Thus, to compute a terminal-regions decomposition a heuristic approach will be used. Note that the Voronoi decomposition method [45] yields a terminal-regions decomposition (by setting $H_0 := T_p^-$) with the same bound (19), but in general not an optimal one. By using a simple local search heuristic that checks edges between different regions and fully includes them in one of the regions if advantageous, it is usually possible to improve the decomposition provided by the

Voronoi method. In most cases one obtains even stronger results with the following adaptation of Dijkstras algorithm to compute a terminal-regions decomposition from scratch: Put all $t_i^+ \in T_p^+$ in the initial priority queue (with distance value 0). Similar to Algorithm 1, we subtract from the distance value of each vertex $v \in V \setminus T_p^+$ its prize $p(v)$ when it is updated. Moreover, the algorithm does not extend a region H_i from a vertex $v \in H_i$ if an upper bound $\bar{b}_i \in \mathbb{Q}_{\geq 0}$ on $r_H^{pc}(t_i^+)$ is already known and $\underline{d}_{H^p}(t_i^+, v) \geq \bar{b}_i$. Such upper bounds can be computed during the execution of the algorithm. Subsequently, we apply the above mentioned local heuristic. This procedure will be called *terminal-regions decomposition* (TRD) test. The corresponding reduction method for biconnected components will be referred to as *component terminal-regions decomposition* (CTRD) test.

4 Changing the problem class

A successful approach for the exact solution of the PCSTP is its transformation to some variant of a directed Steiner tree problem [37, 39, 44], followed by the solution of a corresponding IP or MIP formulation. This section uses the reduced-costs of the LP relaxation of a particular IP formulation to find vertices that need to be part of any optimal solution—which allows for another change of the problem class. The algorithmic part of this section is clustered around a combinatorial criterion that associates with each proper potential terminal t a set of vertices such that any optimal solution that contains one of these vertices also contains t.

4.1 Identifying roots

A cornerstone of the 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 [52] for an SAP (V, A, T, c, r):

Formulation 1. *Directed cut (DCut)*

$$\begin{aligned} min \quad & c^T x & & & (22)\\ x(\delta^-(U)) & \geq 1 & & \text{for all } U \subset V,\ r \notin U, U \cap T \neq \emptyset, & (23)\\ x(a) & \in \{0,1\} & & \text{for all } a \in A. & (24) \end{aligned}$$

In [52] also a dual-ascent algorithm for *DCut* was introduced that can 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 [35, 41] and use *DCut*, but furthermore such transformations can provide stronger LP relaxations, both theoretically and practically [17, 25]. For the PCSTP such a transformation is described in [44]. We additionally apply *cost-shifting* [17, 37] in Step 2.

Transformation 2 (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* $M := \sum_{t \in T_p^+} p(t)$.

2. *Define* $c' : A' \to \mathbb{Q}_{\geq 0}$ *for all* $a = (v, w) \in A'$ *by*

$$c'(a) := \begin{cases} c(\{v,w\}) - p(w) & \textit{if } w \in T_p^-, \\ c(\{v,w\}) & \textit{otherwise.} \end{cases}$$

3. *Add vertices* r' *and* v_0' *to* V'.
4. *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'.
5. *Define set of terminals* $T' := \{t_1', ..., t_{s^+}'\} \cup \{r'\}$.
6. ***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 make it possible to connect t_i' from any original potential terminal t_j with cost $p(t_j)$. See Figure 5b for an example. Note that one needs to compare any solution obtained by the above transformation with the best single vertex tree in order to not miss a trivial optimal solution.

Each optimal solution to the SAP obtained from Transformation 2 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 2 on I_{PC}:

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

$$\begin{aligned} \min \quad & {c'}^T x - M + p(T_p^-) && && (25) \\ & x \textit{ satisfies } (23), (24) && && (26) \\ & y(\{v,w\}) = x((v,w)) + x((w,v)) && \textit{for all } \{v,w\} \in E && (27) \\ & y(e) \in \{0,1\} && \textit{for all } e \in E. && (28) \end{aligned}$$

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.

Implied potential terminals

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 the original PCSTP. To this end, consider a PCSTP and let $v, w \in V$. Further, let $W = (v_1, e_1, v_2, e_2, ..., e_r, v_r)$ with $v_1 = v$ and $v_r = w$ be a prize-constrained (v, w)-walk (as defined in Section 3). Define the *left-rooted prize-constrained length* of W as:

$$\vec{l}_{pc}(W) := \max\{c_{pc}(W(v, v_i)) \mid v_i \in V(W) \cap (T_p^+ \cup \{w\})\}. \tag{29}$$

Furthermore, define the *left-rooted prize-constrained* (v, w)*-distance* as:

$$\vec{d}_{pc}(v, w) := \min\{\vec{l}_{pc}(W) \mid W \in \mathcal{W}_{pc}(v, w)\}. \tag{30}$$

Note that in general $\vec{d}_{pc}$ is not symmetric. Definition (30) gives rise to

Theorem 16. *Let $v, w \in V$. If*

$$p(v) > \vec{d}_{pc}(v, w) \tag{31}$$

is satisfied, then every optimal solution that contains w also contains v.

Proof. Let S be a tree with $w \in V(S)$ and $v \notin V(S)$. Further, let $W = (v_1, e_1, ..., e_r, v_r)$ be a prize-constrained (v, w)-walk with $\vec{l}_{pc}(W) = \vec{d}_{pc}(v, w)$ and define $a := \min\{k \in \{1, ..., r\} \mid v_k \in V(S)\}$ and $b := \min\{k \in \{a, a + 1, ..., r\} \mid v_k \in T_p^+ \cup \{w\}\}$. Note that

$$c_{pc}(W(v, v_a)) \leq c_{pc}(W(v, v_b)). \tag{32}$$

Add the subgraph corresponding to $W(v, v_a)$ 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:

$$\begin{aligned} C(S') &\leq C(S) + c_{pc}(W(v, v_a)) - p(v) \\ &\overset{(32)}{\leq} C(S) + c_{pc}(W(v, v_b)) - p(v) \\ &\leq C(S) + \vec{l}_{pc}(W) - p(v) \\ &= C(S) + \vec{d}_{pc}(v, w) - p(v) \\ &\overset{(31)}{<} C(S). \end{aligned}$$

The relation $C(S') < C(S)$ implicates that any optimal solution that contains w also contains v. □

Corollary 17. *Let $v, w \in V$. If*

$$p(v) \geq \vec{d}_{pc}(v, w) \tag{33}$$

is satisfied and w is contained in an optimal solution, then v is also part of an optimal solution.

The left-rooted prize-constrained distance can be exemplified by means of Figure 4. It holds that $\vec{d}_{pc}(v_0, v_5) = 2$, but $\vec{d}_{pc}(v_5, v_0) = 3$. A walk corresponding to $\vec{d}_{pc}(v_0, v_5)$ is $(v_0, \{v_0, v_1\}, v_1, \{v_1, v_2\}, v_2, \{v_2, v_1\}, v_1, \{v_1, v_3\}, v_3, \{v_3, v_4\}, v_4, \{v_4, v_5\}, v_5)$. Corollary 17 implies that if v_5 is part of an optimal solution, then there is an optimal solution that contains v_0. The converse does not necessarily hold. Indeed, v_5 is not part of any optimal solution even though v_0 is (together with v_2 and v_3).

As for d_{pc}, computing $\vec{d}_{pc}$ is $\mathcal{NP}$-hard (which can be shown analogously). However, one can devise a simple algorithm for finding upper bounds, which is very similar to Algorithm 1. Let $t_0 \in T_p^+$. The subsequently sketched algorithm provides a set of vertices $\bar{T}_{t_0}$ such that $\vec{d}_{pc}(t_0, v) < p(t_0)$ for all $v \in \bar{T}_{t_0}$. Initialize $dist_{pcr}[v] := \infty$ for all $v \in V \setminus \{t_0\}$, and set $dist_{pcr}[t_0] := 0$. Start Dijkstra's algorithm with only t_0 in the priority queue, but apply the following modifications: First, update vertex w from vertex u if and only if both $dist_{pcr}[u] + c(\{u, w\}) < p(t_0)$ and

$$dist_{pcr}[w] > dist_{pcr}[u] + c(\{u, w\}) - p(w). \tag{34}$$

In this case set $dist_{pcr}[w] := dist_{pcr}[u] + c(\{u, w\}) - p(w)$. No $t \in T_p$ can be reinserted into the priority queue after they have been removed. Finally, define $\bar{T}_{t_0} := \{u \in V \mid dist_{pcr}[u] < p(t_0)\}$. Note that $t_0 \in \bar{T}_{t_0}$. This algorithm is basically the same as Algorithm 1, except for using $p(v)$ instead of c_0 and using a slightly different update scheme.

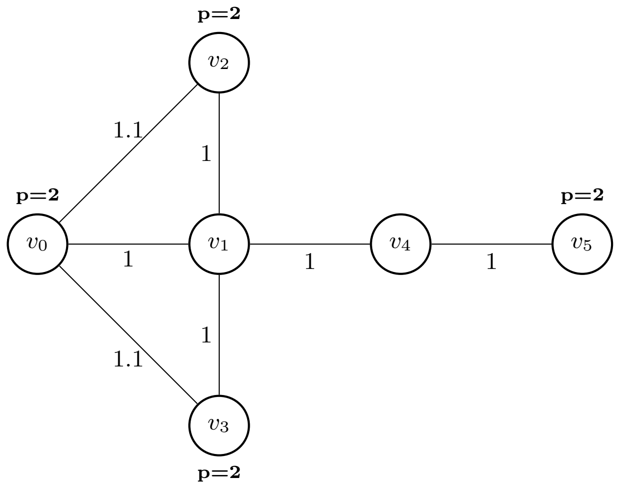


Figure 4: PCSTP instance. The prizes of the individual vertices are specified by p; only non-zero prizes are shown.

Combining implications and reduced costs

By using LP information, the above algorithm can be combined with Transformation 2 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 (23). 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 18. *Consider (V', A', T', c', r') obtained by applying Transformation 2 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:*

$$\begin{aligned} min \quad & {c'}^T x - M + p(T_p^-) & & (35)\\ x(\delta^-(U)) & \geq 1 & \text{for all } U \in \tilde{\mathcal{U}}, & \quad (36)\\ x(a) & \in [0,1] & \text{for all } a \in A'. & \quad (37) \end{aligned}$$

Moreover, let K 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

$$\sum_{j \mid t_j \in \bar{T}_i} \tilde{c}((v_0', t_j')) + \tilde{L} > K \qquad (38)$$

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 Theorem 16 and using (38), Theorem 16 can again be applied—to directly identify further $t_j \in T_p$ that are part of all optimal solutions by using the condition $p(t_j) > \vec{d}_{pc}(t_j, t_i)$. Identifying such *fixed* terminals can considerably improve the strength of the techniques described in Section 3, which usually leads to further graph reductions and the fixing of additional terminals.

4.2 Rooting the problem: RPCSTP and SPG

Once at least one vertex has been shown to be part of at least one optimal solution, the PCSTP can be reduced to a RPCSTP. Recall that we assume $p(t) = 0$ for all $t \in T_f$. We introduce the following simple transformation (which is a straightforward extension of the single-root RPCSTP transformation from [44]).

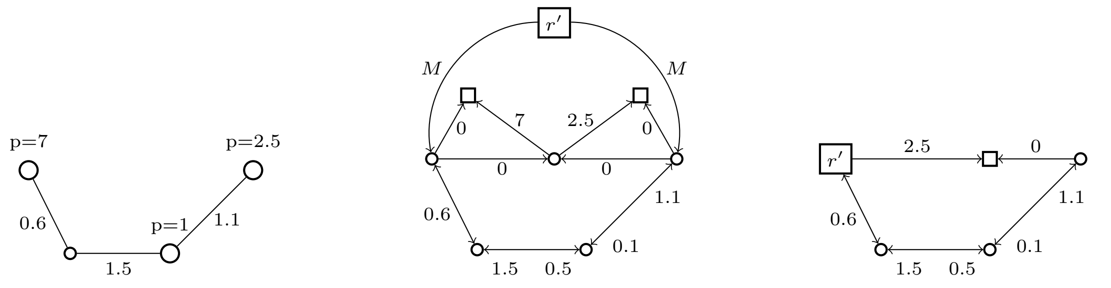
Transformation 3 (RPCSTP to SAP)**.**
 Input: *RPCSTP* (V, E, T_f, c, p) *and* $t_p, t_q \in T_f$
 Output: *SAP* (V', A', T', c', r')

1. *Set* $V' := V$*,* $A' := \{(v, w) \in V' \times V' \mid \{v, w\} \in E\}$*,* $r' := t_q$*.*
2. *Define* $c' : A' \to \mathbb{Q}_{\geq 0}$ *for all* $a = (v, w) \in A'$ *by*

$$c'(a) = \begin{cases} c(\{v, w\}) - p(w) & \textit{if } w \in T_p^-, \\ c(\{v, w\}) & \textit{otherwise.} \end{cases}$$

3. *For each* $i \in \{1, ..., s^+\}$*:*
   (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'*.*
4. *Define set of terminals* $T' := \{t'_1, ..., t'_{s^+}\} \cup T_f$*.*
5. ***Return*** (V', A', T', c', r')*.*

A comparison of Transformation 2 and Transformation 3 is illustrated in Figure 5.



(a) PCSTP instance

(b) SAP instance obtained from Transformation 2

(c) SAP instance obtained from Transformation 3

Figure 5: Illustration of a PCSTP instance (left) and the equivalent SAP obtained by Transformation 2 (middle). Given the information that the potential terminal with weight $p = 7$ is part of at least one optimal solution, Transformation 3 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) 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 3:

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

$$\min\{c'^T x + p(T_p^-) \mid x \textit{ satisfies } (23),(24),\ (x,y) \textit{ satisfies } (27),\ y \textit{ satisfies } (28)\}. \tag{39}$$

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 3. While in [44] the root (t_q) is fixed and $t_p = t_q$ holds, the choice of t_p and t_q in Transformation 3 might potentially affect the value of the LP relaxation $v_{LP}(PrizeRCut(I_{RPC}, t_p, t_q))$. However, this value does not change, and even more:

Theorem 19. *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:*

$$proj_y(R(t_p, t_q)) = proj_y(R(t_{\tilde{p}}, t_{\tilde{q}})). \tag{40}$$

Proof. Let (V, E, T_f, c, p) be the RPCSTP I_{RPC} and denote the SAP resulting from applying Transformation 3 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 theorem 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 3 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 3) 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

$$x(\delta^-_D(U^z)) = \tilde{x}(\delta^-_{\tilde{D}}(U^z)) \leq \tilde{x}(\delta^-_{\tilde{D}}(U)) < 1. \tag{41}$$

Because of $t_q \notin U^z$ and $U^z \cap \tilde{T} \supseteq \{t_{\tilde{p}}\} \neq \emptyset$, one obtains a contradiction from (41). Therefore, $\tilde{x}$ satisfies (23) 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

$$\tilde{x}(\delta^-(U)) \geq x(\delta^-(U)) + 1 \geq 1. \tag{42}$$

Consequently, $\tilde{x}$ satisfies (23) 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, see Section 5.1.

Theorem 19 has also consequences for two well-known IP formulations for SPG and NWSTP. For SPG a widely used formulation is the *bidirected cut formulation* [52]: Replace each edge by two anti-parallel arcs of the same weight as the original edge, and consider the resulting problem as an SAP with an arbitrary terminal as the root. Solve this SAP by Formulation 1. As a direct corollary of Theorem 19 one obtains a result that was already proved by [25].

Corollary 20. *The optimal LP value of the bidirected cut formulation for SPG is independent of the choice of the root.*

For NWSTP a similar formulation has proven effective in practice [22, 37]: Transform the problem to SAP in the same way as for SPG. Additionally, add the weight of each vertex to all its incoming arcs. Note that when transforming an NWSTP to RPCSTP as described in Section 2.2, and applying Transformation 3 to this RPCSTP, one obtains the same SAP as with the procedure described above (given the same choice of the root). Thus, Theorem 19 directly yields the following, new, result.

Corollary 21. *The optimal LP value of the bidirected cut formulation for NWSTP is independent of the choice of the root.*

From the definitions of Transformation 2 and 3 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 in the following proposition; a proof is given in Appendix A.7.

Proposition 22. *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*

$$proj_y(R_{T_0}) \subseteq proj_y(R). \tag{43}$$

Moreover, the inequality

$$v_{LP}(PrizeCut(I_{PC})) \leq v_{LP}(PrizeRCut(I_{T_0})) \tag{44}$$

holds and can be strict.

Finally, by combining the reductions to RPCSTP and SAP with the reductions techniques described in Section 3, it is sometimes possible to either eliminate or fix each potential terminal. Hence the instance becomes an SPG, which allows us to apply a number of additional algorithmic techniques [22, 41].

5 Reduction-based exact solving

In the following, the integration of the individual PCSTP techniques within an exact solving approach will be described. Furthermore, the performance of the resulting solver will be discussed.

5.1 Interleaving the components within branch-and-cut

This section demonstrates that broad applicability of the PCSTP algorithms and techniques introduced so far in a branch-and-cut framework. It also aims to highlight the strong interrelation between the individual techniques.

Presolving The exact solver described in this article is realized within the branch-and-cut based Steiner tree framework SCIP-Jack [22]. SCIP-Jack already includes reduction techniques for PCSTP [45] (in the sense of Section 3), but almost all of them have been replaced by new methods introduced in this article. Besides PCD, EPCD, NSV, TRD, and CTRD (see Section 3), the newly implemented tests include *Non-terminal of Degree k* [50] (by using the new prize-constrained distance).

Probing In addition to the above presolving, we employ a technique similar to the probing [47] 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 can make further graph reductions by means of the above presolving possible.

Domain propagation During the separation phase, SCIP-Jack uses the reduced costs provided by the LP solver and the best known upper bound to perform graph reductions [22]. These reductions can allow for further ones my means of the algorithms described in the presolving paragraph above. Therefore, we also reemploy these reduction techniques for domain propagation (once a predefined number of edges has been deleted by the reduced cost criterion), translating the deletion of edges and the fixing of potential terminals into variable fixings in the IP.

Dual heuristics SCIP-Jack includes the dual-ascent heuristic from [52] that provides a dual solution as well as reduced costs. It is used in presolving (for reduced cost based reduction tests), for primal heuristics (to find a subgraph that contains a good feasible solution), and for computing initial cuts. Whenever a problem has been transformed to RPCSTP, we perform the dual-ascent heuristic on several SAPs resulting from different choices of t_p and t_q, which usually changes the lower bound (and the reduced costs) provided by the heuristic.

Primal heuristics SCIP-Jack 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 trigger further graph reductions (e.g. by the terminal-regions decomposition) or to fix additional terminals (by means of Proposition 18). Additionally, we have implemented a new primal heuristic that starts with a single (potential or fixed) terminal and connects other proper potential terminals t_i to the current subtree S if $\min_{v \in V(S)} \vec{d}_{pc}(t_i, v) \leq p(t_i)$ (only upper bounds on $\vec{d}_{pc}$ are used). At each iteration a $t_i \in T_p^+ \setminus V(S)$ is chosen such that $p(t_i) - \vec{d}_{pc}(t_i, v)$ is maximized. A similar approach has been implemented as a local search heuristic.

LP and cutting-planes 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 [45]—which can even enable further prize-constrained walk based reductions. Moreover, besides the separation of (23), already implemented in SCIP-Jack, we also separate constraints for *TransRCut* of the form

$$x(\delta^-(v)) + x((t_p, t_i')) \leq 1 \qquad t_i \in T_p^+ \setminus T_f, v \in \{u \in V \mid \vec{d}_{pc}(t_i, u) < p(t_i)\},$$

with t_p and t'_i as defined in Transformation 3. The constraints represent the implication that $v \in V(S) \Rightarrow t \in V(S)$ for any optimal solution S if $\vec{d}_{pc}(t,v) < p(t)$. Corresponding constraints are separated for $TransCut$.

Restart In the course of the solution process one can regularly either delete or fix each potential terminal—through the combination of presolving, primal heuristics, the left-rooted prize-constrained distance, and graph transformation and LP methods. In such a case one might restart the solution process and use the SPG solver of SCIP-Jack [22]. The following restricted restart strategy turned out to be empirically successful: If a PCSTP instance is transformed to SPG at the root node of the branch-and-bound tree, we run aggressive SPG presolving and translate it into variable fixings in the IP formulation (although it should be noted that not all presolving reductions can be translated into variable fixings). In the remainder of the solution process SPG specific primal heuristics and reduction techniques are used, but the remaining algorithmic components are left unchanged. If all potential terminals are fixed already during presolving, a full restart is initiated, including full SPG-specific presolving, and the instance is handled entirely as an SPG.

Branching Finally, branching is performed on vertices—by rendering the vertex 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. Branching on vertices is already described for SPG in the literature [41]. As in probing, the implications from the left-rooted prize-constrained distance often set the state for further graph changes, resulting in (local) variable fixings.

5.2 Computational results

To the best of our knowledge, the three currently fastest exact algorithms for PCSTP are *mozartballs* [20] (the winner of the exact PCSTP categories at the 11th DIMACS Challenge), SCIP-Jack [22], and *dapcstp* [37]. While no solver dominates on all benchmarks, the branch-and-bound solver *dapcstp* is very competitive, and on several test-sets orders of magnitude faster than mozartballs—it is even faster than state-of-the-art heuristic methods [21]. Thus, *dapcstp*, which is publicly available [36], will in the following be used for comparison. Furthermore, we will compare the new PCSTP solver against SCIP-Jack without the new methods introduced in this article. We will refer to this version as SCIP-Jack-base. Notably, SCIP-Jack-base is faster than the version of SCIP-Jack described in [22, 45], since the implementation of existing methods (e.g. data structures and cache-efficiency) has been improved.

The computational experiments were performed on Intel Xeon E3-1245 CPUs with 3.40 GHz and 32 GB RAM. For both SCIP-Jack-base and the new solver, CPLEX 12.8 [29] is employed as underlying LP solver. Furthermore, only single-thread mode is used, as *dapcstp* does not support multiple threads. For the following experiments 12 well-known benchmark test-sets are used, as detailed in Table 1. ACTMOD and HIV contain originally MWCSP and NWSTP instances, which have been transformed to PCSTP by using the methods described in Section 2.2.

Strength of preprocessing

Table 2 shows the strength of the original PCSTP reduction algorithms of SCIP-Jack [45], and the improved algorithms developed as part of this article. We provide the average number of remaining vertices and edges per test set, computed as the arithmetic mean. It can be seen that the new techniques are considerably stronger for most tests sets that are not completely or almost completely reduced by the previous techniques. An exception are the test sets PUCNU

Table 1: Details on PCSTP tests sets.

Name	Instances	$\|V\|$	$\|E\|$	Status	Description
Cologne	29	741 - 1810	6332 - 16794	solved	Instances from fiber optic network design for German cities [38].
CRR	80	500 - 1000	625 - 25000	solved	Mostly sparse instances, based on C and D test-sets of the SteinLib [38].
ACTMOD	8	2034 - 5226	3335 - 93394	solved	Instances from integrative biological network analysis [15].
RANDOM	68	200 - 14000	1575 - 112369	solved	Randomly generated instances originally published in [6].
E	40	2500	3125 - 62500	solved	Mostly sparse instances originally for SPG, introduced in [38].
HANDS	20	39600 - 42500	78704 - 84475	solved	Images of hand-written text derived from a signal processing problem [14].
HANDB	28	158400 - 169800	315808 - 338551	unsolved	
I640	100	640	960 - 204480	unsolved	Random graphs with incidence weights. Introduced at 11th DIMACS Challenge.
PUCNU	18	64 - 4096	192 - 28512	unsolved	Instances derived from PUC set for SPG. From 11th DIMACS Challenge.
H2	14	64 - 4096	192 - 24576	unsolved	Hard instances based on hypercubes. Introduced at 11th DIMACS Challenge.
HIV	2	386 - 205717	1477 - 2466001	unsolved	HIV mutation networks. Introduced at 11th DIMACS Challenge.
MA	20	1000000	10000000	unsolved	Random instances with $T_p = V$. Introduced in [49].

and H2. However, these sets consist of instances that have been designed or carefully handpicked to defy reduction techniques (as well as linear programming based algorithms), see [46].

Previous PCSTP reduction techniques are already rather sophisticated, see e.g. [39, 45, 50], and especially dual-ascent based methods [37, 44] can have a large impact. The new reduction algorithms developed in this article, especially PCD and EPCD, have in several cases a drastic impact on instances that are still quite large after the application of the methods from [45]. One example is the hardest instance from the E set, *E18-B*, which is reduced from around 11 thousand to less than six thousand edges. Similarly, the size of several of the hardest MA instances is more than halved. Finally, the slightly worse performance of the new reductions on the I640 instances can be explained by the fact that the new reductions algorithms are executed less exhaustively than their predecessors (to improve the overall run-time).

Table 2: Average problem size after application of original and improved reduction algorithms.

	original reductions		new reduction	
Class	Vertices[%]	Edges[%]	Vertices[%]	Edges[%]
Cologne	0	0	0	0
CRR	2.9	0.3	**1.6**	**0.1**
ACTMOD	0.1	0.0	0.1	0.0
HANDS	0.1	0.1	0.1	**0.0**
RANDOM	2.2	0.6	**1.0**	**0.3**
E	7.2	1.8	**5.4**	**0.6**
HANDBD	5.8	5.8	**3.1**	**3.1**
I640	**37.2**	**31.5**	38.9	32.5
PUCNU	74.9	65.1	**72.6**	**63.3**
H2	92.4	92.3	**91.1**	**89.0**
HIV	10.1	9.1	**0**	**0**
MA	5.3	7.0	**2.3**	**2.5**

Branch-and-cut results, and comparisons

This subsection provides computational results of the entire branch-and-cut framework developed for this article. Table 3 provides aggregated results of the experiments with a time limit of two hours for test-sets that contain only instances with less than a million edges, and a time limit of 24 hours for the remaining (two) sets. The first column shows the test-set considered in the current row. The second columns shows the number of instances in the test-set. Columns three to five show the shifted geometric mean [1] (with shift 1) of the run time taken by the respective solvers: First, *dapcstp*, second, SCIP-Jack-base, third, the new solver. The next three columns provide the maximum run time, the last thee columns the number of solved instances.

Table 3: Computational comparison of the solvers *dapcstp* [37], SCIP-Jack-base, denoted by *base*, and the solver described in this article, denoted by *new*.

		mean time [s]			maximum time [s]			# solved		
test-set	#	dapcstp	base	new	dapcstp	base	new	dapcstp	base	new
Cologne	29	0.1	**0.0**	**0.0**	0.2	0.3	**0.1**	29	29	29
CRR	80	0.2	0.2	**0.1**	4.9	1.7	**0.7**	80	80	80
ACTMOD	8	0.8	0.3	**0.1**	3.3	1.4	**0.9**	8	8	8
HANDS	20	2.5	2.2	**0.2**	54.1	14.5	**2.9**	20	20	20
RANDOM	68	0.8	2.6	**0.2**	78.6	>7200	**19.4**	**68**	67	**68**
E	40	2.0	0.6	**0.2**	>7200	76.4	**18.5**	37	**40**	**40**
HANDB	28	32.9	35.0	**6.9**	>7200	>7200	>7200	25	25	**26**
I640	100	9.2	21.4	**7.8**	>7200	>7200	>7200	90	80	**91**
PUCNU	18	441.7	177.1	**58.2**	>7200	>7200	>7200	7	10	**14**
H2	14	**525.5**	2210.0	708.6	>7200	>7200	>7200	5	3	**6**
HIV	2	293.0	293.0	**83.3**	>86400	>86400	**7100.2**	1	1	**2**
MA	20	>86400	>86400	**9080.0**	>86400	>86400	**29869.3**	0	0	**20**

The new solver is on all but one test-set the fastest one, both in terms of the maximum and average run time. Furthermore, it solves 34 more instances than *dapcstp* and 41 more instances than SCIP-Jack-base to optimality. The first three test-sets can be solved within seconds by all solvers, with *dapcstp* being the slowest—up to 7 times slower than the new solver both for the mean and maximum time. One the next two sets, HANDS and RANDOM, the new solver is more than an order of magnitude faster than the respectively slowest solver, both for the maximum and mean run time. On RANDOM, the new solver also solves all instances within 20 seconds, whether SCIP-Jack-base cannot even solve all instances within two hours. For the next five test-sets, the new solver again consistently dominates, with the exception of test-set H2, where *dapcstp* is slightly faster with respect to the shifted geometric mean. However, the new solver manages to solve more instances on each of the five test-sets than *dapcstp*. A striking example is the PUCNU test-set, where the new solver can solve twice as many instances. Finally, for the two large-scale test-sets HIV and MA the new solver also shows a strong performance. On the first test-set, the new solver is able so solve the *hiv-1* instance, which contains more than two million edges, to optimality in less than two hours. The best previously known result from the literature was achieved in a 72 hours run on a large-memory machine [22]. Notably, *hiv-1* has been the largest instance from the 11th DIMACS Challenge solved to optimality so far. For the MA instances, with 10 million edges each, both SCIP-Jack-base and *dapcstp* fail to solve any instance. In contrast, the new solver can solve all of them, some even in less than two hours. To the best of the authors' knowledge these are by far the largest PCSTP instances that have been solved to optimality in the literature to date.

Table 4: Improvements on unsolved DIMACS instances.

Name	gap [%]	new UB	previous UB
hiv-1	**opt**	**656955.33150**	656970.94
cc7-3nu	**opt**	**270**	271
cc10-2nu	**opt**	**167**	168
hc9u2	**opt**	**190**	190
handbd13	0.0	13.18549	13.19699
handbi13	0.1	4.24964	4.251
cc11-2nu	0.8	303	304
cc12-2nu	0.7	563	565
hc9p	1.1	3015	3043
hc9p2	1.4	30228	30242
hc10p	1.4	59778	59866
hc10p2	1.5	59804	59930
hc11p	1.6	118729	119191
hc11p2	1.7	118869	119236
i640-342	0.4	29790	29806
i640-343	0.5	30038	30056
i640-344	0.4	29879	29921
i640-345	0.5	29984	29991

Newly solved instances

Finally, we report on results on previously unsolved instances from the 11th DIMACS Challenge. The results were obtained with a time limit of 24 hours. All improved instances are listed in Table 4, 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.[2] The previously best known solutions are from the 11th DIMACS Challenge and the articles [9, 20, 21, 22, 37], respectively.

Four DIMACS instances can be solved to optimality, three of them, *hiv-1*, *cc10-2nu*, and *hc9u2*, within the standard time limit of two hours. The remaining instance *cc7-3nu* lies just above the time limit, with 8012 seconds. Furthermore, the new solver improves the best known upper bounds for another 14 instances, which comprises more than half of the still unsolved PCSTP instances from the 11th DIMACS Challenge.

6 Conclusion and outlook

This article has introduced a number of techniques and algorithms that aim at faster exact solution of PCSTP. Based on the newly shown fixed-parameter tractability of PCSTP with respect to the number of proper potential terminals, a key element has been the distinction of these vertices within most new algorithms. As an interesting byproduct, we have also demonstrated that any PCSTP can be transformed to an SPG by adding $s + 1$ terminals. Besides the theoretical analyses of the new methods, a central result of this article is the integration of the various methods into an exact branch-and-cut framework. The resulting solver significantly pushes the boundaries of computational tractability for the PCSTP, being able to solve instances with up to 10 million edges—over 30 times larger than any PCSTP instance solved in the literature so far.

A computationally promising route for further research would be to design and implement a PCSTP version of the extended reduction paradigm [42], based on the techniques described here.

[2]The solution of the instance *cc7-3nu* is already noted in the report [24], but is based on techniques described in this article (which have not been published before).

Additionally, further improving the LP relaxation seems to hold a high potential, both from a computational and theoretical point of view. As to theoretical results, one might also further improve the PCSTP complexity bounds.

Finally, the developments described in this article have been integrated into the SCIP-Jack framework and will be published as part of its next release—with open source code and freely available for academic use. In this way, further algorithmic developments can be built on the work described in this article.

7 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). This work was supported by the Research Campus MODAL funded by the German Federal Ministry of Education and Research (05M14ZAM).

References

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

[2] 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.

[3] Eduardo Álvarez-Miranda, Ivana Ljubić, and Petra Mutzel. The maximum weight connected subgraph problem. In *Facets of Combinatorial Optimization*, pages 245–270. Springer Berlin Heidelberg, 2013.

[4] Aaron Archer, MohammadHossein Bateni, MohammadTaghi Hajiaghayi, and Howard Karloff. Improved approximation algorithms for prize-collecting Steiner tree and TSP. *SIAM Journal on Computing*, 40(2):309–332, 2011.

[5] M. Bateni, Chandra Sekhar Chekuri, A. Ene, M. T. Hajiaghayi, N. Korula, and D. Marx. Prize-collecting Steiner problems on planar graphs. In *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011*, pages 1028–1049, 2011.

[6] Indaco Biazzo, Alfredo Braunstein, and Riccardo Zecchina. Performance of a cavity-method-based algorithm for the prize-collecting steiner tree problem on graphs. *Phys. Rev. E*, 86:026706, Aug 2012.

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

[8] 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.

[9] Alfredo Braunstein and Anna Muntoni. Practical optimization of steiner trees via the cavity method. *Journal of Statistical Mechanics: Theory and Experiment*, 2016(7):073302, jul 2016.

[10] Austin Buchanan, Yiming Wang, and Sergiy Butenko. Algorithms for node-weighted steiner tree and maximum-weight connected subgraph. *Networks*, 72(2):238–248, 2018.

[11] 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*, 38(1):50–58, 2001.

[12] Alexandre Salles da Cunha, Abilio Lucena, Nelson Maculan, and Mauricio G.C. Resende. A relax-and-cut algorithm for the prize-collecting Steiner problem in graphs. *Discrete Applied Mathematics*, 157(6):1198 – 1217, 2009. Reformulation Techniques and Mathematical Programming.

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

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

[15] Marcus Dittrich, Gunnar Klau, Andreas Rosenwald, Thomas Dandekar, and Tobias Mller. Identifying functional modules in protein-protein interaction networks: An integrated exact approach. *Bioinformatics (Oxford, England)*, 24:i223–31, 08 2008.

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

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

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

[19] Paulo Feofiloff, Cristina G. Fernandes, Carlos E. Ferreira, and José Coelho de Pina. Primal-dual approximation algorithms for the prize-collecting steiner tree problem. *Inf. Process. Lett.*, 103(5):195202, August 2007.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

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

[26] Michel X. Goemans and David P. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Comput.*, 24(2):296317, April 1995.

[27] S. Louis Hakimi. Steiner's problem in graphs and its implications. *Networks*, 1(2):113–133, 1971.

[28] S. C. Hidayati, K. Hua, Y. Tsao, H. Shuai, J. Liu, and W. Cheng. Garment detectives: Discovering clothes and its genre in consumer photos. In *2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*, pages 471–474, March 2019.

[29] IBM. Cplex.

[30] Trey Ideker, Owen Ozier, Benno Schwikowski, and Andrew F. Siegel. Discovering regulatory and signalling circuits in molecular interaction networks. *Bioinformatics*, 18(1):S233–S240, 07 2002.

[31] 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.

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

[33] 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.

[34] P. Klein and R. Ravi. A nearly best-possible approximation algorithm for node-weighted steiner trees. *Journal of Algorithms*, 19(1):104 – 115, 1995.

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

[36] Markus Leitner, Ivana Ljubić, Martin Luipersbeck, and Markus Sinnl. dapcstp.

[37] 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.

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

[39] 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.

[40] 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.

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

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

[43] D. Rehfeldt and T. Koch. Combining NP-Hard Reduction Techniques and Strong Heuristics in an Exact Algorithm for the Maximum-Weight Connected Subgraph Problem. *SIAM Journal on Optimization*, 29(1):369–398, 2019.

[44] 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.

[45] Daniel Rehfeldt, Thorsten Koch, and Stephen J. Maher. Reduction techniques for the prize collecting Steiner tree problem and the maximum-weight connected subgraph problem. *Networks*, 73(2):206–233, 2019.

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

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

[48] 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.

[49] Y. Sun, M. Brazil, D. Thomas, and S. Halgamuge. The fast heuristic algorithms and post-processing techniques to design large and low-cost communication networks. *IEEE/ACM Transactions on Networking*, pages 1–14, 2019.

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

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

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

A Further definitions and proofs

A.1 The bottleneck Steiner distance

The bottleneck Steiner distance for PCSTP was originally introduced in [50]. Let $v, w \in V$ be two distinct vertices. Denote by $\mathcal{P}(v, w)$ the set of all simple paths between v and w. For a simple path P and vertices $x, y \in V(P)$ let $P(x, y)$ be the subpath of P between x and y. Define the *Steiner distance* of path P as

$$SD(P) := \max_{x,y \in V(P)} \sum_{e \in E(P(x,y))} c(e) - \sum_{v \in V(P(x,y)) \setminus \{x,y\}} p(v). \tag{45}$$

With this definition at hand, define the *bottleneck Steiner distance* between v and w as

$$B(v, w) := \min\{SD(P) \mid P \in \mathcal{P}(v, w)\}. \tag{46}$$

A.2 Proof of Theorem 1

The proof of Theorem 1 is based on the well-known dynamic programming algorithm for SPG by [16] that runs in $O(3^{|T|}n + 2^{|T|}n^2 + n^2 \log n + mn)$, where T is the set of terminals. We will refer to this algorithm as *Dreyfus-Wagner* for short. See also [10] for an extension of Dreyfus-Wagner to the node-weighted Steiner tree problem. Dreyfus-Wagner exploits the fact that any optimal Steiner tree S for an SPG (G, T, c), with $T \neq \emptyset$ and positive c, can be split at any $v \in V(S)$ into two non-empty trees S_1 and S_2 such that $T_1 := V(S_1) \cap T \neq \emptyset$, $T_2 := V(S_2) \cap T \neq \emptyset$, and:

1. $T_1 \cap T_2 \subseteq \{v\}$ and $T_1 \cup T_2 = T$,
2. S_1 is optimal for $(G, T_1 \cup \{v\}, c)$, and S_2 is optimal for $(G, T_2 \cup \{v\}, c)$.

We show that a similar property holds for PCSTP. To this end, consider the RPCSTP I_f, defined in Section 2.1. Moreover, we will consider only optimal (PCSTP and RPCSTP) solutions that contain only proper potential or fixed terminals as leafs. If no such solution exists, there is a trivial optimal solution, which can be found in linear time. For any $T \subseteq T_f$ we denote by $I_f(T)$ the RPCSTP (G, T, c, p); so in particular $I_f(T_f) = I_f$. For I_f we obtain the following result.

Lemma 23. *Let $T_1, T_2 \subseteq T_f$ be non-empty with $T_1 \cup T_2 = T_f$. Let S_1, S_2 be trees in G such that all leaves of S_1 are contained in T_1, all leaves of S_2 are contained in T_2, and $S_1 \cap S_2 \neq \emptyset$. In this case, there is a tree $S \subseteq S_1 \cup S_2$ such that $T_f \subseteq V(S)$, and*

$$C(S) \leq C(S_1) + C(S_2) - \sum_{u \in V} p(u) + \min_{u \in V(S_1 \cap S_2)} p(u). \tag{47}$$

Proof. Initially, set $S := S_1 \cup S_2$ and $\hat{S} := S_1 \cap S_2$. Let $v_0 \in V(\hat{S})$ such that $p(v_0) = \min_{u \in V(\hat{S})} p(u)$. Let $\vec{S}_1$ be the arborescence corresponding to S_1 that is rooted in v_0. Denote its arcs by $A(\vec{S}_1)$. For any $w \in V(\hat{S}) \setminus \{v_0\}$ there is an (incoming) arc $(u, w) \in A(\vec{S}_1)$. Let $E'_S := \{\{u, w\} \in E(S_1) \setminus E(S_2) \mid w \in V(\hat{S}) \wedge (u, w) \in A(\vec{S}_1)\}$ and $E''_S := \{\{u, w\} \in E(\hat{S}) \mid (u, w) \in A(\vec{S}_1)\}$. Note that $|E'_S \dot{\cup} E''_S| = |V(\hat{S})| - 1$. Because of $T_p^+ = \emptyset$ it holds that

$$\sum_{e \in E'_S} c(e) + \sum_{e \in E''_S} c(e) \geq \sum_{u \in V(\hat{S}) \setminus \{v_0\}} p(u). \tag{48}$$

Because of $E''_S \subseteq E(\hat{S})$, inequality (48) implies

$$\sum_{e \in E'_S} c(e) + \sum_{e \in \hat{S}} c(e) \geq \sum_{u \in V(\hat{S}) \setminus \{v_0\}} p(u). \tag{49}$$

Note that any $e \in E'_S$ lies in a cycle of S and that each cycle of S contains an $e \in E'_S$. Remove E'_S from S to obtain a new tree $\tilde{S}$ (which contains T_f). It holds that:

$$\begin{aligned}
C(S_1) + C(S_2) &= C(S) + \sum_{e \in \hat{S}} c(e) + \sum_{u \in V} p(u) - \sum_{u \in V(\hat{S})} p(u) && (50)\\
&= C(\tilde{S}) + \sum_{e \in E'_S} c(e) + \sum_{e \in \hat{S}} c(e) + \sum_{u \in V} p(u) - \sum_{u \in V(\hat{S})} p(u) && (51)\\
&\overset{(49)}{\geq} C(\tilde{S}) + \sum_{u \in V} p(u) - p(v_0) && (52)\\
&= C(\tilde{S}) + \sum_{u \in V} p(u) - \min_{u \in V(S_1 \cap S_2)} p(u). && (53)
\end{aligned}$$

Thus, $\tilde{S}$ satisfies (47). □

This lemma sets the stage for the desired result:

Lemma 24. *Let S be an optimal solution to I_f and choose any, arbitrary but fixed, $v \in V(S)$. Further, let $S_1, S_2 \subseteq S$ be trees such that $V(S_1 \cap S_2) = \{v\}$ and $S_1 \cup S_2 = S$. Define $T_1 := (T_f \cap V(S_1)) \cup \{v\}$ and $T_2 := (T_f \cap V(S_2)) \cup \{v\}$. It holds that S_1 is an optimal solution to $I_f(T_1)$, and S_2 to $I_f(T_2)$. Furthermore:*

$$C(S) = C(S_1) + C(S_2) - \sum_{u \in V \setminus \{v\}} p(u) \tag{54}$$

holds.

Proof. First, observe that (54) holds because of $V(S_1 \cap S_2) = \{v\}$. Suppose S_1 is not optimal. Thus, there exists a tree $\tilde{S}_1$ such that all its leaves are contained in T_1 and such that

$$C(\tilde{S}_1) < C(S_1). \tag{55}$$

We also assume that all leaves of S_2 are contained in T_2; note that because of $T_p^+ = \emptyset$ one can always modify S_2 to satisfy this property without increasing $C(S_2)$. By Lemma 23 there exists a $\tilde{S} \subseteq \tilde{S}_1 \cup S_2$ such that $T_f \subseteq V(\tilde{S})$ and

$$\begin{aligned}
C(\tilde{S}) &\leq C(\tilde{S}_1) + C(S_2) - \sum_{u \in V} p(u) + p(v) && (56)\\
&\overset{(55)}{<} C(S_1) + C(S_2) - \sum_{u \in V} p(u) + p(v) && (57)\\
&= C(S), && (58)
\end{aligned}$$

which is a contradiction to the assumption that S is optimal. □

Based on the proceeding lemma, we can apply an extension of Dreyfus-Wagner to solve I_f. We define an slight modification of the prize-collecting cost from Section 3.1. For a (v, w)-walk W let

$$c'_{pc}(W) := \sum_{e \in E(W)} c(e) - \sum_{u \in V(W) \setminus \{w\}} p(u). \tag{59}$$

Let $\mathcal{W}(v, w)$ be the set of all finite walks from v to w and define

$$d'_{pc}(v, w) := \min\{c'_{pc}(W) \mid W \in \mathcal{W}(v, w)\}. \tag{60}$$

Note that if $T_p^+ = \emptyset$, it is sufficient to consider only simple paths instead of walks. The next subsection concludes the proof of Theorem 1.

A.2.1 Proof of Proposition 2

Proof. Initially, choose an arbitrary $t_0 \in T_f$ and set $T_f^- := T_f \setminus \{t_0\}$. For every pair (v, w) of vertices, set

$$g(\{w\}, v) := d'_{pc}(v, w) + \sum_{u \in V \setminus \{w\}} p(u). \tag{61}$$

For $i = 2, ..., |T_f^-|$ define the functions f and g recursively as follows. For $T_i \subseteq T_f^-$ with $|T_i| = i$ set

$$f(T_i, w) = \min_{T \subsetneq T_i | T \neq \emptyset} \big(g(T, w) + g(T_i \setminus T, w) - \sum_{u \in V \setminus \{w\}} p(u)\big) \tag{62}$$

and

$$g(T_i, v) = \min_{u \in V} \big(f(T_i, u) + d'_{pc}(v, u)\big). \tag{63}$$

These values can be computed by a dynamic programming algorithm.

Claim 1: *After the termination of the above dynamic programming algorithm it holds that $g(T_f^-, t_0) = C(S)$ for any optimal solution S to I_f.*

We will show by induction on $i \in \{1, ..., |T_f^-|\}$ that for any $T_i \subseteq T_f^-$ with $|T_i| = i$, and any $v \in V \setminus T_i$ it holds that

$$g(T_i, v) = C(S) \tag{64}$$

for any optimal solution S to $I_f(T_i \cup \{v\})$. First, one observes from the definition of d'_{pc} that (64) holds for any $t \in T_f^-$ and any $v \in V \setminus \{t\}$. Next, let $i \in \{2, ..., |T_f^-|\}$. Assume that (64) holds for all non-empty $T \subset T_f^-$ with $|T| < i$. Choose any $T_i \subseteq T_f^-$ with $|T_i| = i$, and choose any $v \in V \setminus T_i$. Let S be an optimal solution to $I_f(T_i \cup \{v\})$. Split S as follows: If $\delta_S(v) = 1$ let $P := (v, \emptyset)$, otherwise let $P \subseteq S$ be the path from v to the first vertex $w \in V(S)$ with $\delta_S(w) > 2$ or $w \in T_i$. Observe that

$$C(P) = d'_{pc}(v, w) + \sum_{u \in V \setminus \{w\}} p(u). \tag{65}$$

Let $\tilde{S} := (V(S \setminus P) \cup \{w\}, E(S \setminus P))$. Because of Lemma 24, $\tilde{S}$ is an optimal solution to $I_f(T_i \cup \{w\})$ and P to $I_f(\{v, w\})$. Further:

$$C(S) = C(\tilde{S}) + C(P) - \sum_{u \in V \setminus \{w\}} \overset{(65)}{=} C(\tilde{S}) + d'_{pc}(v, w). \tag{66}$$

Moreover, $\tilde{S}$ can be split into two trees $\tilde{S}_1$ and $\tilde{S}_2$ such that $\tilde{S}_1 \cap \tilde{S}_2 = \{w\}$, $\tilde{S}_1 \cup \tilde{S}_2 = \tilde{S}$, and $\tilde{S}_1 \cap T_i \neq \emptyset$, $\tilde{S}_2 \cap T_i \neq \emptyset$. With $\tilde{T}_1 := (T_i \cap \tilde{S}_1) \cup \{w\}$ and $\tilde{T}_2 := (T_i \cap \tilde{S}_2) \cup \{w\}$, it holds by

Lemma 24 that $\tilde{S}_1$ is an optimal solution to $T_f(\tilde{T}_1)$ and $\tilde{S}_2$ to $T_f(\tilde{T}_2)$. Lemma 24 furthermore implies that:

$$f(T_i, w) \leq C(\tilde{S}_1) + C(\tilde{S}_2) - \sum_{u \in V \setminus \{w\}} p(u). \tag{67}$$

From the optimality of $\tilde{S}$ combined with Lemma 23 we obtain:

$$f(T_i, w) = C(\tilde{S}_1) + C(\tilde{S}_2) - \sum_{u \in V \setminus \{w\}} p(u). \tag{68}$$

Similarly, from Lemma 23 and Lemma 24 we obtain.

$$g(T_i, w) \overset{(68)}{\leq} C(\tilde{S}_1) + C(\tilde{S}_2) + d'_{pc}(v, w) \tag{69}$$

$$= C(\tilde{S}) - \sum_{u \in V \setminus \{w\}} p(u) + d'_{pc}(v, w) \tag{70}$$

$$\overset{(66)}{=} C(S). \tag{71}$$

Equality follows from Lemma 23 and the optimality of S.

Claim 2: *The above dynamic programming algorithm terminates in time* $O(3^{|T_f|}n + 2^{|T_f|}n^2 + n^2 \log n + mn)$.

For $i \geq 2$ the algorithm differs from Dreyfus-Wagner essentially only in the weight functions of the trees. For $i = 1$ one observes the following. For a given $v \in V$, the distances $d'_{pc}(v, w)$ on I_f to all $w \in V$ can be computed in time $O(n \log n + m)$ by using an adaptation of Dijkstra's algorithm, similar to Algorithm 1, that runs in time $O(n \log n + m)$. Thus, the distances for all pairs (v, w) can be computed in $O(n^2 \log n + mn)$. Consequently, the overall dynamic programming algorithm algorithm has the same run time as Dreyfus-Wagner. $\square$

A.3 Proof of Proposition 10

Proof. First, one can verify from the definition of Algorithm 1 that if it returns *deletable*, then $c(\{v, w\}) \geq d^-_{pc}(v, w)$ holds—also without condition (16). To show the converse, assume in the following that $c(\{v, w\}) \geq d^-_{pc}(v, w)$. To simply the presentation it will also be assumed that

$$p(v) = 0, \tag{72}$$

which does neither change $d^-_{pc}(v, w)$, nor the behavior of Algorithm 1. Further, note that because of (16) one can assume that W is a (simple) path. Otherwise, replace W by a shortest path (with respect to the edge costs c) between v and w in the subgraph corresponding to W. Indeed, because of (16) the prize-constrained length of this shortest path is not higher than that of W. As before, write $W = (v_1, e_1, v_2, e_2, ..., e_r, v_r)$ with $v_1 = v$ and $v_r = w$. Condition (16) furthermore implies that

$$l_{pc}(W(v, v_{k+1})) = l_{pc}(W(v, v_k)) + c(\{v_k, v_{k+1}\}) - p(v_k) \tag{73}$$

for any $k \in \{1, 2, ..., r - 1\}$.

In the following, we will show that

$$dist_{pc}[v_k] \leq l_{pc}(W(v, v_k)) - p(v_k) \tag{74}$$

holds for any $k \in \{1, ..., r-1\}$. Thereby, the proof is concluded: Algorithm 1 can in this case reach w from v_{r-1} due to

$$\begin{aligned} dist_{pc}[v_{r-1}] + c(\{v_{r-1}, v_r\}) &\overset{(74)}{\leq} l_{pc}(W(v, v_{r-1})) - p(v_{r-1}) + c(\{v_{r-1}, v_r\}) && (75) \\ &\overset{(73)}{=} l_{pc}(W(v, v_r)) && (76) \\ &= d^-_{pc}(v, w) && (77) \\ &\leq c(\{v, w\}). && (78) \end{aligned}$$

Thus, the algorithm returns *deletable*.

We will show (74) by induction on $k = 1, ..., r-1$. First, one readily verifies that (74) holds for $k = 1$. Next, let $k \in \{1, ..., r-2\}$ and assume that (74) holds for k. Suppose

$$dist_{pc}[v_{k+1}] > l_{pc}(W(v, v_{k+1})) - p(v_{k+1}). \tag{79}$$

Now perform Algorithm 1 until $dist_{pc}[v_k]$ satisfies (74). At this point, $forbidden[v_{k+1}]$ must have been set to *true*, otherwise one could update $dist_{pc}[v_{k+1}]$ from vertex v_k: Indeed, $dist_{pc}[v_k] + c(\{v_k, v_{k+1}\}) \leq c(\{v, w\})$ holds, which can be shown equivalently to (75)-(78). Thus, also the second condition for updating $dist_{pc}[v_{k+1}]$ from vertex v_k is fulfilled. For the third (and last condition), one obtains:

$$\begin{aligned} dist_{pc}[v_k] + c(\{v_k, v_{k+1}\}) - p(v_{k+1}) &\overset{(74)}{\leq} l_{pc}(W(v, v_k) - p(v_k) + c(\{v_k, v_{k+1}\}) - p(v_{k+1}) && (80) \\ &\overset{(73)}{=} l_{pc}(W(v, v_{k+1}) - p(v_{k+1}) && (81) \\ &\overset{(79)}{<} dist_{pc}[v_{k+1}]. && (82) \end{aligned}$$

If $forbidden[v_{k+1}]$ is set to *true*, the vertex v_{k+1} must already have been removed from Q in Algorithm 1 (which happens exactly one time, because at this point v_{k+1} will be marked as forbidden). We will show (by induction) for $j = 1, ..., k$ that v_j satisfies (74) at the point when v_{k+1} is removed from Q. In this way, we obtain a contradiction, because if v_k satisfies (74), then

$$\begin{aligned} dist_{pc}[v_k] &\overset{(74)}{\leq} l_{pc}(W(v, v_k) - p(v_k) && (83) \\ &\overset{(73)}{=} l_{pc}(W(v, v_{k+1}) - c(\{v_k, v_{k+1}\}) && (84) \\ &\overset{(16)}{\leq} l_{pc}(W(v, v_{k+1}) - p(v_{k+1}) && (85) \\ &\overset{(79)}{<} dist_{pc}[v_{k+1}]. && (86) \end{aligned}$$

This implies that v_k would have been removed before v_{k+1} from Q. Consequently, the algorithm would have updated $dist_{pc}(v_{k+1})$ to $dist_{pc}[v_k] + c(\{v_k, v_{k+1}\}) - p(v_{k+1})$, and (74) would hold for v_{k+1}, as shown in (80),(81).

We conclude with the induction for $j = 1, ..., k$. By definition, v_1 satisfies (74) when v_{k+1} is removed from Q. Assume that the same holds for v_j with $j \in \{2, ..., k-1\}$. Then v_j must have been removed from Q before v_{k+1}, because $dist_{pc}[v_j] < dist_{pc}[v_{k+1}]$ holds, which can be shown similarly to (83)-(86). Thus, one could update $dist_{pc}[v_{j+1}]$ from v_j to

$$dist_{pc}[v_{j+1}] := dist_{pc}[v_j] + c(\{v_j, v_{j+1}\}) - p(v_{j+1}) \overset{(73),(74)}{\leq} l_{pc}(W(v, v_{j+1})) - p(v_{j+1}), \tag{87}$$

which shows that v_{j+1} satisfies (74).

□

A.4 Proof of Proposition 12

Proof. Initially, define $b : V \to \{0, 1, 2, ..., s^+\}$ such that $v \in H_{b(v)}$ for all $v \in V$. Assume that v_i is contained in all optimal solutions. This assumption implies that $|T_p^+| \geq 2$. Let S be any optimal solution. Denote the (unique) path in S between v_i and any $t_j \in V(S) \cap H^p$ by Q_j and the set of all such paths by $\mathcal{Q}$. First, we can assume that $|\mathcal{Q}| \geq 2$, because if $\mathcal{Q}$ was just contained in one path, say Q_k, then we could simply remove v_i from Q_k to obtain another optimal solution. 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. This edge 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

$$\big|\{\{v, w\} \in E(Q_k) \cup E(Q_l) \mid b(v) \neq b(w)\}\big| \tag{88}$$

is minimized. Define $\mathcal{Q}^- := \mathcal{Q} \setminus \{Q_k, Q_l\}$. Consider a (t, v_i)-path $Q_r \in \mathcal{Q}^-$. If $t \in T_p^+$, let Q_r' be the subpath of Q_r between t and the first vertex not in the region of t. 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 any Q_r'. Conclusively, the paths Q_k, Q_l, and all Q_r' are edge-disjoint. Next, we use these paths to derive the lower bound (19) on $C(S)$. To this end we introduce additional notation. First, denote the union of Q_k, Q_l, and all Q_r' by Q. Define $S_Q := S \cap Q$. Because for each non-proper potential terminal in $V(S) \setminus V(S_Q)$ there is one incident edge in $E(S) \setminus E(S_Q)$, and because this mapping can be chosen to be bijective, it holds that

$$c(E(S) \setminus E(S_Q)) - p((V(S) \setminus V(S_Q)) \cap T_p^-) \geq 0. \tag{89}$$

From the definitions of $\underline{d}_{H^p}$ and r_H^{pc} one infers

$$c(E(S_Q)) + p(T_p^+ \setminus V(S)) - p(V(S_Q) \cap T_p^-) \tag{90}$$

$$\geq \sum_{q=1}^{s^+-2} r_H^{pc}(t_q^+) + \underline{d}_{H^p}(v_i, \underline{v}_{i,1}^{H^p}) + \underline{d}_{H^p}(v_i, \underline{v}_{i,2}^{H^p}) - p(v_i). \tag{91}$$

Finally, one obtains:

$$\begin{aligned}
C(S) &= c(E(S)) + p(V \setminus V(S)) \\
&= c(E(S)) + p(T_p^+ \setminus V(S)) + p(T_p^- \setminus V(S)) \\
&= c(E(S)) + p(T_p^+ \setminus V(S)) + p(T_p^-) - p(V(S) \cap T_p^-) \\
&= c(E(S)) + p(T_p^+ \setminus V(S)) + p(T_p^-) \\
&\quad - p(V(S_Q) \cap T_p^-) - p((V(S) \setminus V(S_Q)) \cap T_p^-) \\
&= c(E(S_Q)) + c(E(S) \setminus E(S_Q)) + p(T_p^+ \setminus V(S)) + p(T_p^-) \\
&\quad - p(V(S_Q) \cap T_p^-) - p((V(S) \setminus V(S_Q)) \cap T_p^-) \\
&\overset{(89)}{\geq} c(E(S_Q)) + p(T_p^+ \setminus V(S)) + p(T_p^-) - p(V(S_Q) \cap T_p^-) \\
&\overset{(90)}{\geq} \sum_{q=1}^{s^+-2} r_H^{pc}(t_q^+) + \underline{d}_{H^p}(v_i, \underline{v}_{i,1}^{H^p}) + \underline{d}_{H^p}(v_i, \underline{v}_{i,2}^{H^p}) - p(v_i) + p(T_p^-) \\
&= \sum_{q=1}^{s^+-2} r_H^{pc}(t_q^+) + \underline{d}_{H^p}(v_i, \underline{v}_{i,1}^{H^p}) + \underline{d}_{H^p}(v_i, \underline{v}_{i,2}^{H^p}) + p(T_p^- \setminus \{v_i\}).
\end{aligned}$$

The first equality is just the definition of $C(S)$. The second inequality follows from $T_p = T_p^+ \dot\cup T_p^-$. The next three equalities result from splitting up individual sums. The last equality follows from the fact that either $v_i \in T_p^-$ or $p(v_i) = 0$. $\square$

A.5 Proof of Proposition 14

Proof. Throughout this proof it will be assumed that no trivial (i.e. single-vertex) optimal solution exists—otherwise, the proof is already complete. Note that this assumption implies that $T_p^+ \neq \emptyset$.

First, assume that $T_p^+ \cap B = \emptyset$. Thus, $T_p^+ \subseteq R$. Recall that because no trivial solution exists, there is at least one optimal solution S whose leaves are a subset of T_p^+. Consequently, S cannot contain any edge of B. Otherwise, from the ends of this edge there would be two disjoint paths to T_p^+, and thus there would be at least two gate vertices from B to R.

Second, assume that $T_p^+ \cap B \neq \emptyset$. Let S be a feasible solution with $S \nsubseteq B$ and $E(S) \cap E(B) \neq \emptyset$. We will show that a feasible solution S' with $C(S') \leq C(S)$ exists, such that either $S' \subseteq B$ or $E(S') \cap E(B) = \emptyset$. In this way, the proof is concluded. Assume that all leaves of S are contained in T_p^+, otherwise one can always choose a S of no higher cost that satisfies this property. Because this assumption implies $S \cap R \neq \emptyset$, the gate vertex v_i from B to R is contained in S. Moreover, any path $Q \subseteq S$ starting from v_i satisfies either $E(Q) \cap E(B) = \emptyset$ or $E(Q) \subseteq E(B)$. Otherwise, there would be at least two gate vertices from B to R. Let S_B be the subgraph of S that consists of all paths in S from v_i to vertices in B. The above considerations imply that the subgraph S' obtained by removing S_B from S and adding v_i is connected. Similar to the proof of Proposition 12 one can now show that

$$c(E(S_B)) + p(X \setminus V(S_B)) \geq L. \tag{92}$$

Therefore, it holds that

$$\begin{aligned}
C(S) &= c(E(S)) + p(V \setminus V(S)) \\
&= c(E(S')) + p((V \setminus V(S')) \setminus X) + c(E(S_B)) + p(X \setminus V(S_B)) \\
&\overset{(92)}{\geq} c(E(S')) + p((V \setminus V(S')) \setminus X) + L \\
&\overset{(21)}{\geq} c(E(S')) + p((V \setminus V(S')) \setminus X) + p(X) \\
&= c(E(S')) + p((V \setminus V(S')) \\
&= C(S'),
\end{aligned}$$

which concludes the proof. □

A.6 Proof of Proposition 15

We will show that the $\mathcal{NP}$-hardness holds already for the SPG variant of the terminal-regions decomposition (which implies the $\mathcal{NP}$-hardness for PCSTP). Note that if $T_p^- = \emptyset$, then there always exists a terminal-regions decomposition with $H_0 = \emptyset$ that maximizes the lower bound (19). Therefore, for an SPG (V, E, T, c) define the terminal-regions decomposition as a partition $H = \big\{H_t \subseteq V \mid T \cap H_t = \{t\}\big\}$ of V such that for each $t \in T$ the subgraph induced by H_t is connected. Define for all $t \in T$

$$r_H(t) := \min\{d(t, v) \mid v \notin H_t\}. \tag{93}$$

Note that this definition is just a special case of the PCSTP version (for PCSTP instances with sufficiently high prizes). 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 (19), or (20)—which corresponds to $\alpha = 2$ and $\alpha = 1$, respectively.

Lemma 25. *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 [23]) 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.

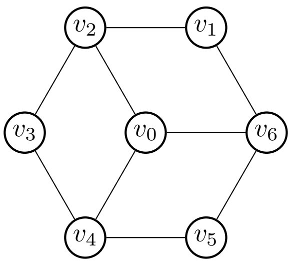
First, one observes that the size $|V_0| + |E_0|$ of the new graph G_0 is a polynomial in the size $|V_{ind}| + |E_{ind}|$ of G_{ind}. Next, $r_{H_0}(v_i) = 2$ holds for a vertex $v_i \in G_0 \cap G_{ind}$ if and only if H_{v_i} contains all (newly inserted) adjacent vertices of v_i in G_0. Moreover,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

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

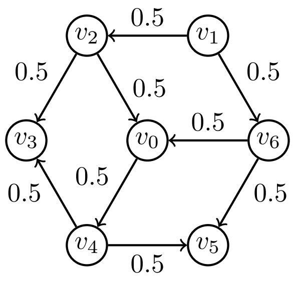
This proves the proposition. □

A.7 Proof of Proposition 22

Proof. We only show the second part of the proposition. First it follows from the construction of Transformation 2 and 3 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 $c(e) = 1$ for all edges e. Further, 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 $v_{LP}(TransCut(I)) = 4.5 + \frac{\epsilon}{2} < 5 = v_{LP}(TransRCut(I_{T_0}))$. 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., $x((r', v_1)) = 1$).



□