A dual-ascent-based branch-and-bound framework for the prize-collecting Steiner tree and related problems

Markus Leitner*[1], Ivana Ljubić †[2], Martin Luipersbeck‡[1], and Markus Sinnl§[1]

[1]Department of Statistics and Operations Research, University of Vienna, Austria
[2]ESSEC Business School of Paris, France

Abstract

In this work we present a branch-and-bound (B&B) framework for the asymmetric prize-collecting Steiner tree problem (APCSTP). Several well-known network design problems can be transformed to the APCSTP, including the Steiner tree problem (STP), prize-collecting Steiner tree problem (PCSTP), maximum-weight connected subgraph problem (MWCS) and the node-weighted Steiner tree problem (NWSTP). The main component of our framework is a new dual ascent algorithm for the rooted APCSTP, which generalizes Wong's dual ascent algorithm for the Steiner arborescence problem. The lower bounds and dual information obtained from the algorithm are exploited within powerful bound-based reduction tests and for guiding primal heuristics. The framework is complemented by additional alternative-based reduction tests. All tests are applied in every node of the B&B tree. Extensive computational results on benchmark instances for the PCSTP, MWCS and NWSTP indicate the framework's effectiveness, as most instances from literature are solved to optimality within seconds, including most of the (previously unsolved) largest instances from the recent DIMACS Challenge on Steiner Trees. In many cases the framework even manages to outperform recently proposed state-of-the-art exact and heuristic algorithms. Since the network design problems addressed in this work are frequently used for modeling various real-world applications (e.g., in bioinformatics), the presented B&B framework will also be made publicly available.

1 Introduction

Variants of the Steiner tree problem appear in a broad range of diverse applications, ranging from infrastructure network design (Ljubić et al. 2006) to the analysis of biological networks (Dittrich et al. 2008) and pattern recognition (Chen and Grauman 2012, Hegde et al. 2014). Many of these

*markus.leitner@univie.ac.at
†ivana.ljubic@essec.edu
‡martin.luipersbeck@univie.ac.at
§markus.sinnl@univie.ac.at

variants can be covered by a common problem definition. For this purpose, let an *arborescence* $S = (V_S, A_S)$ rooted at $r \in V_S$ be defined as a subgraph of a given directed graph $G = (V, A)$, such that for each node $i \in V_S \setminus \{r\}$ there exists exactly one directed path from r to i. Furthermore, if node $i \in V_S$, we say that S *spans* i. Using this definition, the problem addressed in this work can be stated as follows:

Definition 1 (Asymmetric prize-collecting Steiner tree problem (APCSTP))**.** *Given a directed graph $G = (V, A)$, arc costs $c : A \mapsto \mathbb{R}_{\geq 0}$, node prizes $p : V \mapsto \mathbb{R}_{\geq 0}$ and a set of fixed terminals T_f, the goal is to find an arborescence $S = (V_S, A_S) \subseteq G$ that spans T_f such that the cost*

$$c(S) = \sum_{(i,j) \in A_S} c_{ij} + \sum_{i \notin V_S} p_i$$

is minimized.

Let $T_p := \{i \in V \setminus T_f : p_i > 0\}$ be the set of *potential terminals*, $T := T_p \cup T_f$ the set of *terminals* and $V \setminus T$ the set of *Steiner nodes*. Optionally, the given problem definition may be extended by specifying a root node $r \in T_f$, such that each feasible solution is an arborescence rooted at r. This extension of the APCSTP is referred to as the rooted APCSTP. As will be shown, several network design problems can be transformed to the unrooted/rooted APCSTP and the restriction regarding the non-negativity of c and p is made *without loss of generality*.

Our research has been partially motivated by the 11th DIMACS challenge on Steiner tree problems. This event has provided new results on the state-of-the-art of existing computational methods, but has also highlighted algorithmic ideas that have not received sufficient attention in the existing literature. The challenge has shown that it should be possible to design flexible algorithmic frameworks that can exploit similarities between classes of Steiner tree problems. Such frameworks are not only flexible with respect to the range of problem classes they can address — more importantly, they are capable of generalizing successful algorithmic techniques to various problem classes. One example is the framework proposed by Gamrath et al. (2015), in which several Steiner tree related problems are solved by the same integer linear programming (ILP) model via transformations of the input instance. A similar approach is followed by Fischetti et al. (2014), whose implementation achieved the best overall computational results among the proposed exact algorithms. Unfortunately, in spite of the recent contributions of the DIMACS challenge, some computationally successful methodologies have remained restricted to the Steiner tree problem (STP), like the design of branch-and-bound (B&B) frameworks based on dual heuristics. Pajor et al. (2014) have proposed a new, empirically more efficient, implementation of the dual ascent algorithm for the Steiner arborescence problem (SAP) originally proposed by Wong (1984), and have presented computational results for a B&B procedure based on this algorithm for the STP. In the past, Wong's dual ascent algorithm has been successfully applied within a sophisticated B&B framework for the STP by Polzin and Daneshmand (2001), in which a wealth of reduction tests

and various other algorithmic techniques are applied.

Contributions. To the best of our knowledge no framework based on a dual-ascent-based methodology has been proposed for any other Steiner-tree-related network design problem, although the approach seems promising. We therefore present a novel B&B framework for the APCSTP based on a dual-ascent procedure. The latter generalizes the dual ascent algorithm for the SAP by Wong (1984), and shares similarities with the primal-dual 2-approximation algorithm for the PCSTP proposed by Goemans and Williamson (1995). The framework is complemented with new bound-based and alternative-based reduction tests formulated for the APCSTP. Via simple problem transformations this framework is capable of solving instances of the prize-collecting Steiner tree problem (PCSTP), the maximum-weight connected subgraph problem (MWCS), the node-weighted Steiner tree problem (NWSTP), and the STP. Extensive computational results are presented, showing that in many cases the framework manages to outperform more sophisticated state-of-the-art exact and heuristic algorithms presented at the 11th DIMACS challenge. In addition, as the network design problems addressed in this work are used for modeling various real-world applications (e.g., in bioinformatics), we decided to make our framework publicly available.

Outline. Section 2 gives an overview of the B&B framework's general layout. Section 3 describes our new dual ascent algorithm for the rooted APCSTP. Section 4 presents bound-based and alternative-based reduction tests. Section 5 covers implementation details of the framework. Section 6 lists computational results, while concluding remarks are drawn in Section 7.

2 General framework

Figure 1 shows a schematic representation of the B&B framework. In Step 1, the given input instance is transformed to the APCSTP (see Section 5.3). An initial preprocessing and heuristics are applied in Step 2. This phase consists of an exhaustive application of the reduction tests that will be detailed in Section 4.2, as well as the application of primal heuristics described in Section 5.1 for a limited number of iterations.[1]

Steps 3 to 7 represent the main loop of the B&B procedure. In Step 3, a B&B node is selected for processing. A dual ascent algorithm for the APCSTP (see Section 3) is used in Step 4 to compute a lower bound for the given B&B node. A primal heuristic is executed in Step 4 to possibly improve the global upper bound while reduction tests (Section 4) are applied in Step 6. Depending on the computed bounds, the B&B node is either pruned or a node-based branching is performed in Step 7. The procedure terminates if all B&B nodes have been pruned or a given time limit has been reached.

[1] We have here 2 times references to Sections with reduction tests. I would mention it only one. But, once we refer to Section 4, the other time to Section 4.2 - can we make it consistent?

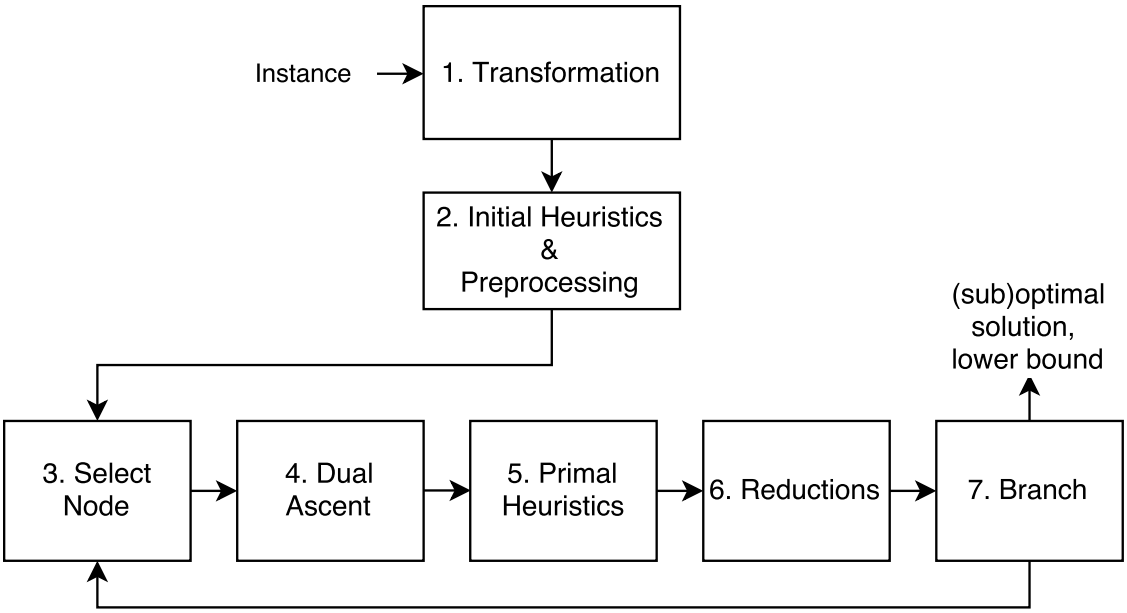


Figure 1: Overview of the B&B procedure.

3 A dual ascent algorithm for the rooted APCSTP

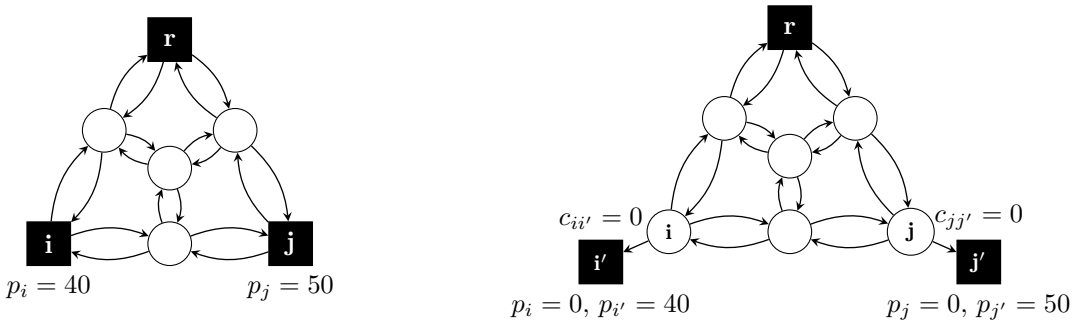
The following notation is used: For any node set $W \subseteq V$, let $\delta^-(W) = \{(i,j) \in A : i \notin W, j \in W\}$ and $\delta^+(W) = \{(i,j) \in A : i \in W, j \notin W\}$ denote its set of incoming resp. outgoing arcs. For brevity, we write $\delta^-(i)$ and $\delta^+(i)$ if W is a singleton.

Without loss of generality, the dual ascent algorithm is stated assuming that the condition $|\delta^-(i)| = 1, |\delta^+(i)| = 0, \forall\, i \in T_p$ is satisfied by the given input instance. Otherwise, any given instance can be transformed into an equivalent instance for which the condition holds in the following way: For each $i \in T_p$ with $|\delta^-(i)| > 1$ or $|\delta^+(i)| > 0$, an additional node i' and arc (i, i') are added to G. The associated prizes and arc cost are set to $p_{i'} := p_i$, $c_{ii'} := 0$ and $p_i := 0$. This transformation corresponds to shifting the prize p_i of node i to node i'. Figure 2 shows an instance before and after the transformation. By definition, the transformation allows a one-to-one mapping between feasible solutions to the original and transformed instance, in which all leaf nodes are terminals. Note that all other solutions may be trivially improved by pruning Steiner nodes.

A cut-based ILP formulation for the rooted APCSTP can be stated in terms of connectivity cuts. Herefore, let $\mathcal{W} := \{W \subset V : r \notin W, V \cap T \neq \emptyset\}$ be all node subsets inducing Steiner cuts. Due to the applied transformation, it is sufficient to consider the following subsets of $\mathcal{W}$:

$$\mathcal{W}_f := \{W \in \mathcal{W} : |W \cap T_f| \geq 1\}$$
$$\mathcal{W}_p := \{W \in \mathcal{W} : |W \cap T_p| = 1\}$$

Clearly, the Steiner cuts induced are sufficient to ensure connectivity, as due to the transformation,



(a) Original instance

(b) Transformed instance

Figure 2: APCSTP instance and its counterpart after transformation (costs of original arcs omitted).

there exists no path $P_G(r, i), i \in T$, which crosses a potential terminal. Moreover, each $W \in \mathcal{W}_p$ can now be uniquely associated to exactly one potential terminal which greatly simplifies the following problem formulation and subsequently the given dual ascent algorithm. Therefore these cuts also correspond to those that can be found by the algorithm.

Model (CUT) shown below states the linear programming (LP) relaxation of the cut-based ILP formulation.

$$\text{(CUT)} \quad \min \quad \sum_{(i,j)\in A} c_{ij}x_{ij} + \sum_{i'\in T_p} (1 - x_{ii'})p_{i'} \tag{1}$$

$$\text{s.t.} \quad x(\delta^-(W)) - 1 \geq 0 \qquad \forall W \in \mathcal{W}_f \qquad (\beta_W) \tag{2}$$

$$x(\delta^-(W)) - x_{ii'} \geq 0 \qquad \forall W \in \mathcal{W}_p, i' \in W \cap T_p \qquad (\beta'_W) \tag{3}$$

$$-x_{ii'} \geq -1 \qquad \forall i' \in T_p \qquad (\pi_{i'}) \tag{4}$$

$$x_{ij} \geq 0 \qquad \forall (i,j) \in A \tag{5}$$

A variable $x_{ij} \in [0, 1]$ is associated to each arc $(i, j) \in A$. In the original formulation, each x_{ij} is a binary variable, such that $x_{ij} = 1$ if $(i, j) \in A_S$, and $x_{ij} = 0$ otherwise. Note that in the relaxation, upper bound constraints (4) are only necessary for arcs (i, i') with $i' \in T_p$, as at optimality the rest is redundant due to minimization and non-negativity of the cost vector c.

The objective function (1) minimizes the sum over the costs of all arcs part of a solution, plus the sum over the prizes of all unconnected nodes T_p. Note that $(i, i') \in S$ implies that the prize $p_{i'}$ is collected. Constraints (2) and (3) ensure that there exists a directed path from the root r to each fixed terminal $i \in T_f$, and to each potential terminal $i' \in T_p$ with $x_{ii'} = 1$.

(CUT-D) model shown below denotes the dual of (CUT), where $\boldsymbol{\beta}$, $\boldsymbol{\beta'}$ and $\boldsymbol{\pi}$ denote the dual

Data: Rooted APCSTP instance $(G = (V, A), c, p, T, r)$
Result: Lower bound LB, reduced costs $\tilde{\mathbf{c}}$, dual vector $\boldsymbol{\pi}$

$LB \leftarrow 0$
$\tilde{c}_{ij} \leftarrow c_{ij} \qquad \forall\, (i,j) \in A$
$\pi_{i'} \leftarrow p_{i'} \qquad \forall\, i' \in T_p$
$T_a \leftarrow T_f \cup T_p \setminus \{r\}$
/* Implicitly set $\beta_W = 0, \forall\, W \in \mathcal{W}_f$, $\beta'_W = 0, \forall\, W \in \mathcal{W}_p$. */
while $T_a \neq \emptyset$ **do**
  $k \leftarrow$ `chooseActiveTerminal`(T_a)
  $W \leftarrow W(k)$
  $\Delta \leftarrow \texttt{min}_{(i,j)\in\delta^-(W)} \tilde{c}_{ij}$
  if $k \in T_p$ **then**
    $\Delta \leftarrow \texttt{min}\{\Delta, \pi_k\}$
    $\pi_k \leftarrow \pi_k - \Delta$
  $\tilde{c}_{ij} \leftarrow \tilde{c}_{ij} - \Delta \qquad \forall\, (i,j) \in \delta^-(W)$
  $LB \leftarrow LB + \Delta$
  $T_a \leftarrow$ `removeInactiveTerminals`(T_a)
  /* Implicitly set $\beta_W = \Delta$ or $\beta'_W = \Delta$, depending on $k \in T_f$ or $k \in T_p$. */

Algorithm 1: Dual ascent algorithm for the rooted APCSTP.

vectors associated to constraints (2), (3) and (4), respectively. The structure of (CUT-D) shares similarities with the dual of the directed cut formulation for the SAP (cf., Polzin and Daneshmand (2001)). For each arc $(i,j) \in A$, packing constraints (7) restrict the sum of dual variables $\boldsymbol{\beta}$ and $\boldsymbol{\beta'}$ whose associated cut contains (i,j). Constraints (8) link dual variables β'_W and $\pi_{i'}$, $i' \in W, W \in \mathcal{W}_p$. Thus each β'_W only appears in exactly one constraint from (8).

$$\text{(CUT-D)} \quad \max \quad \sum_{i' \in T_p} (p_{i'} - \pi_{i'}) + \sum_{W \in \mathcal{W}_f} \beta_W \tag{6}$$

$$\text{s.t.} \quad \sum_{W \in \mathcal{W}_p : (i,j) \in \delta^-(W)} \beta'_W + \sum_{W \in \mathcal{W}_f : (i,j) \in \delta^-(W)} \beta_W \leq c_{ij} \qquad \forall\, (i,j) \in A \tag{7}$$

$$-\pi_{i'} - \sum_{W \in \mathcal{W}_p : i' \in W} \beta'_W \leq -p_{i'} \qquad \forall\, i' \in T_p \tag{8}$$

$$(\boldsymbol{\beta}, \boldsymbol{\beta'}, \boldsymbol{\pi}) \in \mathbb{R}_{\geq 0}^{|\mathcal{W}_f| \times |\mathcal{W}_p| \times |T_p|} \tag{9}$$

The simple structure of formulation (CUT-D) allows the design of a dual ascent procedure. Starting with a dual feasible solution, dual variables are iteratively adapted by a greedy scheme such that the objective value of (CUT-D) increases monotonically, while dual feasibility is preserved in each step. The following notation is used: The *reduced cost* of an arc $(i,j) \in A$ is defined as $\tilde{c}_{ij} := c_{ij} - \sum_{W \in \mathcal{W}_f : (i,j) \in \delta^-(W)} \beta_W - \sum_{W \in \mathcal{W}_p : (i,j) \in \delta^-(W)} \beta'_W$. The *saturation graph* $G_S \subseteq G$ is the subgraph induced by the set of *saturated arcs* $\{(i,j) \in A : \tilde{c}_{ij} = 0\}$. Let $P_H(i,j)$ denote a directed

path from $i \in V$ to $j \in V$ on some graph $H = (V_H, A_H)$. The set of *active terminals* is defined as $T_a := \{k \in T_p : \nexists P_{G_S}(r,k), \text{ and } \pi_k > 0\} \cup \{k \in T_f : \nexists P_{G_S}(r,k)\}$, and let an *active component* rooted at an active terminal k be defined as $W(k) := \{i \in V : \exists P_{G_S}(i,k)\}$, i.e., this is a set of nodes that can be reached from i in the saturated graph. It follows that each $W(k)$ induces a valid Steiner cut $\delta^-(W(k))$.

Algorithm 1 shows the dual ascent procedure. In Steps 1–4, variables are initialized. The algorithm implicitely tracks the values of variables $\boldsymbol{\beta}$ and $\boldsymbol{\beta'}$ in the form of reduced costs, which are initially set to the respective arc costs. The lower bound (LB) is set to zero, and π_i is set to p_i. In the context of the dual ascent algorithm, $\boldsymbol{\pi}$ can be seen as node potential, sharing similarities with the 2-approximation algorithm for the PCSTP by Goemans and Williamson (1995). Note that the chosen initial values imply that all variables $\boldsymbol{\beta}$ and $\boldsymbol{\beta'}$ are set to zero, and that the initial dual solution is feasible. Initially, the set of active terminals T_a corresponds to $T_f \cup T_p \setminus \{r\}$. Steps 4–14 comprise the main loop. In each iteration, an active terminal $k \in T_a$ is chosen by some priority scheme (see Section 5.4 for details). The associated active component $W(k)$ induces a Steiner cut based on its node set W. Based on W, Δ is computed, representing the maximum possible increase for the associated dual variables β_W or β'_W without violating (7) and (8). If $k \in T_f$, Δ is chosen as the minimum reduced cost over all arcs in $\delta^-(W)$, and the value of the dual variable β_W is increased. If $k \in T_p$, Δ is chosen as before, but restricted by π_k (Step 10), due to the non-negativity of $\boldsymbol{\pi}$. The value of dual variable β'_W is increased, while π_k is decreased by the same amount. In either case, LB is increased by Δ. In each iteration, a subset of active terminals will become inactive, so $|T_a|$ decreases monotonically. The algorithm terminates if no active terminal remains.

Theorem 1. *In each iteration of Algorithm 1, $(\boldsymbol{\beta}, \boldsymbol{\beta'}, \boldsymbol{\pi})$ is a feasible solution to (CUT-D).*

Proof. Proof. By definition, the initial solution is feasible. First, consider an iteration where $k \in T_f$. In this case, exactly one β_W is set to Δ. Since Δ is computed as the minimum reduced cost over all cut arcs, clearly (7) holds. The values of all variables included in (8) remain unchanged for this case. Next, consider an iteration where $k \in T_p$. In this case, exactly one β'_W is set to Δ. The same argument as for $k \in T_f$ holds with respect to (7). For (8), equality is preserved at each step, since both the left and right-hand-side are increased by the same amount. Finally, at each step $\pi_k \geq 0, \forall k \in T_p$, since in any iteration where $k \in T_p$, due to Step 10, Δ is bounded by the current value of π_k. □

Theorem 2. *An execution of Algorithm 1 requires at most $O(|A| \cdot \min\{|T||V|, |A|\})$ steps.*

Proof. Proof. Each iteration takes at most $|A|$ steps. An active terminal k can be chosen in $|A|$ steps, since the set of terminals not reachable from r on G_S can be identified by a breadth-first-search (BFS).The subset $\{i' \in T_p : \pi_{i'} > 0\}$ can also be identified in linear time.[2] Similarly, the

[2]Our efficient implementation avoids using flags to store active terminals and relies on the BFS, see Section 5.4 for further details.

Steiner cut $W(k)$ associated to k can be computed by a BFS on G_S. In each iteration, the number of active terminals or the number of unsaturated arcs decreases. If $\pi_k < \min_{(i,j)\in W(k)} c_{ij}$ for k, then exactly one active terminal becomes inactive, but no arc becomes saturated. The number of iterations where no arc becomes saturated is thus bounded by $|T|$. Otherwise, at least one arc becomes saturated, and at least one node is added to the active component rooted at k. Thus the total number of iterations is bounded by $\min\{|T||V|, |A|\}$. □

For the case $T_p = \emptyset$, Algorithm 1 corresponds to the dual ascent algorithm by Wong (1984). The two algorithms also share the same worst-case complexity.

4 Reduction tests

The given classification into *bound-based* and *alternative-based reduction tests* follows the one proposed by Polzin and Daneshmand (2001) for the STP. Bound-based reductions fix nodes and arcs based on available lower and upper bounds. Alternative-based reductions argue based on the existence of alternative solutions. For example, for each feasible solution that contains some arc, there exists an equivalent or better solution without this arc.

4.1 Bound-based reduction tests

Given an instance $I = (G = (V, A), c, p, T_f, r)$ of the rooted APCSTP, reduction tests can be performed based on the information computed by Algorithm 1, i.e., the reduced costs $\tilde{\boldsymbol{c}}$, dual solution values of $\boldsymbol{\pi}$ and lower bound LB. In addition, an upper bound UB on the optimal objective value of I is required, which is easily obtained from any primal heuristic. Among the proposed reduction tests, Test 1 and 2 share similarities with tests described by (Duin 1993, Polzin and Daneshmand 2001) for the STP.

For any pair of nodes $i, j \in V$, let $d(i, j)$ denote the shortest path distance on G, with $d(i, i) = 0$. Similarly, let $\tilde{d}(i, j)$ denote the shortest path distance using $\tilde{\boldsymbol{c}}$ as costs. For each $i \in T_p$, let $\tilde{p}_i := \sum_{W:i\in W} \beta'_W - p_i + \pi_i$ denote the reduced costs for constraints (8).

Test 1 (Bound-based arc elimination (BAE)). *An arc $(i, j) \in A$ can be removed if $LB + \tilde{d}(r, i) + \tilde{c}_{ij} + \min_{t\in T\setminus\{r\}} \tilde{d}(j, t) > UB$.*

Proof. Proof. Given instance I, construct an instance $I' = (G = (V, A), c', p', T_f, r)$ with modified costs and prizes, where $c' = c - \tilde{c}$ and $p' = p - \tilde{p}$. For any feasible solution S, let $c'(S)$ denote its cost based on $\boldsymbol{p'}$ and $\boldsymbol{c'}$, and $\tilde{c}(S)$ denote its cost based on $\tilde{\boldsymbol{p}}$ and $\tilde{\boldsymbol{c}}$. By definition, the dual solution computed for I by Algorithm 1 is feasible for I', and $LB = LB'$. Similarly, for any feasible solution $S = (V_S, A_S)$ to I, it holds that $c'(S) + \tilde{c}(S) = c(S)$. Since, $LB' \leq c'(S)$, the inequality $LB' + \tilde{c}(S) \leq c(S)$ holds.

Assume that S is optimal and $(i,j) \in A_S$. Then $P_S(r,i)$ exists due to feasibility of S. Without loss of generality, assume that all leafs of S are terminals and $|V_S| > 1$ (there must exist at least one optimal solution in which all leafs are terminal nodes, and single-node solutions can be identified during a preprocessing step). Then $P_S(j,t)$ exists for some $t \in T \setminus \{r\}$. Thus $\tilde{c}(S)$ can be underestimated by the costs of a shortest path from r to the nearest terminal $t \in T$, such that (i,j) lies on that path, using $\tilde{\boldsymbol{c}}$ as arc costs. Thus $LB + \tilde{d}(r,i) + \tilde{c}_{ij} + \min_{t \in T \setminus \{r\}} \tilde{d}(j,t) \leq c(S)$ holds. The inequality must also hold if $c(S)$ is replaced by any valid upper bound UB, and hence we may conclude that an arc (i,j) is redundant if the underestimation of the objective value of a solution that contains (i,j) exceeds the UB. □

Test 2 (Bound-based node elimination (BNE))**.**
A node $i \in V \setminus T_f$ can be removed if $LB + \tilde{d}(r,i) + \min_{t \in T \setminus \{r\}} \tilde{d}(i,t) > UB$.

The validity of Test 2 follows from similar arguments as Test 1 by assuming that node i (rather than arc (i,j)) is part of an optimal solution.

Test 3 (Bound-based node inclusion (BNI))**.**
A node $i' \in T_p$ can be added to T_f if $LB + \pi_{i'} > UB$.

Concerning the validity of Test 3, note that from LP-duality it follows that by increasing the right-hand side of (4) by one unit, the objective value increases by $\pi_{i'}$. This change implies that $x_{ii'}$ is fixed to zero, and thus node i' is not spanned. Consequently, $LB + \pi_{i'}$ is a lower bound under the assumption that i' is not spanned. If this lower bound exceeds UB, any optimal solution must span i'.

For all nodes $i \in V$, the distances $\tilde{d}(r,i)$ and $\min_{t \in T} \tilde{d}(i,t)$ can be computed by two executions of Dijkstra's algorithm. Note that in any dual solution constructed by Algorithm 1, equality holds for constraints (8), and thus $\tilde{p}$ is always zero.

4.2 Alternative-based reduction tests

The presented reduction tests generalize and extend various tests initially introduced for the STP and PCSTP (Duin and Volgenant 1989, Uchoa 2006, Ljubić et al. 2006). For ease of presentation, we define the following operations: A node $i \in V$ can be *removed* from G, if there exists an optimal solution that does not contain i. The elimination of i requires that p_i is added to the cost of any solution. Two nodes $i, j \in V$ can be *merged* into a single node, if any optimal solution contains either both i and j or neither of them. The merged node has prize $p_i + p_j - c_{ij}$, only the cheapest arc for each possible arising pair of parallel arcs (multiarcs) is kept, and c_{ij} must be added to the cost of any solution. Let c_{fixed} denote the sum of all fixed cost accumulated this way. The following two tests are valid for both the unrooted and rooted APCSTP.

Test 4 (Least cost (LC))**.**
Let (i,j) be an arc such that $d(i,j) < c_{ij}$, then (i,j) can be removed.

Test 4 has originally been proposed in the context of the STP and is clearly valid for the APCSTP. Similarly, a variant of the minimum adjacency test for the PCSTP can be easily stated for the APCSTP.

Test 5 (Minimum adjacency (MA))**.** *Let $i,j \in V$ be two adjacent nodes with $c_{ij} = c_{ji}$. If $c_{ij} < \min\{p_i, p_j\}$, $c_{ij} = \min_{(k,j)\in\delta^-(j)} c_{kj}$, and $c_{ji} = \min_{(k,i)\in\delta^-(i)} c_{ki}$, then i and j can be merged.*

The validity of Test 5 is based on the following argument: Due to $c_{ij} = c_{ji} \leq \min\{p_i, p_j\}$, if either i or j is spanned by any feasible solution S, then the other node can be connected without increasing $c(S)$. Furthermore, since the cost of (i,j) and (j,i) is minimal among all incoming arcs to i and j, for any solution S spanning i and j, there exists a solution of equivalent cost that contains either (i,j) or (j,i). Note that the condition $c_{ij} = c_{ji}$ is somewhat restricting if arc costs are asymmetric. A test without this condition can be formulated for the rooted APCSTP.

Test 6 (Asymmetric minimum adjacency (AMA))**.** *Let $i,j \in V, j \neq r$ be two adjacent nodes with $c_{ij} < c_{ji}$. If $c_{ij} < p_j$, $c_{ji} < p_i$, $c_{ij} = \min_{(k,j)\in\delta^-(j)} c_{kj}$, and $c_{ji} = \min_{(k,i)\in\delta^-(i)} c_{ki}$, then the following operation is valid: Let $\Delta := c_{ji} - c_{ij}$, and set $c_{kj} := c_{kj} + \Delta$, for all $(k,j) \in \delta^-(j)$, $p_j := p_j + \Delta$ and $c_{\text{fixed}} := c_{\text{fixed}} - \Delta$. Then the nodes i and j can be merged.*

Test 6 adjusts arc costs and node prizes favorably in order to balance the costs of two anti-parallel arcs. The performed operation leaves the cost of any feasible solution unchanged. Two cases can be distinguished: If the node j is not spanned by a solution, the increase in p_j is cancelled out by the decrease in c_{fixed}. Conversely, if the node j is spanned, then exactly one incoming arc of (i,j) is chosen and the increase in the c_{ij} and the decrease in c_{fixed} cancel. The rest of the argument follows the same reasoning as for Test 5. Note that the test can be strengthened if $i = r$, as incoming arcs for the root node are redundant. In addition, for both Test 5 and 6, if $i \in T_f$ or $j \in T_f$, the respective prize can be treated as infinite. The following two tests are strengthened variants of Test 6, that are based on strong graph connectivity:

Test 7 (Minimum successor (MS))**.**
Let (i,j) be an arc such that $p_j > c_{ij} = \min_{(k,j)\in\delta^-(j)} c_{kj}$ and i is a cut node with

$$V = \{i\} \cup W \cup \overline{W}, r \in W, j \in \overline{W},$$

then i and j can be merged.

Test 8 (Single successor (SS))**.**
Let (i,j) be a cut arc such that $p_j > c_{ij}, V = W \cup \overline{W}, \delta^-(\overline{W}) = \{(i,j)\}, r \in W, j \in \overline{W}$, then i and j can be merged.

For the validity of Test 7, note that from the definition it follows that if node j is part of a solution, so is i. Since j has no cheaper incoming arc than (i, j), there exists an optimal solution that contains (i, j) if $i \in S$. The validity of Test 8 follows from the same arguments. Finally, three simple additional tests are given without proof, as their validity is obvious from their definition. For these tests counterparts are known for both the PCSTP and STP.

Test 9 (Non-reachability (NR))**.**
Let $i \in V \setminus T_f$ be a node such that there exists no path from root r to i, then i can be removed.

Test 10 (Degree one (D1))**.**
Let $i \in V \setminus T_f$ be a node with exactly one adjacent node j and $(j, i) \in A$. If $c_{ji} \geq p_i$, then i can be removed. If $c_{ji} < p_i$, then i and j can be merged.

Test 11 (Degree two (D2))**.**
Let $i \in V \setminus T_f$ be a node with exactly two adjacent nodes j and k such that $(j, i) \in A, (i, k) \in A$, and $p_i \leq \min\{c_{ji}, c_{ki}\}$. Then (j, i) and (i, k) can be replaced by an arc (j, k) with $c_{jk} = c_{ji} + c_{ik} - p_i$ and $c_{\text{fixed}} := c_{\text{fixed}} + p_i$.

Although most of the given alternative-based reduction tests follow rather simple rules, their effectiveness is usually considerable when applied in combination with bound-based reductions at each node of a B&B tree. For example, the availability of tighter bounds may cause G to become sparser, such that disconnected subgraphs, cut arcs or cut nodes appear. It is also noteworthy that merging nodes is usually beneficial to Test 3, as nodes with higher prize are more likely to be fixed. Note that most of the presented tests can be trivially extended to equality, such that equivalent solutions can be eliminated. Finally, note that Uchoa (2006) already observed that the minimum adjacency test for the PCSTP can be strengthened under various conditions, but did not explore the idea computationally.

5 Algorithmic framework

Combining the presented techniques into an empirically successful framework requires some non-trivial techniques which are detailed within the following sections. Important aspects include the design of primal heuristics, implementation details of the proposed dual ascent algorithm, problem transformation, and details of the B&B procedure.

5.1 Primal heuristic

Previous empirical results on the STP (see, e.g., Polzin and Daneshmand 2001) suggest that the saturation graph G_S obtained by the dual ascent usually contains good primal solutions. This property is due to the complementary slackness condition of linear programming, which states that in an optimal LP-solution, a decision variable or its associated reduced costs is set to zero. The

quality of solutions computed by constructive heuristics can thus be boosted by applying them on G_S instead of G. Among the most successful constructive heuristics for the STP is the shortest path heuristic (SPH) (see Takahashi and Matsuyama 1980, Poggi de Aragão and Werneck 2002). Its application to the SAP is straightforward, and it can also be applied to the APCSTP when $T_p \neq \emptyset$ by simply connecting all nodes in T. By using this approach the computed solution will most likely contain nodes from T_p that can be trivially pruned. Therefore, in our implementation, SPH is followed by an application of the *strong pruning* procedure proposed by Johnson et al. (2000) for the PCSTP, which solves the PCSTP on a tree in linear time, and whose generalization to the APCSTP is straightforward. The whole procedure is referred to as `SPHprune`. We have also attempted to devise new variants of the SPH where node costs are considered directly, but did not manage to achieve improvements with respect to the aforementioned approach.

However, we managed to achieve substantial improvements by using the following simple idea: Based on the fact that G_S does not necessarily contain an optimal solution, it can be beneficial to perturb arc costs in order to explore different saturation graphs G_S. In our approach we use the best incumbent solution $S = (S_V, S_A)$ as seed for perturbation. For each arc $(i, j) \in A$, an adjusted cost c'_{ij} is computed. If $(i, j) \in S_A$, $c'_{ij} := c_{ij} \cdot (1 - \varepsilon)$, otherwise $c'_{ij} := c_{ij} \cdot (1 + \varepsilon)$.

The complete procedure is shown in Algorithm 2. First, the adjusted cost vector c' is computed. Next the dual ascent algorithm (DA) is applied to the resulting instance. Finally, SPH is executed on the resulting G_S, followed by the strong pruning procedure.

Data: Instance $I = (G = (V, A), c, p, r, T_f)$, perturbation factor ε,
seed solution $S = (V_S, A_S)$
Result: Heuristic solution S'

1 $c'_{ij} \leftarrow c_{ij} \cdot (1 - \varepsilon) \quad \forall (i, j) \in A_S$
2 $c'_{ij} \leftarrow c_{ij} \cdot (1 + \varepsilon) \quad \forall (i, j) \notin A_S$
3 $(LB, \tilde{\boldsymbol{c}}, \boldsymbol{\pi}) \leftarrow \texttt{DA}(G, c', p, T, r)$
4 $G_S \leftarrow G[\tilde{c}_{ij} = 0, \quad \forall (i, j) \in A]$
5 $S' \leftarrow \texttt{SPHprune}(G_S, c, p, T_f, r)$
6 **return** S'

Algorithm 2: Primal heuristic.

Another useful strategy for finding near-optimal primal solutions is to apply an exact algorithm to various subgraphs of G, e.g., the graph resulting from the union of multiple solutions or the saturation graph G_S. Both strategies are incorporated into the framework, and are executed during the initialization phase before processing the root node of the B&B tree. As an exact algorithm we apply the same B&B procedure that is used later-on for exact solution on the whole graph G.

5.2 Cost shifting

A strategy similar to the one applied in Test 6 can be used to adjust the cost structure of the given problem instance favorably. The procedure forms an essential preprocessing step to various algorithmic components, since it effectively decreases the size of T_p by (partially) "shifting" the prize of a node into its incoming arcs. A similar idea has been used by Fischetti et al. (2014) to strengthen and reduce the size of an ILP formuluation for the PCSTP. The idea of applying cost shifting to enable further reductions has also been explored by Duin (1993) for the STP.

Given an instance of the rooted APCSTP, the following transformation is valid and may decrease the size of T_p: For each node $i \in T_p \setminus \{r\}$, compute $\Delta := \min\{p_i, \min_{(j,i)\in\delta^-(i)} c_{ji}\}$, and set $c_{ji} := c_{ji} - \Delta \; \forall (j,i) \in \delta^-(i)$, $p_i := p_i - \Delta$, $c_{\mathit{fixed}} := c_{\mathit{fixed}} + \Delta$. This operation is equivalent to performing the first few iterations of Algorithm 1 in which each active terminal is selected for the first time. Let $T'_p := \{i \in T_p : p_i \leq c_{ji} \in \delta^-(i)\}$ be the subset of nodes with positive prize whose prize becomes zero after the transformation, and which are thus removed from T_p. Note that there always exists an optimal solution S in which no node from T'_p is a leaf of S, as such nodes can be removed from S without increasing $c(S)$. Thus even before the transformation T'_p are essentially Steiner nodes.

The transformation can be generalized to the unrooted APCSTP by only considering nodes $T_p \setminus R$, where $R \subseteq V$ is the set of potential root nodes, i.e., all $i \in V$ such that there exists an optimal solution rooted at $i \in R$. For example, for instances with symmetric arc costs, $R = \{i \in T_p : p_i > c_{ji} \in \delta^-(i)\}$. For this, note that the cost of all solutions spanning the same set of nodes is equal in such symmetric instances. Without loss of generality, assume that there exists an optimal solution consisting of more than one node, as the best single-node solution can be found during a simple preprocessing step in linear time. It follows that there exists an optimal solution $S = (V_S, A_S)$ rooted at some $i \in R$. Otherwise, there would exist a solution $S' = (V_S, A'_S)$ rooted at $j \in V_S, j \neq i$, such that $c(S) = c(S')$ for which i can be removed without increasing $c(S')$.

The presented cost shifting procedure significantly increases the solution quality for the primal heuristic given in Section 5.1, as node prizes can now be partially considered within path lengths. Similarly, the amount of successful arc eliminations performed by Test 4 (LC) may be increased. For the dual ascent algorithm, the shifting can be interpreted as a warm start, as the set of iterations in which each active terminal is selected for the first time is independent from each other.

5.3 Problem transformations

The APCSTP covers several known network design problems. For example, for problems defined on an undirected graph like the STP and PCSTP, it is known that they can be stated equivalently on a directed graph with symmetric costs (see, e.g., Koch and Martin (1998), Ljubić et al. (2006)). Herefore a given undirected graph $G = (V, E)$ with edge costs c_{ij}, $\forall \{i, j\} \in E$, is replaced by a directed graph $G_A = (V, A)$ constructed by replacing each edge $\{i, j\} \in E$ by two anti-parallel arcs

$(i,j) \in A$ and $(j,i) \in A$, such that $c_{ij} = c_{ji}$. For the STP, the resulting problem is also known as the SAP. For the PCSTP, we will refer to the resulting problem simply as directed PCSTP (DPCSTP). If $T_p = \emptyset$ and $T_f \neq \emptyset$, the rooted APCSTP corresponds to the SAP. Conversely, if $T_p \neq \emptyset$ and $T_f = \emptyset$, the APCSTP corresponds to the DPCSTP. The MWCS can be transformed to the PCSTP, see Dittrich et al. (2008), and thus also to the APCSTP. Finally, also the NWSTP can be transformed into an instance of the SAP, and consequently to the APCSTP. The NWSTP is a generalization of the STP, where nodes are associated a non-negative cost, which in contrast to the PCSTP, is paid if the corresponding node is part of the solution. The NWSTP can be transformed to the SAP by replacing the input graph by its bidirected equivalent and adding the cost of a node to its incoming arcs. In the following, when referring to instances of the PCSTP, MWCS and NWSTP, we will always refer to their respective representation as rooted/unrooted APCSTP.

Finally, note that Definition 1 requires that arc costs and node prizes of an APCSTP instances are non-negative. At least for the rooted APCSTP, negative costs/prizes can still be addressed by a simple problem transformation which exploits the cost shifting strategy from Section 5.2: For each node $i \in V \setminus \{r\}$ with $\Delta < 0$, where $\Delta := \min\{\min_{(j,i)\in\delta^-(i)} c_{ji}, p_i\}$, set $c_{ji} := c_{ji} - \Delta$ for all $(j,i) \in \delta^-(i)$, $p_i := p_i - \Delta$, $c_{\mathit{fixed}} := c_{\mathit{fixed}} + \Delta$. Clearly, after applying this procedure, the cost/prize of every arc and node is non-negative. As described in Section 5.2, these operations do not change the objective value of any feasible solution, and thus the transformed instance remains equivalent to the original instance.

5.4 Dual ascent implementation

Our implementation of the dual ascent algorithm for the APCSTP follows the one proposed by Pajor et al. (2014) for the SAP. A notable aspect of this implementation is that it avoids the computationally expensive task of storing and updating the set of active components explicitely. Instead, active components are tracked implicitly, i.e., in each iteration where $k \in T_a$ is selected, $W(k)$ is computed from G_S via breadth-first search. In preliminary experiments we have found this strategy to be essential for tackling large-scale instances containing many terminal nodes. For this class of instances, explicit tracking may become computationally burdensome for two reasons: (i) In each iteration, the saturation of an arc usually affects a large number of active components, which all need to be updated. (ii) The high memory requirements in the order of $\Theta(|T||V|)$ may be prohibitive. Both (i) and (ii) are avoided by implicit tracking, as in each iteration exactly one component is updated, and no additional information besides G_S needs to be stored. However, this strategy necessitates a more involved scheme for the selection of active terminals. For this, note that the lower bound computed by the dual ascent algorithm is highly dependent on the order in which active terminals $k \in T_a$ are processed, and successful schemes typically aim at minimizing the size of the induced cut $\delta^-(W(k))$. If the sets of active components are not known, a heuristic selection scheme has to be applied. However, results reported by Pajor et al. (2014) suggest that

this does not significantly affect the quality of lower bounds.

In our implementation, the same heuristic scheme is applied, although augmented with another idea originally proposed by Polzin and Daneshmand (2001). The score of an active terminal k is defined by the following non-decreasing score function $\sigma(k) := \sum_{i \in W} \delta^-(i) - (|W| - 1)$, i.e., the sum of all incoming arcs for each node in the component minus the number of arcs contained within an inverse arborescence spanning all nodes in W, which must exist since W is connected. This value is a non-decreasing estimate for $\delta^-(W(k))$. In each iteration, the active terminal k with the smallest score is selected using a priority queue.

We augment this approach with the following idea proposed by Polzin and Daneshmand (2001), who made the following observation: Given an optimal solution S, LB can only reach $c(S)$ if in each iteration the chosen cut contains exactly one incoming arc from S. Even if S is not optimal, selecting active terminals based on this criteria may in some cases improve the quality of lower bounds, and in turn also the quality of a heuristic solution computed on the support graph. Given a feasible solution $S = (V_S, A_S)$, we incorporate this criteria by using an extended score function $\sigma'(k) := \sigma(k) + \min\{0, |A_S \cap \delta^-(W(k))| - 1\}|V|$. Thus the selection of k follows a lexicographical ordering, where the first part represents the original score function, while the second part counts the number of surplus solution arcs within the cut.

Pajor et al. (2014) observed that as soon as only one terminal remains active, the remaining iterations of the dual ascent algorithm can be equivalently solved as shortest path problem on a graph with adjusted costs. This procedure can be stated as follows: Let k be the last remaining active terminal, LB' the lower bound and $\tilde{\boldsymbol{c}}'$ the reduced costs up to this point. Apply Dijkstra's algorithm using $\tilde{\boldsymbol{c}}'$ as cost for computing the reverse path from k to r. Set an upper bound u on the maximum length of the computed shortest path, i.e., $d(i,k) := \min\{d(i,k), u\}$. If $k \in T_f$, $u := d(r,k)$, otherwise, if $k \in T_p$, $u := \min\{d(r,k), \pi_k\}$. Clearly, the final lower bound is then $LB = LB' + u$. The final reduced costs $\tilde{\boldsymbol{c}}$ can be updated in a linear pass by the following formula:

$$\tilde{c}_{ij} = \tilde{c}'_{ij} - (d(j,k) - d(i,k)) \qquad \forall (i,j) \in A$$

For further implementation details, the reader is referred to Pajor et al. (2014).

5.5 Branch & bound

The following paragraphs list implementation details for the full B&B procedure, which also includes the initial preprocessing and heuristic phase.

Initial preprocessing & heuristic. We begin by applying the alternative-based tests described in Section 4.2 exhaustively, since they are computationally cheap to perform. Cost shifting (see Section 5.2) is also applied. Afterwards, ten iterations of the heuristic procedure detailed in Section 5.1 are performed, using the ten highest-prize nodes as roots and the best incumbent as

seed solution. The initial solution is computed on the unperturbed saturation graph. Afterwards, a restricted B&B search is performed on different subgraphs of G, using a small time limit. The subgraphs include the saturation graphs computed in each iteration, as well as the graph generated by forming the union of all constructed solutions.

Root node processing. Since Algorithm 1 can only be used to compute valid lower bounds for unrooted APCSTP instances if $T_f \neq \emptyset$, the following procedure is applied to deal with the case $T_f = \emptyset$. For a given unrooted APCSTP instance I with $T_f = \emptyset$, let LB_r denote the lower bound computed by Algorithm 1 for each potential root $r \in R \subseteq V$. Then $\min_{r \in R} LB_r$ is a valid lower bound for I. The main disadvantage of this approach is that the set R may be very large. However, for instances with symmetric arc costs the computational effort can be decreased since the cost of all optimal solutions spanning the same set of nodes is equal. Therefore, after each iteration with $r \in R$ chosen as root, r can be removed temporarily during all remaining iterations (c_{fixed} is increased appropriately). Furthermore, if a valid upper bound UB is available, the procedure can be terminated as soon as $c_{fixed} \geq UB$. Consequently, the sequence in which roots are selected may affect the total number of processed roots. In our implementation, we select roots based on their prize in descending order, as the increase in c_{fixed} is maximized. The presented scheme is akin to a restricted B&B search, in which at most one node in R is fixed to one. Similar approaches for exploiting symmetry in PCSTP instances have also been used in ILP formulations (see, e.g., Ljubić et al. (2006)). Finally, since the evaluation of each root can be performed independent from each other, the procedure could be parallelized. However, since preliminary computational experiments have shown that the procedure's running time is usually already quite short on the tested benchmark instances, we refrained from parallelization in our implementation.

Computing lower bounds. Depending on the branching strategy, B&B nodes might get evaluated such that $LB > UB$. Since LB is valid at each iteration of Algorithm 1 and increases monotonically, it follows that the algorithm can be terminated as soon as $LB \geq UB$, since the B&B node will be pruned anyway. Notice, that the final iterations of Algorithm 1 are often much more computationally demanding as the size of active components increases. Thus, the computation time saved can be significant. In general, preliminary experiments indicated that LB is usually already very close to the optimum. However, in some cases, having slightly stronger bounds available may drastically decrease the number of explored B&B nodes. Due to the heuristic nature of Algorithm 1, perturbing the order in which active terminals are processed may yield different but nonetheless valid lower bounds. Therefore, in each B&B node, if the node still remains open after an initial run of Algorithm 1, the algorithm is executed for the ten different guiding solutions computed during the initialization phase. Although computationally more expensive, this technique makes the B&B procedure much more robust. This technique almost always improves lower bounds, and generates different reduced costs which may cause different parts of the instance graph to be fixed by bound-based reductions.

Reduction tests. Within each B&B node, all alternative-based tests are applied exhaustively in the following order: D1, D2, MA/AMA, MS, SS, LC, NR. Bound-based reductions (BNA, BNE, BNI) are applied after each execution of Algorithm 1. Since one optimal solution needs to be found during the B&B procedure, variable fixing due to bound-based tests is also performed if equality holds for the tested node or arc. This may discard an optimal solution only if UB is already optimal. During the B&B, Test 4 (LC) is only applied for paths of length two. Cost shifting is also re-applied at each B&B node. Concerning Tests 7 and 8, for simplicity reasons, cut nodes and arcs are identified in linear time by computing the set of articulation points of graph $G_U = (V, E)$, i.e., the undirected counterpart of G, in which directed arcs between each pair of nodes are replaced by an undirected edge. The connectivity information in the form of articulation points can also be used to fix to one all cut vertices that separate another fixed terminal from the root. Note that since this approach essentially ignores the direction of arcs, only a subset of all possible reductions for tests MS and SS will be identified. However, since the set of addressed benchmark instances is undirected, this heuristic variant has proven to be sufficient. The implementation could be further improved by using a near-linear time algorithm for the detection of cut nodes in a directed graph (see, e.g., Lengauer and Tarjan 1979).

Branching & node selection. Node-based branching is performed on the set of nodes $V \setminus T_f$, such that in one problem a given node is added to T_f, while it is removed in the other. Our branching and node selection strategies aim at fixing nodes as fast as possible. This strategy encourages further reductions. It also exploits the fact that Algorithm 1 requires less time during its first iterations. So even if many B&B nodes get explored that are unlikely to contain an optimal solution, Algorithm 1 usually requires little time to exceed UB. For this purpose, subproblems are selected based on the maximum lower bound, since these nodes are most likely to be pruned. Similarly, the branching priority is defined to accelerate up this behavior. Nodes are scored based on the number of times branching on them has lead to the subproblem getting pruned. In case of ties, priority is computed based on the largest degree in G_S and subsequently based on largest degree in the current incumbent.

6 Computational results

The presented framework has been implemented in C++ and compiled with GCC 4.9.2. The implementation is single-threaded and uses data structures from the Boost library. All test runs have been performed on a machine with an Intel Xeon CPU (2.5 GHz, 20 cores) and 64GB of memory. Each test run uses one core, with a memory limit of 16GB and a time limit of one hour. Unless noted otherwise, all tables list running times in seconds and relative optimality gaps between the computed lower and upper bounds in percent ($(UB - LB)/UB$). Test runs which exceeded their given time or memory limit are denoted by *TL*, resp. *ML*. The tested benchmark instances include all instances for the PCSTP, MWCS and NWSTP that have been collected during the 11th

DIMACS challenge. Although our framework is capable of handling instances of the STP, due to space reasons no results are reported, since for this problem more specialized frameworks have already been proposed (Fischetti et al. 2014, Pajor et al. 2014, Polzin and Daneshmand 2001), which are unlikely to be outperformed by our methods. Table 6 gives a short overview on instance metrics per data set. Notice the size of NWSTP instances - to the best of our knowledge, this is a first computational study on such large benchmark instances. After transformation to the APCSTP, the larger of the two instances has 205717 nodes, 4932002 arcs, and 54857 terminals.

Table 1: Benchmark instances.

			$\|V\|$			$\|A\|$			$\|T\|$		
problem	*data set*	*#inst.*	*min*	*avg*	*max*	*min*	*avg*	*max*	*min*	*avg*	*max*
PCSTP	CRR	80	500	750	1000	1250	12469	50000	5	140	500
	JMP	34	100	247	400	568	1701	3152	10	46	120
	RANDOM	68	200	4000	14000	3150	64056	224738	200	4000	13999
	HANDSD	10	39600	39600	39600	157408	157408	157408	2073	19135	38348
	HANDSI	10	42500	42500	42500	168950	168950	168950	2293	19905	40702
	HANDBD	14	169800	169800	169800	677102	677102	677102	8854	80065	162862
	HANDBI	14	158400	158400	158400	631616	631616	631616	8572	74675	153756
	PUCNU	18	64	1311	4096	384	19158	57024	8	134	473
	I640	25	640	640	640	1920	100700	408960	8	61	160
	H	14	64	1161	4096	384	12837	49152	32	581	2048
	H2	14	64	1161	4096	384	12837	49152	17	481	2048
RPCSTP	COLOGNE	29	741	1294	1810	12586	23435	33588	3	9	15
MWCS	JMPALMK	72	500	938	1500	5194	17390	41054	499	936	1499
	ACTMOD	8	2034	3933	5226	6670	82311	186788	1351	3557	5225
NWSTP	HIV	2	386	103051	205717	2954	2467478	49320002	35	27446	54857

6.1 Primal heuristics

Table 2 compares the performance of several variants of the implemented primal heuristic. For each variant and data set, the running time and the average gap between the computed and the (previously) best known upper bound is given. Note that negative gap values indicate that, on average, a better upper bound has been computed. All variants apply initial preprocessing, whose running time is included in the reported time. SPH corresponds to the execution of SPHPRUNE, see Section 5.1. SPHDA denotes the same procedure, but applied to G_S instead of G, where G_S is computed by executing Algorithm 1 for the same root node as SPHPRUNE. SPHDAG denotes the application of Algorithm 2. The parameter ε is set to a small value, i.e., $\varepsilon := 0.05$ for instances with integer prizes and costs, and $\varepsilon := 0.005$ otherwise. Finally, SPHDABB denotes the same procedure as the previous variant, followed by a B&B search with a short time limit of ten seconds.

The results indicate that on average already the simple SPH procedure constructs adequate solutions, however it fails to do so on data sets I640, H, H2 and PUCNU. Note that the relatively

Table 2: Performance comparison for primal heuristics. Columns *Pgap* and *time* list average running times in seconds and relative gaps between the computed and the best known upper bound in percent. SPH - construction heuristic applied to $\mathbf{G}$. SPHDA - construction heuristic applied to saturation graph $\mathbf{G_S}$. SPHDAG - construction heuristic applied to perturbed $\mathbf{G_S}$. SPHDAGBB adds a restricted B&B search to SPHDAG.

	SPH		SPHDA		SPHDAG		SPHDAGBB	
data set	*Pgap*	*time*	*Pgap*	*time*	*Pgap*	*time*	*Pgap*	*time*
CRR	1.607	0.2	0.681	0.2	0.139	0.2	0.022	0.3
JMP	0.567	0.1	0.097	0.1	0.087	0.1	0.030	0.1
RANDOM	0.049	0.7	0.013	1.6	0.011	1.8	0.001	2.4
COLOGNE	0.308	0.2	0.547	0.2	0.547	0.2	0.000	0.2
HANDBD	0.012	2.0	-0.027	54.3	-0.027	55.9	-0.038	56.7
HANDBI	0.139	1.9	0.001	30.0	0.001	28.3	-0.005	34.3
HANDSD	0.067	0.4	0.007	3.8	0.018	2.9	0.001	3.4
HANDSI	0.025	0.4	0.005	2.1	0.006	2.7	0.001	3.5
I640	16.031	0.7	2.136	2.4	1.041	3.3	0.511	3.5
H	6.705	0.1	5.394	0.8	3.477	1.4	1.688	7.4
H2	6.766	0.1	5.896	0.8	4.033	1.4	1.625	7.5
PUCNU	6.716	0.1	6.587	0.2	4.820	0.2	2.306	5.2
ACTMOD	0.153	0.2	0.013	1.5	0.011	1.6	0.001	1.9
HIV	0.677	1182.9	0.430	1187.1	0.430	1187.0	0.090	1195.0

large running times on HIV occur due to the initial preprocessing phase. The actual time spent for the heuristic is always below 25 seconds. SPHDA reliably improves the solution quality on almost all data sets, except COLOGNE. However, SPHDA necessitates the execution of Algorithm 1, which notably increases running times on large-scale data sets HANDBI and HANDBD. We note that, even though the gap reduction seems to be small, this additional computational effort turned out to pay off as more reductions are possible and less B&B nodes need to be considered subsequently. Variant SPHDAG further increases solution quality on instance sets where SPHDA fails to obtain average gaps below 1%. Finally, SPHDAGBB reliably improves average solution quality on all data sets, while hardly increasing the average running time in comparison to SPHDAG. Therefore, in all subsequent experiments, SPHDAGBB is applied to compute a starting solution before the performing the B&B.

6.2 Preprocessing & Root node evaluation

Table 3 details average performance for the phases executed before the framework finally begins to branch. These include the initial preprocessing, start heuristic, and root node evaluation. For each phase, the average running time is given in seconds. For phase PREPROCESSING, columns $\%|V|$ and $\%|A|$ list the percentage of the remaining nodes and arcs after reduction tests have been applied exhaustively. Note that for unrooted instances, this includes only a subset of the alternative-based reduction tests detailed in Section 4 (LC, MA). The results indicate that these tests are most

Table 3: Average performance for root node processing. Columns *Pgap* and *Dgap* denote the relative primal and dual optimality gap to the best known upper bound in percent. Column *gap* denotes the relative optimality gap between the computed lower and upper bound. Columns $\%|V|$ and $\%|A|$ denote the percentage of remaining nodes and arcs after preprocessing. Columns *%eval.* and *%open* denote the percentage of all root nodes that have been evaluated and that remain open. The running time of each phase is given in seconds.

	Preprocessing			Heuristic		Root Evaluation						Total	
data set	$\%\|V\|$	$\%\|A\|$	*time*	*Pgap*	*time*	*%eval.*	*%open*	$\%\|V\|$	$\%\|A\|$	*Dgap*	*time*	*gap*	*time*
CRR	95.0	82.7	0.1	0.02	0.1	28.5	0.2	3.8	0.5	0.08	0.0	0.10	0.3
JMP	99.0	96.3	0.0	0.03	0.0	60.0	0.2	0.4	0.1	0.00	0.0	0.03	0.2
RANDOM	64.0	49.1	0.5	0.00	1.3	28.0	0.1	2.1	0.5	0.00	0.1	0.00	2.5
HANDSD	98.4	98.7	0.0	0.00	3.1	58.2	0.1	0.1	0.1	0.00	0.5	0.01	4.0
HANDSI	98.1	98.4	0.0	0.00	3.1	58.2	0.1	0.2	0.1	0.01	0.4	0.01	4.0
HANDBD	98.2	98.5	0.4	-0.04	54.3	56.1	0.0	6.3	6.2	0.08	18.8	0.04	76.6
HANDBI	97.8	98.1	0.4	-0.01	31.9	53.3	0.0	8.7	8.6	0.15	8.9	0.14	47.4
COLOGNE	92.8	94.9	0.0	0.00	0.0	100.0	0.0	0.0	0.0	0.00	0.0	0.00	0.2
I640	100.0	100.0	0.6	0.51	2.8	29.4	2.4	46.6	43.1	0.75	1.2	1.25	4.9
H	100.0	100.0	0.0	1.69	1.6	68.8	8.7	91.4	89.9	3.43	87.5	5.02	94.9
H2	100.0	100.0	0.0	1.63	1.5	67.3	8.2	91.0	88.3	3.66	69.8	5.18	77.3
PUCNU	99.4	99.9	0.0	2.33	0.2	71.1	12.8	69.2	68.1	4.25	1.3	6.34	6.4
ACTMOD	99.3	98.6	0.1	0.00	1.5	77.9	0.4	0.2	0.0	0.00	0.2	0.00	1.9
HIV	81.7	61.1	1115.3	0.09	7.1	100.0	50.0	37.4	16.3	0.01	93.7	0.10	1224.9

effective on the randomly generated data sets CRR and RANDOM, as well as on HIV.

For phase Heuristic, column *Pgap* lists the relative primal optimality gap, i.e., the relative difference between the computed and the best known upper bound. For phase Root Evaluation, note that all tested data sets except COLOGNE and HIV contain unrooted instances (for HIV a fixed terminal is chosen as root). Columns *%eval.* and *%open* list the percentage of all evaluated roots $r \in R \subseteq V$, and the percentage of those that remain open after evaluation, i.e., $LB_r < UB$. Columns $\%|V|$ and $\%|A|$ list average percentages (over all open nodes) of remaining nodes and arcs after additional preprocessing. As in this phase formerly unrooted APCSTP instances are now decomposed into a set of rooted instances, all reduction tests presented in Section 4 can be applied, including the bound-based reduction tests. Column *Dgap* lists the relative dual optimality gap, i.e., the relative difference between the computed lower and the best known upper bound.

Our results indicate that on average only about half of all root nodes need to be evaluated, as all others can be discarded using the techniques described in Section 4. Most importantly, for almost all tested instances, only one root node remains open. The only exception are data sets H, H2 and PUCNU, in which approximately 10% of all potential roots remain open, and the applied reduction tests are much less effective. Similarly, *Dgap* is far worse than *Pgap*. This behavior is a consequence of the fact that data sets H, H2 and PUCNU have been specifically designed with

the aim to be challenging for existing exact methods and reduction tests, as their structure is highly symmetrical. As is evident from the presented results, the techniques applied within the implemented framework are not very well suited for this type of instances, but are nonetheless highly effective for all others. Finally, a noteworthy aspect is the performance of reduction tests during this phase. The running time of the initial preprocessing phase is usually negligible, except on large-scale instances from HIV. In all data sets where lower and upper bound are already very close to the optimum, almost 99% of all arcs can already be fixed. Note that as some of the considered benchmark instances contain over 300.000 arcs, fixing this amount does not necessarily guarantee that the resulting instance is trivial to solve.

6.3 Branch-and-bound

Table 4 summarizes the performance of applying the full B&B procedure. Only data sets are listed in which all instances have been solved to optimality within the given time limit. Minimum, average and maximum numbers of enumerated B&B nodes and running time are reported. The last column lists the average speedup factors with respect to a state-of-the-art ILP-based exact solver for the PCSTP, MWCS and STP by Fischetti et al. (2014), in the following denoted by DIMACS. The instance set I640 has been split into multiple parts, such that I640-0 and I640-1 contain instances i640-001 to i640-045 and i640-101 to i640-145. The most impressive speedup is achieved for COLOGNE, which can be solved without any branching. Also notable are our results for MWCS instances ACTMOD and JMPALMK. Our performance outperforms the framework proposed in El-Kebir and Klau (2014), which applies a sophisticated divide-and-conquer strategy and reduction tests for the MWCS. Furthermore, for many instances our framework manages to find better or equally good solutions in less time than the recent heuristic procedure by Fu and Hao (2014). The latter achieved the best performance in the PCSTP heuristic categories during the recent DIMACS challenge. Detailed results on all benchmark instances are reported in the Appendix.

Tables 5–8 report detailed results on data sets in which not all instances have been solved to optimality within the given time limit. Results on data sets H2 and PUCNU can be found in the Appendix. Columns ROOT describe the relative optimality gap in percent between upper and lower bound, as well as the running time in seconds spent before branching. Columns TOTAL describe information on the complete B&B procedure. Column *#BBn* lists the number of B&B nodes. Column *ub* lists the best computed upper bound. Column *gap* and *time* list optimality gap and running time. The final two columns show gap and running time for DIMACS. For each instance the results of the best approach are marked in bold.

Table 5 lists results on bioinformatics instances for the NWSTP. No results are available for DIMACS, as it only solves instances of the PCSTP, MWCS and STP. Instead, the results of Gamrath et al. (2015) are reported for both instances, denoted by SCIPJACK. Their algorithm solves hiv-2 almost instantly, as does the implemented B&B framework. For hiv-1, they report a primal bound

Table 4: Data sets solved to optimality within one hour by the presented framework and a state-of-the-art exact algorithm for the PCSTP by Fischetti et al. (2014), denoted by DIMACS. Column *#inst* lists the number of instances per data set. Columns *#BBn* and *time* list the minimum, average and maximum number of B&B nodes and running time in seconds, respectively. The last column lists the average speedup achieved by the presented framework with respect to DIMACS. (*) Data sets that contained instances previously unsolved within an hour.

		#BBn			*time*			*avg. speedup w.r.t.*
data set	*#inst*	*min*	*avg*	*max*	*min*	*avg*	*max*	DIMACS
CRR	80	0	29	1751	0.1	0.6	13.4	3.16
JMP	34	0	0	1	0.1	0.2	0.5	7.70
RANDOM	68	0	75	3065	0.1	4.5	99.2	7.76
HANDSD	10	0	2	9	1.3	4.0	13.2	312.11*
HANDSI	10	0	81	811	1.3	5.5	18.9	194.31*
I640-0	25	0	1	13	0.1	2.7	13.5	10.48
I640-1	25	0	47	213	0.1	4.9	19.8	21.86
COLOGNE	29	0	0	0	0.1	0.2	0.5	249.96
ACTMOD	8	0	1	5	0.3	2.0	5.5	2.49
JMPALMK	72	0	0	0	0.1	0.1	0.3	1.94

Table 5: Results for NWSTP HIV instances.

				ROOT		TOTAL				SCIPJACK	
instance	$\|V\|$	$\|A\|$	$\|T\|$	*gap*	*time*	*#BBn*	*ub*	*gap*	*time*	*gap*	*time*
hiv-1	205717	4932002	54857	0.05	2851.4	3	657201.449931	0.05	*TL*	0.0049	72 (hrs.)
hiv-2	386	2954	35	0.00	0.01	0	568.418461	0.00	0.01	0.0000	0.10 (sec.)

of 656970.94 with an optimality gap of 0.0049%, after 72 hrs. of runtime on a machine with 386 GB of memory. No better upper bound could be achieved by the implemented B&B, even after increasing the memory and time limit. However, note that our results are reported for a time limit of one hour and a memory limit of 16GB, while Gamrath et al. (2015) do not report the time until their best upper bound has been found, and state that the instance hiv-1 exceeded the available memory on a machine with 48GB.

Table 6 lists results on instances for image recognition of hand-written text. The instances are planar grid graphs, on which cursive letters are represented as the prize of nodes. Due to their size and structure, most of these instances remain unsolved by state-of-the-art ILP solvers based on branch-and-cut. On the contrary, our B&B solves all but three of these instances to proven optimality and consistently outperforms DIMACS on this instance set.

Table 7 lists results on I640 instances. They have been proposed for the STP, and then converted to the PCSTP. Originally, they have been created to resist common preprocessing tests. Only the latter half of these instances are given, since they contain instances which could not be solved to

optimality by either methods. Results suggest that the bound-based reductions are crucial to solve instances with many arcs including some instances that could not be solved by the ILP solver (321 to 325, and 241).

Table 8 lists results on hypercube instances H on which reductions are usually not effective. They also contain a high level of symmetry and the obtained dual gaps are rather bad. Successful methods based on ILP (e.g., Fischetti et al. 2014) have used node-based models for instances with uniform costs (denoted by hc*u). These methods are able to process much more branch-and-bound nodes in the same time for dense instances and benefit from automatically detected general-purpose cuts. Results show that the proposed B&B is not very well suited for solving this class of instances as only the four smallest instances are solved. Its performance is thus complementary to the framework by Fischetti et al. (2014), achieving significant speedups in case reductions are possible, but proving much less scalable in all other cases.

7 Conclusions

We have introduced the APCSTP as a generalization of several well-known network design problems, and proposed a B&B framework based on a new dual ascent algorithm for the rooted APCSTP. The dual information obtained from the algorithm is exploited within bound-based reduction tests and to guide the construction of primal solutions. The framework is further complemented by a set of simple reduction tests. Unrooted instances are decomposed into a set of rooted instances by an enumerative scheme, which exploits potential symmetries within the instance structure.

The framework's performance has been evaluated on an extensive set of benchmark instances from literature, including known test instances for the PCSTP, MWCS and NWSTP, which are handled by transformation to the APCSTP. Results indicate that for almost all instances, the computed lower and upper bounds are very tight, allowing bound-based reduction tests to fix large parts of the input graph, often already at the root node. In the best case, speedups of two orders of magnitude are achieved in comparison with a state-of-the-art exact algorithm based on a commercial MIP solver (see Fischetti et al. (2014)). In many cases, the framework even outperforms recent heuristic methods for the PCSTP, where it computes optimal solutions within seconds which are not found by the heuristics even after one hour of computation. In addition, optimal solutions have been computed for some of the (previously unsolved) largest instances considered in the recent DIMACS Challenge on Steiner Trees (with more than 150 000 nodes and 650 000 edges). For the largest instance from the DIMACS Challenge with $\approx$ 200 000 nodes and almost 2.5 million of edges, we provide a solution of 0.05% optimality gap obtained within one hour of computing time on a single core. For the family of PUC instances (that have been artificially generated so as to be "resistant" to bound-based reductions), the state-of-the-art solver remains the branch-and-cut code recently proposed in Fischetti et al. (2014).

We point out that some of the presented techniques may also be useful by themselves to improve

the performance of ILP solvers. For example, the cuts computed by the dual ascent algorithm can be used to initialize a cutting plane procedure, or the bound-based reductions may be used to remove parts of the graph which otherwise would slow down the solver.

Finally, since the network design problems addressed in this work are used for modeling various real-world applications (e.g., in bioinformatics), the presented B&B framework will also be made publicly available.

Acknowledgement

The authors thank Matteo Fischetti for useful discussions on this and related topics. M. Luipersbeck acknowledges the support of the University of Vienna through the uni:docs fellowship programme. M. Leitner has been supported by the Vienna Science and Technology Fund (WWTF) through project ICT15-014. M. Sinnl is supported by the Austrian Research Fund (FWF, Project P 26755-N19).

References

11th DIMACS challenge. 11th DIMACS Implementation Challenge: Steiner Tree Problems, 2014. `http://dimacs11.zib.de/`.

Boost. Boost C++ libraries, 2016. `http://boost.org`.

C.-Y. Chen and K. Grauman. Efficient activity detection with max-subgraph search. In *Conference on Computer Vision and Pattern Recognition*, pages 1274–1281. IEEE, 2012.

M. T. Dittrich, G. W. Klau, A. Rosenwald, T. Dandekar, and T. Müller. Identifying functional modules in protein–protein interaction networks: an integrated exact approach. *Bioinformatics*, 24 (13):i223–i231, 2008.

C. W. Duin. *Steiner's problem in graphs*. PhD thesis, University of Amsterdam, 1993.

C. W. Duin and A. Volgenant. Reduction tests for the Steiner problem in graphs. *Networks*, 19(5): 549–567, 1989. ISSN 1097-0037.

M. El-Kebir and G. W. Klau. Solving the maximum-weight connected subgraph problem to optimality. 11th DIMACS challenge workshop, 2014.

M. Fischetti, M. Leitner, I. Ljubić, M. Luipersbeck, M. Monaci, M. Resch, D. Salvagnin, and M. Sinnl. Thinning out Steiner trees: a node-based model for uniform edge costs. 2014.

Z.-H. Fu and J.-K. Hao. Knowledge guided tabu search for the prize collecting Steiner tree problem in graphs. 11th DIMACS challenge workshop, 2014.

G. Gamrath, T. Koch, S. J. Maher, D. Rehfeldt, and Y. Shinano. SCIP-Jack – a solver for STP and variants with parallelization extensions. 2015.

M. X. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24(2):296–317, 1995.

C. Hegde, P. Indyk, and L. Schmidt. A fast, adaptive variant of the Goemans-Williamson scheme for the prize-collecting Steiner tree problem. 11th DIMACS challenge workshop, 2014.

D. S. Johnson, M. Minkoff, and S. Phillips. The prize collecting Steiner tree problem: theory and practice. In D. B. Shmoys, editor, *Symposium on Discrete Algorithms*, pages 760–769. ACM/SIAM, 2000. ISBN 0-89871-453-2.

T. Koch and A. Martin. Solving Steiner tree problems in graphs to optimality. *Networks*, 32(3): 207–232, 1998.

T. Lengauer and R. E. Tarjan. A fast algorithm for finding dominators in a flowgraph. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 1(1):121–141, 1979.

I. Ljubić, R. Weiskircher, U. Pferschy, G. W. Klau, P. Mutzel, and M. Fischetti. An algorithmic framework for the exact solution of the prize-collecting Steiner tree problem. *Mathematical Programming*, 105(2-3):427–449, 2006.

T. Pajor, E. Uchoa, and R. F. Werneck. A robust and scalable algorithm for the Steiner problem in graphs. 2014. 11th DIMACS challenge workshop.

M. Poggi de Aragão and R. F. Werneck. On the implementation of MST-based heuristics for the Steiner problem in graphs. In D. M. Mount and C. Stein, editors, *ALENEX*, volume 2409 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2002.

T. Polzin and S. V. Daneshmand. Improved algorithms for the Steiner problem in networks. *Discrete Applied Mathematics*, 112(1):263–300, 2001.

H. Takahashi and A. Matsuyama. An approximate solution for the Steiner problem in graphs. *Math. Japonica*, 24(6):573–577, 1980.

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

R. T. Wong. A dual ascent approach for Steiner tree problems on a directed graph. *Mathematical Programming*, 28(3):271–287, 1984. ISSN 0025-5610.

Table 6: Results for PCSTP data sets handsi, handsd, handbi, handbd.

				Root		Total				Dimacs	
instance	$\|V\|$	$\|A\|$	$\|T\|$	*gap*	*time*	*#BBn*	*ub*	*gap*	*time*	*gap*	*time*
handsd01	42500	168950	42077	0.00	1.6	0	171.636766	**0.00**	**1.9**	0.00	77.1
handsd02	42500	168950	2324	0.02	9.7	9	159.751395	**0.00**	**10.2**	1.81	*TL*
handsd03	42500	168950	41290	0.00	1.2	0	31.306275	**0.00**	**1.4**	0.00	47.3
handsd04	42500	168950	4362	0.01	13.0	1	491.733164	**0.00**	**13.2**	1.60	*TL*
handsd05	42500	168950	41416	0.00	1.4	0	21.937611	**0.00**	**1.6**	0.00	62.3
handsd06	42500	168950	4150	0.03	4.2	5	279.903130	**0.00**	**4.6**	0.00	948.8
handsd07	42500	168950	40978	0.00	1.7	0	11.804120	**0.00**	**2.1**	0.00	68.4
handsd08	42500	168950	4379	0.00	1.4	0	143.237729	**0.00**	**1.6**	0.00	610.3
handsd09	42500	168950	39877	0.00	2.3	0	3.818683	**0.00**	**2.6**	0.78	*TL*
handsd10	42500	168950	3562	0.00	1.0	0	1034.767359	**0.00**	**1.3**	0.00	24.5
handsi01	39600	157408	39331	0.00	1.0	0	295.453616	**0.00**	**1.3**	0.00	49.1
handsi02	39600	157408	2076	0.06	3.3	1	125.429411	**0.00**	**3.6**	0.00	2239.6
handsi03	39600	157408	38386	0.00	1.2	0	56.149422	**0.00**	**1.4**	0.00	57.4
handsi04	39600	157408	4049	0.01	18.1	1	722.508197	**0.00**	**18.4**	0.00	2863.3
handsi05	39600	157408	38882	0.00	1.2	0	35.043506	**0.00**	**1.5**	0.00	41.8
handsi06	39600	157408	3982	0.00	2.6	0	452.953621	**0.00**	**2.8**	0.00	775.5
handsi07	39600	157408	38537	0.00	1.3	0	18.410135	**0.00**	**1.5**	0.00	56.4
handsi08	39600	157408	3957	0.00	1.1	0	229.529930	**0.00**	**1.4**	0.00	532.4
handsi09	39600	157408	37753	0.01	3.9	1	5.962166	**0.00**	**4.3**	1.49	*TL*
handsi10	39600	157408	3304	0.04	3.0	811	1803.697508	**0.00**	**18.9**	0.00	482.9
handbd01	169800	677102	168211	0.00	87.8	0	728.963591	**0.00**	**88.8**	1.26	*TL*
handbd02	169800	677102	8877	0.04	54.3	35	296.496486	**0.00**	**67.0**	5.14	*TL*
handbd03	169800	677102	163401	0.00	11.9	0	135.070605	**0.00**	**12.8**	0.00	554.7
handbd04	169800	677102	16966	0.02	421.8	391	1813.959161	**0.00**	**453.6**	5.43	*TL*
handbd05	169800	677102	165802	0.00	10.1	0	105.474688	**0.00**	**11.0**	0.00	704.6
handbd06	169800	677102	16431	0.04	190.8	545	1528.765436	**0.00**	**274.4**	2.75	*TL*
handbd07	169800	677102	165126	0.00	15.6	0	77.861959	**0.00**	**16.6**	0.00	2941.8
handbd08	169800	677102	17464	0.02	38.9	55	1368.166769	**0.00**	**44.8**	1.94	*TL*
handbd09	169800	677102	164501	0.00	13.6	0	62.717160	**0.00**	**14.5**	0.00	1004.4
handbd10	169800	677102	17262	0.00	17.5	0	1137.429734	**0.00**	**18.9**	1.28	*TL*
handbd11	169800	677102	163788	0.00	14.1	0	46.772533	**0.00**	**15.0**	0.00	719.7
handbd12	169800	677102	16427	0.00	15.2	0	321.204744	**0.00**	**16.5**	0.22	*TL*
handbd13	169800	677102	158381	0.41	227.4	2479	13.188859	**0.41**	***TL***	2.67	*TL*
handbd14	169800	677102	16371	0.00	0.6	0	4379.104236	**0.00**	**1.5**	0.00	45.5
handbi01	158400	631616	157385	0.00	216.0	0	1358.563381	**0.00**	**217.0**	1.10	*TL*
handbi02	158400	631616	8589	0.02	32.3	33	531.810883	**0.00**	**37.5**	2.71	*TL*
handbi03	158400	631616	154148	0.00	7.9	0	243.134201	**0.00**	**8.8**	0.00	1246.2
handbi04	158400	631616	16288	0.05	171.7	29518	3202.185740	**0.04**	***TL***	4.22	*TL*
handbi05	158400	631616	155695	0.00	10.4	0	184.467331	**0.00**	**11.2**	0.00	916.3
handbi06	158400	631616	16002	0.05	89.4	689	2921.544716	**0.00**	**204.8**	2.69	*TL*
handbi07	158400	631616	155449	0.01	11.8	1	150.974258	**0.00**	**12.7**	0.00	1265.8
handbi08	158400	631616	15530	0.01	12.8	27	2270.284625	**0.00**	**16.4**	1.44	*TL*
handbi09	158400	631616	154731	0.00	11.7	0	107.768806	**0.00**	**12.6**	0.00	985.8
handbi10	158400	631616	15870	0.00	11.7	0	1874.292962	**0.00**	**12.7**	0.37	*TL*
handbi11	158400	631616	153918	0.00	18.4	0	68.944709	**0.00**	**19.4**	0.11	*TL*
handbi12	158400	631616	15546	0.00	6.0	0	138.257023	**0.00**	**7.1**	0.00	3378.7
handbi13	158400	631616	140634	1.87	110.8	205	4.274497	**1.87**	**ML**	2.55	*TL*
handbi14	158400	631616	14551	0.00	3.4	0	7881.768740	**0.00**	**4.3**	0.00	47.7

Table 7: Results for PCSTP instances i640-201 to i640-345.

				Root		Total				Dimacs		
instance	$\|V\|$	$\|A\|$	$\|T\|$	*gap*	*time*	*#BBn*	*ub*	*gap*	*time*	*#BBn*	*gap*	*time*
i640-201	640	1920	50	0.48	0.1	7	14372	**0.00**	**0.1**	0	0.00	0.3
i640-202	640	1920	49	0.03	0.1	1	15059	**0.00**	**0.1**	0	**0.00**	**0.1**
i640-203	640	1920	50	0.00	0.1	0	13848	**0.00**	**0.1**	0	0.00	0.3
i640-204	640	1920	50	0.34	0.1	15	13108	**0.00**	**0.1**	0	0.00	0.4
i640-205	640	1920	50	0.99	0.1	63	15308	**0.00**	**0.2**	0	0.00	1.6
i640-211	640	8270	50	5.08	0.5	128343	11109	0.00	1865.1	11205	**0.00**	**439.2**
i640-212	640	8270	50	3.66	0.5	5673	10351	0.00	87.4	1154	**0.00**	**69.6**
i640-213	640	8270	50	4.57	0.5	6141	10388	**0.00**	**78.3**	1833	0.00	93.5
i640-214	640	8270	50	2.46	0.4	621	10675	**0.00**	**12.4**	31	0.00	14.8
i640-215	640	8270	50	4.90	0.6	47995	10740	0.00	663.2	3760	**0.00**	**123.3**
i640-221	640	408960	50	0.43	12.7	23	8400	**0.00**	**14.8**	511	0.00	1486.7
i640-222	640	408960	50	0.35	10.3	5	8993	**0.00**	**10.8**	60	0.00	877.4
i640-223	640	408960	50	0.37	9.1	5	9210	**0.00**	**10.1**	246	0.00	774.8
i640-224	640	408960	49	0.33	10.1	11	8870	**0.00**	**10.9**	455	0.00	1300.6
i640-225	640	408960	49	0.00	8.4	0	8386	**0.00**	**8.7**	164	0.00	878.8
i640-231	640	2560	50	2.53	0.2	361	14279	**0.00**	**1.6**	29	0.00	14.7
i640-232	640	2560	50	1.45	0.1	31	13526	**0.00**	**0.3**	9	0.00	3.6
i640-233	640	2560	49	1.47	0.1	31	12948	**0.00**	**0.3**	0	0.00	3.5
i640-234	640	2560	50	0.97	0.1	23	13645	**0.00**	**0.2**	0	0.00	1.4
i640-235	640	2560	50	0.36	0.1	5	12650	**0.00**	**0.1**	0	0.00	1.0
i640-241	640	81792	50	1.30	5.5	1751	9716	**0.00**	**81.0**	12581	0.24	*TL*
i640-242	640	81792	50	1.48	4.9	1157	9250	**0.00**	**53.8**	1560	0.00	439.8
i640-243	640	81792	50	1.57	5.2	1125	9315	**0.00**	**58.0**	5846	0.00	1151.6
i640-244	640	81792	50	2.81	5.2	3205	8950	**0.00**	**114.8**	14499	0.00	1966.0
i640-245	640	81792	50	1.95	5.3	2433	9448	**0.00**	**107.3**	9587	0.00	1826.4
i640-301	640	1920	159	0.72	0.2	139	42701	**0.00**	**0.9**	0	**0.00**	**0.9**
i640-302	640	1920	160	0.65	0.2	191	42606	**0.00**	**0.4**	0	0.00	1.2
i640-303	640	1920	160	0.10	0.1	19	41286	**0.00**	**0.2**	0	0.00	0.8
i640-304	640	1920	159	0.73	0.1	249	42050	**0.00**	**0.7**	0	**0.00**	**0.7**
i640-305	640	1920	160	0.56	0.2	421	42798	**0.00**	**1.1**	8	0.00	2.6
i640-311	640	8270	160	4.13	0.9	151554	33953	4.13	*TL*	72680	**0.68**	***TL***
i640-312	640	8270	160	3.76	0.8	222446	33096	3.70	*TL*	51318	**0.83**	***TL***
i640-313	640	8270	159	2.88	0.6	182904	32628	2.85	*TL*	59731	**0.00**	**3198.8**
i640-314	640	8270	160	3.07	0.6	197307	33108	3.07	*TL*	71012	**0.73**	***TL***
i640-315	640	8270	159	3.35	0.6	173883	32821	3.35	*TL*	83624	**0.62**	***TL***
i640-321	640	408960	160	1.03	47.8	25615	28787	**0.00**	**2671.2**	665	0.36	*TL*
i640-322	640	408960	160	0.87	40.9	6583	28456	**0.00**	**731.2**	835	0.31	*TL*
i640-323	640	408960	160	0.79	33.0	3163	28153	**0.00**	**363.9**	1106	0.26	*TL*
i640-324	640	408960	160	0.95	41.6	16955	28746	**0.00**	**2541.3**	825	0.26	*TL*
i640-325	640	408960	160	0.85	30.3	3195	28385	**0.00**	**457.0**	849	0.29	*TL*
i640-331	640	2560	160	0.49	0.2	871	39315	**0.00**	**3.5**	8	0.00	7.5
i640-332	640	2560	160	0.96	0.2	677	39030	**0.00**	**5.2**	3	0.00	6.0
i640-333	640	2560	160	1.78	0.4	15455	39775	0.00	117.4	324	**0.00**	**29.6**
i640-334	640	2560	160	1.13	0.2	3529	39338	0.00	29.9	11	**0.00**	**4.2**
i640-335	640	2560	160	1.33	0.3	21329	39601	**0.00**	**187.2**	8346	0.00	771.4
i640-341	640	81792	160	1.39	8.4	73067	29902	1.39	*TL*	6181	**0.79**	***TL***
i640-342	640	81792	160	1.17	8.4	84008	29936	1.05	*TL*	3739	**0.56**	***TL***
i640-343	640	81792	160	1.76	8.1	147786	30324	1.53	*TL*	2536	**0.58**	***TL***
i640-344	640	81792	156	1.47	8.4	126109	30131	1.30	*TL*	7022	**0.60**	***TL***
i640-345	640	81792	160	1.28	8.6	107953	29991	**0.68**	***TL***	2691	0.88	*TL*

Table 8: Results for PCSTP instances h.

				Root		Total				Dimacs		
instance	$\|V\|$	$\|A\|$	$\|T\|$	*gap*	*time*	*#BBn*	*ub*	*gap*	*time*	*#BBn*	*gap*	*time*
hc6p	64	384	32	4.85	0.0	416	3908	**0.00**	**0.4**	1324	0.00	1.5
hc6u	64	384	32	0.00	0.0	0	36	**0.00**	**0.1**	10	**0.00**	**0.1**
hc7p	128	896	64	5.25	0.1	104904	7721	**0.00**	**231.2**	61061	0.00	588.4
hc7u	128	896	64	5.48	0.1	1687	72	0.00	1.7	322	**0.00**	**0.2**
hc8p	256	2048	128	5.05	0.3	562769	15422	5.05	*TL*	178611	**1.07**	***TL***
hc8u	256	2048	128	4.14	10.1	1668275	143	2.80	*TL*	18410	**0.00**	**12.6**
hc9p	512	4608	256	4.19	1.2	214179	30318	4.19	*TL*	36536	**1.69**	***TL***
hc9u	512	4608	256	5.88	10.4	605820	289	5.88	*TL*	1001800	**0.00**	**2551.3**
hc10p	1024	10240	512	4.66	19.6	105097	60715	4.66	*TL*	7193	**2.71**	***TL***
hc10u	1024	10240	512	6.60	13.2	251115	576	6.60	*TL*	376031	**1.25**	**2187.0**
hc11p	2048	22528	1024	4.48	91.7	42501	120418	4.48	*TL*	1129	**3.42**	***TL***
hc11u	2048	22528	1024	6.86	42.3	78523	1152	6.86	*TL*	203282	**1.34**	**2032.8**
hc12p	4096	49152	2048	5.46	855.6	19139	241162	5.46	*TL*	15	**3.55**	***TL***
hc12u	4096	49152	2048	7.35	287.2	33090	2299	7.35	*TL*	55328	**1.48**	***TL***