TECHNISCHE UNIVERSITÄT WIEN
Institut für Computergraphik und Algorithmen

Solving the Prize-Collecting Steiner Tree Problem to Optimality

Ivana Ljubi, René Weiskircher, Ulrich Pferschy, Gunnar Klau, Petra Mutzel and Matteo Fischetti

Forschungsbericht / Technical Report

TR–186–1–04–05

8. Oktober 2004

Favoritenstraße 9-11 / E186, A-1040 Wien, Austria
Tel. +43 (1) 58801-18601, Fax +43 (1) 58801-18699
www.cg.tuwien.ac.at

Mathematical Programming manuscript No.
(will be inserted by the editor)

Ivana Ljubić[1] · René Weiskircher[1] · Ulrich Pferschy[2] · Gunnar Klau[1] ·

Petra Mutzel[1] · Matteo Fischetti[3]

Solving the Prize-Collecting Steiner Tree Problem to Optimality*

Received: date / Revised version: date

Abstract. The Prize-Collecting Steiner Tree Problem (PCST) on a graph with edge costs and vertex profits asks for a subtree minimizing the sum of the total cost of all edges in the subtree plus the total profit of all vertices **not** contained in the subtree. PCST appears frequently in the design of utility networks where profit generating customers and the network connecting them have to be chosen in the most profitable way.

Our main contribution is the formulation of an integer linear program on a directed graph model based on connectivity inequalities corresponding to cuts in the graph. The main advantage of this model is the efficient separation of sets of violated inequalities by a maximum flow algorithm.

The new approach manages to solve all benchmark instances from the literature to optimality, including eight of them for which the optimum was not known. Compared to a recent algorithm by Lucena and Resende [21], our new method is faster by more than two orders of magnitude. We also introduce two new classes of more challenging instances and reach satisfying results for most of them.

Key words. Branch-and-Cut – Steiner Arborescence – Network Design

---

Vienna University of Technology, Favoritenstr. 9-11, A-1040 Vienna, Austria

University of Graz, Universitätsstr. 15, A-8010 Graz, Austria

University of Padova, via Gradenigo 6/a, I-35131 Padova, Italy

Mathematics Subject Classification (1991): 20E28, 20G40, 20C20

* The first author's research was supported by the Doctoral Scholarship Program of the Austrian Academy of Sciences (DOC) and partially supported by the Austrian Science Fund (FWF), grant P16263-N04.

1. Introduction

The recent deregulation of public utilities such as electricity and gas in Austria has shaken up the classical business model of energy companies and opened up the way towards new opportunities. Of particular interest in this field is the planning and expansion of district heating networks. This area of energy distribution is characterized by extremely high investment costs but also by an unusually loyal customer base and limited competition. Moreover, the required reduction of greenhouse emissions forces many energy companies to seek ways of improving their ecological balance sheet. A very attractive possibility to meet this goal is the use of biomass for heat generation. The combination of these two factors has made the planning of heating networks one of the major challenges for companies in this field [16].

In a typical planning scenario the input is a set of potential customers with known or estimated heat demands (represented by discounted future profits), and a potential network for laying the pipes (which is usually identical to the street network of the district or town). Costs of the network are dominated by labor and right-of-way charges for laying the pipes and the costs for building the heating plant.

Essentially, the decision process faced by a profit oriented company consists of two parts: On one hand, a subset of particular profitable customers has to be selected, on the other hand, a network has to be designed to connect all selected customers in a cost-efficient way to the heating plant. The natural trade-off between maximizing the sum of profits over all selected customers and minimizing the cost of the network leads to a prize-collecting objective function.

We can formulate this problem as an optimization problem on an undirected graph $G = (V, E, c, p)$, where the vertices V are associated with profits, $p : V \to \mathbb{R}^{\geq 0}$, and the edges E with costs, $c : E \to \mathbb{R}^{\geq 0}$. The graph in our application corresponds to the local street map, with the edges representing street segments and vertices representing street intersections and the location of potential customers. The profit p associated with a vertex is an estimate of the potential gain of revenue caused by that customer being connected to the network and receiving its service. Vertices corresponding to street intersections have profit zero. The cost c associated with an edge is the cost of establishing the connection, i.e., of laying the pipe on the corresponding street segment.

The formal definition of the problem can be given as follows:

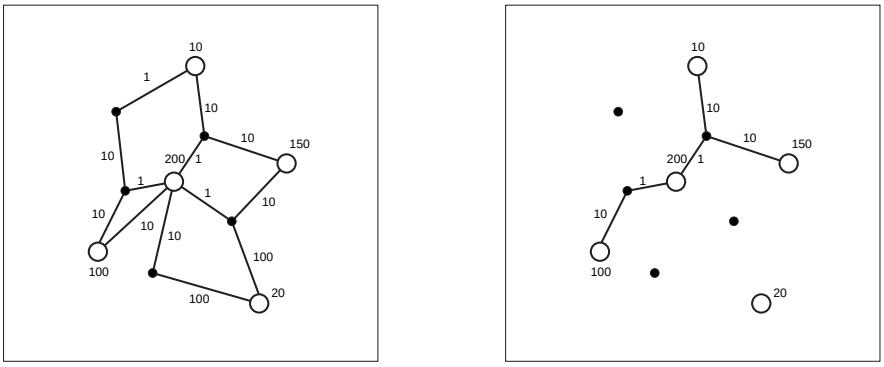
Definition 1 (Prize-Collecting Steiner Tree Problem, PCST). *Let $G = (V, E, c, p)$ be an undirected vertex- and edge-weighted graph as defined above. The* Linear Prize-Collecting Steiner Tree problem *(PCST) consists of finding a connected subgraph $T = (V_T, E_T)$ of G, $V_T \subseteq V$, $E_T \subseteq E$ that maximizes*

$$\mathit{profit}(T) = \sum_{v \in V_T} p(v) - \sum_{e \in E_T} c(e) \ . \tag{1}$$

It is easy to see that every optimal solution T will be a tree. Otherwise removing any edge from a cycle in T would increase $\mathit{profit}(T)$ without violating connectivity of T. Throughout this paper we will distinguish between *customer vertices*, defined as

$$R = \{v \in V \mid p(v) > 0\} \ ,$$

and *non-customer vertices* (corresponding to street intersections) with the assumption that $R \neq \emptyset$. Figure 1 illustrates an example of a PCST instance and a feasible solution for that instance.



(a) (b)

Fig. 1. Example of a PCST instance. Each connection has fixed costs, hollow circles and filled circles represent customer and non-customer vertices, respectively (Fig. 1(a)). Figure 1(b) shows a feasible, but not optimal solution of PCST.

The profit function given above is known in the literature as a function describing the *Net Worth Maximization* Problem (NW) [17]. In the so-called *Goemans and Williamson Minimization* Problem (GW) [15] the goal is to find a subtree $T = (V_T, E_T)$ that minimizes the following function:

$$GW(T) = \sum_{v \notin V_T} p(v) + \sum_{e \in E_T} c(e) \ . \tag{2}$$

Here, $p(v)$ is interpreted as penalty for *not* connecting a vertex v. As far as optimization is concerned, the NW and GW formulations are equivalent, since for every subtree T of G, their objective functions add up to the total sum of profits in G. In this paper we are going to concentrate on minimizing (2) as an objective function, as it has been considered in the literature before (see [15, 21, 3]).

In practice, we often face additional side constraints. The planning problem of the heating network clearly requires that the heating plant is connected to the network. This can be modeled as a PCST by introducing a special vertex for the plant with a very high profit. In general, the *rooted prize-collecting Steiner tree problem* (RPCST), is defined as a variant of PCST with an additional source vertex $v_s \in V$ (representing a depot or repository) which must be part of every feasible solution T.

In the next section we give a short overview of previous work on PCST and some of its relatives. Preprocessing, which helps to significantly reduce the size of many instances, is treated in Section 3. Different ILP models for PCST are presented and discussed in Section 4. In Section 5 we introduce our cut-based ILP model and describe how it can be solved in a branch-and-cut framework in Section 6. Extensive computational experiments on instances from the literature and on new instances are reported in Section 7. It turns out that all the former can be solved to optimality within a few seconds while also the latter are successfully attacked.

2. Previous Work

In 1987, Segev [26] introduced the so called *Node Weighted Steiner Tree Problem* (NWST) – the Steiner tree problem with vertex weights in addition to regular edge weights in which the sum of edge-costs and vertex-weights is minimized. His contribution concerns a special case of NWST, called the *single point weighted Steiner tree* problem (SPWST), where we are given a special vertex to be included in the solution. The weights on the remaining vertices are non-positive profit values, while non-negative weights on the edges reflect the costs incurred in obtaining or collecting these profits.

Negating the vertex weights to make them positive and thus subtracting them from the edge costs in the objective function, immediately yields the minimization version of (1). Thus, the optimization of SPWST is equivalent to RPCST.

The PCST has been introduced by Bienstock et al. [2], where a factor 3 approximation algorithm has been proposed. Several other approximation algorithms have been developed. Goemans and Williamson presented in [15] an approximation algorithm which runs in $O(n^3 \log n)$ time ($n := |V|$), and yields solutions within a factor of $2-\frac{1}{n-1}$ of optimality. This has been improved in Johnson et al. [17], where a $(2-\frac{1}{n-1})$–approximation algorithm with $O(n^2 \log n)$ running time has been proposed. The new algorithm of Feofiloff et al. [12] achieves a ratio of $2 - \frac{2}{n}$ within the same time.

Recently, two metaheuristic approaches for PCST have been developed: Canuto et al. [3] proposed a multi-start local-search-based algorithm with perturbations; Klau et al. [18] developed an evolutionary algorithm with incorporated local improvement for the problem.

2.1. Lower Bounds and Polyhedral Studies

Both Section 4 and 5 are devoted to ILP-formulations for PCST. Therefore, we will in this section only point out references without going into details.

In [26], Segev presented single- and multi-commodity flow formulations for SPWST (cf. Section 4.2). Furthermore, the author developed two bounding procedures based on Lagrangian relaxations of the corresponding flow formulations which were embedded in a branch-and-bound procedure. In addition, heuristics to compute feasible solutions were also included. Benchmark instances with up to 40 vertices were tested.

Fischetti [13] studied the facial structure of a generalization of the problem, the so-called *Steiner arborescence* (or *directed Steiner tree*) problem and pointed out that the NWST can be transformed into it. The author considered several classes of valid inequalities and introduced a new inequality class with arbitrarily large coefficients, showing that all of them define distinct facets of the underlying polyhedron.

Goemans provided in [14] a theoretical study on the polyhedral structure of the NWST and showed that this characterization is complete in case the input graph is series-parallel. Here, SPWST, i.e. RPCST, appears as the r-tree problem.

Engevall et al. [11] proposed another ILP formulation for the NWST, based on the *shortest spanning tree* problem formulation, introduced originally by Beasley [1] for the Steiner tree problem. In their formulation, besides the given root vertex r, an artificial root vertex 0 is introduced, and an edge between vertex 0 and r is set. They searched for a tree with additional constraints: each vertex v connected to vertex 0 must have degree one. The solution is interpreted so that the vertices adjacent to vertex 0 are not taken as a part of the final solution. For the description of the tree, the authors use a modification of the generalized subtour elimination constraints (cf. Section 4.1). For finding good lower bounds, the authors use a Lagrangian heuristic and subgradient procedure based on the shortest spanning tree formulation. Experimental results done for instances with up to 100 vertices indicated that the new approach outperformed Segev's algorithm.

Lucena and Resende [21] presented a cutting plane algorithm for the PCST based on generalized subtour elimination constraints (see again Section 4.1). Their algorithm contains basic reduction steps similar to those already given by Duin and Volgenant [9], and was tested on two groups of benchmark instances: the first group contains instances

adopted from Johnson et al. [17], ranging from 100 vertices and 284 edges to 400 vertices and 1 507 edges. The second group is derived from the Steiner problem instances (series C and D) of the OR-Library[1] with the size ranging from 500 vertices and 625 edges to 1 000 vertices and 25 000 edges. The proposed algorithm solved many of the considered instances to optimality, but not all of them (cf. Section 7).

3. Preprocessing

In this section, we briefly describe reduction techniques adopted from the work of Duin and Volgenant [9] for the NWST, which have been partially used also in [21]. From the implementation point of view, we transform the graph $G = (V, E, c, p)$ into a reduced graph $G' = (V', E', c', p')$ by applying the steps described below and maintain a *backmapping* function to transform each feasible solution T' of G' into a feasible solution T of G.

Least-Cost Test Let d_{ij} represent the shortest path length between any two vertices i and j from V (considering only edge-costs). If $\exists e = (i, j)$ such that $d_{ij} < c_{ij}$ then edge e can simply be discarded from G.

Degree-l Test Consider a vertex $v \notin R$ of degree $l \geq 3$, connected to vertices from $Adj(v) = \{v_1, v_2, \ldots, v_l\}$. For any subset $K \subset V$, denote with $\mathrm{MST}_d(K)$, the minimum spanning tree of K with distances d_{ij}. If

$$\mathrm{MST}_d(K) \leq \sum_{w \in K} c_{vw}, \quad \forall K \subseteq Adj(v), \quad |K| \geq 3, \tag{3}$$

[1] OR-library: J. E. Beasley, http://mscmga.ms.ic.ac.uk/info.html

then v's degree in an optimal solution must be zero or two. Hence, we can remove v from G by replacing each pair (v_i, v), (v, v_j) with (v_i, v_j) either by adding a new edge $e = (v_i, v_j)$ of cost $c_e = c_{v_i v} + c_{v v_j} - p_v$ or in case e already exists, by defining $c_e = \min\{c_e, c_{v_i v} + c_{v v_j} - p_v\}$.

It is straightforward to apply a simplified version of this test to all vertices $v \in V$ with $l = 1$ and $l = 2$.

Minimum Adjacency Test This test is also known as $V \setminus K$ *reduction test* from [9]. If there are adjacent vertices $i, j \in R$ such that:

$$\min\{p_i, p_j\} - c_{ij} > 0 \text{ and } c_{ij} = \min_{it \in E} c_{it},$$

then i and j can be fused into one vertex of weight $p_i + p_j - c_{ij}$.

Summary of the Preprocessing Procedure We apply the steps described above iteratively, as long as any of them changes the input graph. The total number of iterations is bounded by the number of edges in G. Each iteration is dominated by the time complexity of the least-cost test, i.e., by the computation of all-pair shortest paths, which is $O(|E||V| + |V|^2 \log |V|)$. Thus, the preprocessing procedure requires $O(|E|^2|V| + |E||V|^2 \log |V|)$ time in the worst case, in which the input graph would be reduced to a single vertex. However, in practice, the running time is much lower, as documented in Section 7. The space complexity of preprocessing does not exceed $O(|E|^2)$.

4. ILP Formulations of the Problem

In this section we present an ILP-formulation based on generalized subtour elimination constraints and two flow formulations based on the representation of the solutions as rooted trees.

For every formulation (P) the optimal solution value of the resulting LP-relaxation will be denoted by $c(LP_P)$.

4.1. Formulation Based on Generalized Subtour Elimination Constraints

This formulation has been used in Lucena and Resende [21] and is based on the description of the tree structure of the solution. To every subtree $T' = (V'_T, E'_T)$ of the input graph $G' = (V', E', c', p')$, we associate two incidence vectors:

$$X_{ij} = \begin{cases} 1 & (i,j) \in E'_T \\ 0 & \text{otherwise} \end{cases} \quad \forall (i,j) \in E', \quad y_i = \begin{cases} 1 & i \in V'_T \\ 0 & \text{otherwise} \end{cases} \quad \forall i \in V'$$

The PCST can be formulated as the following ILP (the edge-variables are denoted by a capital X to distinguish them from the variables for the directed formulations we

will introduce later):

$$(GSEC) \quad \min \quad \sum_{ij \in E'} c'_{ij} X_{ij} + \sum_{i \in V'} p'_i (1 - y_i) \tag{4}$$

$$\text{subject to} \quad \sum_{ij \in E'} X_{ij} = \sum_{i \in V'} y_i - 1 \tag{5}$$

$$\sum_{i,j \in S} X_{ij} \leq \sum_{i \in S \setminus \{k\}} y_i \qquad \forall S \subseteq V', |S| \geq 2, \forall k \in S \tag{6}$$

$$0 \leq X_{ij} \leq 1 \qquad \forall (i,j) \in E' \tag{7}$$

$$0 \leq y_i \leq 1 \qquad \forall i \in V' \tag{8}$$

$$y_i \in \{0,1\} \qquad \forall i \in V' \tag{9}$$

Constraints (5) and (6) describe the tree structure of the solution. Constraint (5) excludes the empty tree from the set of feasible solutions. However, the selection of any isolated customer vertex dominates this trivial solution. Constraints (6) are called *generalized subtour elimination* constraints. They have been studied also by Goemans [14], and Margot et al. [22]. The validity of this formulation follows from Edmonds' characterization of the spanning tree polytope (see, for example, [10]).

4.2. Rooted Tree Flow-Formulations

Directed tree formulations rely on a transformation of the PCST to the problem of finding a minimum subgraph in a related, directed graph as proposed by Fischetti [13]. We transform the reduced graph $G' = (V', E', c', p')$ that results from the application of preprocessing into the directed edge-weighted graph $G_{\text{SA}} = (V_{\text{SA}}, A_{\text{SA}}, c'')$.

Transformation into the Steiner Arborescence Problem The vertex set $V_{\text{SA}} = V' \cup \{r\}$ contains the vertices of the input graph G' and an artificial root vertex r. The arc set

A_{SA} contains two directed arcs (i,j) and (j,i) for each edge $(i,j) \in E'$ plus a set of arcs from the root r to the customer vertices $R_{\text{SA}} = \{i \in V' \mid p'_i > 0\}$. We define the cost vector c'' as follows:

$$c''_{ij} = \begin{cases} c'_{ij} - p'_j & \forall (i,j) \in A_{\text{SA}},\ i \neq r \\ -p'_j & \forall (r,j) \in A_{\text{SA}}\ . \end{cases}$$

An example of this transformation can be found in Figures 3 (a) and (b) on page 19.

A subgraph T_{SA} of G_{SA} that forms a directed tree rooted at r is called a *Steiner arborescence*. It is easy to see that such a subgraph corresponds to a solution of the PCST if r has degree 1 in G_{SA} (*feasible arborescence*). In particular, a feasible arborescence with minimal total edge cost corresponds to an optimal prize-collecting Steiner tree.

We model the problem of finding a minimum Steiner arborescence T_{SA} by means of an integer linear program. Therefore, we introduce a variable vector $x \in \{0,1\}^{|A_{\text{SA}}|}$ with the following interpretation:

$$x_{ij} = \begin{cases} 1 & (i,j) \in T_{\text{SA}} \\ 0 & \text{otherwise} \end{cases} \qquad \forall (i,j) \in A_{\text{SA}}\ .$$

The small letters x indicate arc variables in the directed model whereas capital letters X were used in Section 4.1 for edges in the undirected case. Furthermore, to indicate which of the vertices from $V_{\text{SA}} \setminus \{r\}$ belong to the solution, we use a variable vector $y \in \{0,1\}^{|V_{\text{SA}}|-1}$ already introduced in the undirected formulation.

Single-Commodity Flow Formulation This is one of the simplest ILP formulations of the problem. Segev [26] used a very similar formulation for the single vertex-weighted

Steiner tree problem, under the name *tree-type formulation*. A related formulation for the Steiner tree problem has been studied by several authors, see, e.g., [8].

This formulation describes the structure of a rooted arborescence by a flow using the variables f_{ij} for the amount of flow on arc (i,j) for all $(i,j) \in A_{\text{SA}}$. One unit of flow is sent from the root vertex r to every customer vertex in the solution tree and f_{ij} represents the sum of all such flows via arc (i,j). In the optimal solution the values of x_{ij} indicate directed paths from r to every selected vertex. The resulting ILP-formulation can be written as follows:

$$(SF) \qquad \min \quad \sum_{ij \in A_{\text{SA}}} c''_{ij} x_{ij} + \sum_{i \in V_{\text{SA}}} p'_i \tag{10}$$

$$\text{subject to} \quad \sum_{ji \in A_{\text{SA}}} x_{ji} = y_i \qquad \forall i \in V_{\text{SA}} \setminus \{r\} \tag{11}$$

$$\sum_{ji \in A_{\text{SA}}} f_{ji} - \sum_{ij \in A_{\text{SA}}} f_{ij} = y_i \qquad \forall i \in R_{\text{SA}} \tag{12}$$

$$\sum_{ji \in A_{\text{SA}}} f_{ji} - \sum_{ij \in A_{\text{SA}}} f_{ij} = 0 \qquad \forall i \in V_{\text{SA}} \setminus R_{\text{SA}}, i \neq r \tag{13}$$

$$0 \leq f_{ij} \leq (|V_{\text{SA}}| - 1) \cdot x_{ij} \qquad \forall (i,j) \in A_{\text{SA}} \tag{14}$$

$$\sum_{ri \in A_{\text{SA}}} x_{ri} = 1 \tag{15}$$

$$y_i, x_{ij} \in \{0,1\} \qquad \forall i \in V_{\text{SA}} \setminus \{r\}, \forall (i,j) \in A_{\text{SA}} \tag{16}$$

The constant term in the objective function is added such that (SF) yields the desired overall solution value (2). The so-called *in-degree* equation (11) guarantees that every selected vertex has exactly one predecessor on its path from the root. The classical flow preservation constraints are given by (12) and (13), where the former requires that every selected vertex receives one unit of flow to be consumed in this vertex. Constraints (14)

force the arcs which are used by any flow to be included in the final directed tree of paths to the vertices. They resemble the so-called "big M" constraints which are known to be computationally inefficient in the sense that the LP-solution value is likely to deviate considerably from the optimal ILP value. Finally, the so-called *root-degree* constraint (15) makes sure that the artificial root r is connected only to a single vertex which is crucial for the connectedness of the solution.

Multi-Commodity Flow Formulation A straightforward strengthening of the previous formulation is the so-called *multi-commodity flow formulation* of the problem. Here, we split the flow from (SF) into separate commodities for every selected customer vertex. In this way the flow f_{ij}^k of a single commodity k describes an arc (i,j) on the directed path from the root vertex r to a selected customer vertex $k \in R_{\text{SA}}$ with $y_k = 1$.

$$(MCF) \quad \min \quad \sum_{ij \in A_{\text{SA}}} c''_{ij} x_{ij} + \sum_{i \in V_{\text{SA}}} p'_i \tag{17}$$

$$\text{subject to} \quad \sum_{ji \in A_{\text{SA}}} x_{ji} = y_i \qquad \forall i \in V_{\text{SA}} \setminus \{r\} \tag{18}$$

$$\sum_{ji \in A_{\text{SA}}} f_{ji}^i - \sum_{ij \in A_{\text{SA}}} f_{ij}^i = y_i \qquad \forall i \in R_{\text{SA}} \tag{19}$$

$$\sum_{ji \in A_{\text{SA}}} f_{ji}^k - \sum_{ij \in A_{\text{SA}}} f_{ij}^k = 0 \qquad \forall i \in V_{\text{SA}} \setminus \{k, r\}, \forall k \in R_{\text{SA}} \tag{20}$$

$$0 \leq f_{ij}^k \leq x_{ij} \qquad \forall (i,j) \in A_{\text{SA}}, \forall k \in R_{\text{SA}} \tag{21}$$

$$\sum_{ri \in A_{\text{SA}}} x_{ri} = 1 \tag{22}$$

$$y_i, x_{ij} \in \{0,1\} \qquad \forall i \in V_{\text{SA}} \setminus \{r\}, \forall (i,j) \in A_{\text{SA}} \tag{23}$$

The meaning of the constraints is almost equivalent to the (SF) formulation. Of course the flow preservation constraints are extended to hold for every single commodity in

(19) and (20). The former selects commodity i as the only possibility to deliver any flow into vertex i. Conditions (21) force an arc to be included in the solution tree as soon as the flow of any commodity traverses through. A multi-commodity flow formulation is well known for the standard Steiner tree problem (see e.g. [24]) and it was applied to the SPWST by Segev [26].

It is not surprising that (MCF) strictly dominates the (SF) formulation. This means that every solution of the LP-relaxation of (MCF) can be mapped into an equivalent LP-solution of (SF) by simply merging the commodities on every arc into a single flow. Figure 3 (c) and (d) on page 19 shows an example where $c(LP_{MCF}) > c(LP_{SF})$ holds, and so the domination relation between the two formulations is strict.

5. Cut Formulation

A different ILP-formulation (introduced in [13] for the NWST) concentrates on the connectedness of the solution. Therefore, *cuts* are introduced with the fairly simple condition that for every selected vertex which is separated from r by a cut there must be an arc crossing this cut.

For convenience we introduce the following notation: A set of vertices $S \subset V_{\text{SA}}$ and its complement $\overline{S} = V_{\text{SA}} \setminus S$ induce two directed cuts: $\delta^+(S) = \{(i,j) \mid i \in S, j \in \overline{S}\}$ and $\delta^-(S) = \{(i,j) \mid i \in \overline{S}, j \in S\}$. We also write $x(A) = \sum_{ij \in A} x_{ij}$ for any subset of arcs $A \subset A_{\text{SA}}$. The corresponding ILP model then reads as follows:

$$(CUT) \qquad \min \quad \sum_{ij \in A_{\text{SA}}} c''_{ij} x_{ij} + \sum_{i \in V_{\text{SA}}} p'_i \tag{24}$$

$$\text{subject to} \quad \sum_{ji \in A_{\text{SA}}} x_{ji} = y_i \qquad \forall i \in V_{\text{SA}} \setminus \{r\} \tag{25}$$

$$x(\delta^-(S)) \geq y_k \qquad k \in S, r \notin S, \forall S \subset V_{\text{SA}} \tag{26}$$

$$\sum_{ri \in A_{\text{SA}}} x_{ri} = 1 \tag{27}$$

$$x_{ij}, y_i \in \{0,1\} \qquad \forall (i,j) \in A_{\text{SA}}, \forall i \in V_{\text{SA}} \setminus \{r\} \tag{28}$$

The cut constraints (26) are also called *connectivity inequalities*. They guarantee that for each vertex v in the solution, there must be a directed path from r to v. Note that disconnectivity would imply the existence of a cut S separating r and v which would clearly violate the corresponding cut constraint.

As already observed in [13], the connectivity inequalities (26) can be put in an LP equivalent form by adding together $-x(\delta^-(S)) \leq -y_k$ and the in-degree equations $\sum_{ji \in A_{\text{SA}}} x_{ji} = y_i$ for all $i \in S$, to produce the generalized subtour elimination constraint $\sum_{i,j \in S} x_{ij} \leq \sum_{i \in S \setminus \{k\}} y_i$, the directed counterpart of (6). Chopra and Rao [6] have shown for the Steiner tree problem that directed GSECs dominate directed counterparts of several other facet defining inequalities of the undirected (GSEC) formulation. This is also the reason why the directed formulation is preferable in practice.

The following theorem shows the equivalence of the LP-relaxations of (CUT) and (MCF). Note that this fact has already been observed in a related setting for the classical Steiner tree problem (see for example [8]).

Theorem 1. *The polytopes of the LP-relaxations for (MCF) and (CUT) are identical.*

Proof. Let x_{ij} be feasible for (MCF) and assume that there exist S and k violating (26) in (CUT), i.e., $x(\delta^-(S)) < y_k$. Considering (19) for the same vertex k, it follows that there is a flow of value y_k from r to k in the directed network defined by the arc capacities x_{ij}. The classical max-flow min-cut theorem implies that every cut separating r and k must have a cut value at least y_k in contradiction to the assumption.

Let x_{ij} be feasible for (CUT). We want to construct a corresponding feasible multi-commodity flow for (MCF). If $y_k = 0$ we simply set $f_{ij}^k = 0$ for all $(i,j) \in A_{\text{SA}}$. For $y_k > 0$ consider the network with source r and sink k', where k' is connected only to k by an arc (k,k') with capacity y_k and capacities x_{ij} for all other arcs $(i,j) \in A_{\text{SA}}$. The maximum flow in this network delivers the flow of commodity k. Its flow value f^k is exactly y_k as required in (19). If this maximum flow f^k were smaller than y_k, then by the max-flow min-cut theorem there must exist a minimal cut between r and k with capacity less than y_k thus violating (26). Exceeding the flow value of y_k is prevented by the introduction of the artificial vertex k' which cannot receive a larger inflow. The feasibility of all other constraints in (MCF) is obvious. □

5.1. Asymmetry Constraints

In order to create a bijection between arborescence and PCST solutions, we introduce the so-called *asymmetry constraints*:

$$y_i \leq x_{rj},\ \forall i < j,\ i \in R \tag{29}$$

These inequalities assure that for each PCST solution the customer vertex adjacent to root is the one with the smallest index. Figure 2 illustrates an example. Computational

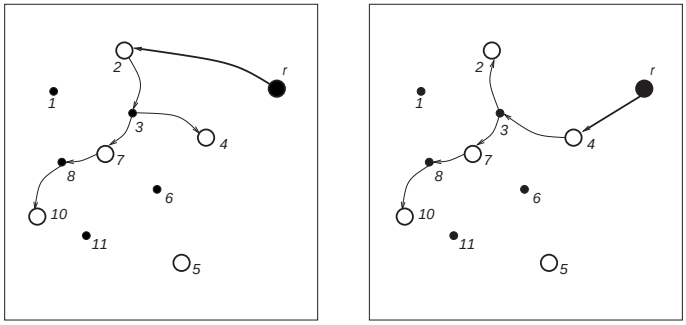


Fig. 2. Two feasible Steiner arborescences representing the same PCST solution. Using the asymmetry inequalities only the solution on the left-hand side is considered as feasible.

results have shown that they significantly reduce the computation time, because they exclude many unnecessary solutions.

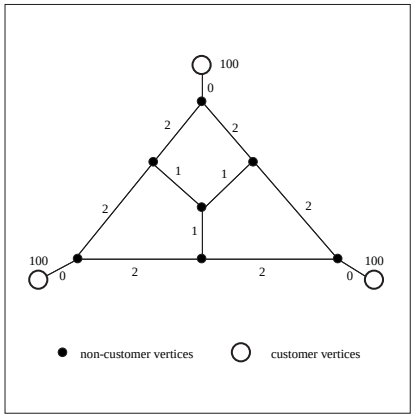
5.2. Strengthening the Formulation

Each feasible solution of the Steiner arborescence problem can be seen as a set of flows sending one unit from the root to all customer vertices j with $y_j = 1$.

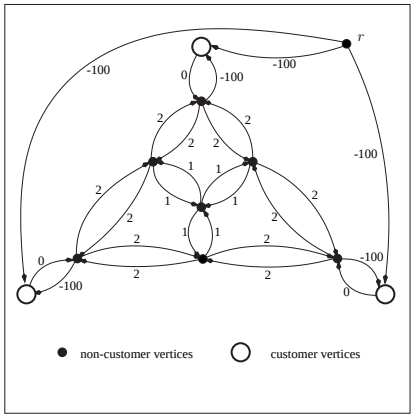
Considering the tree structure of the solution it is obvious that in every non-customer vertex, which is not a branching vertex in the Steiner arborescence, indegree and outdegree must be equal, whereas in a branching non-customer vertex indegree is always less than outgoing degree. Thus, we have:

$$\sum_{ji \in A_{\mathrm{SA}}} x_{ji} \leq \sum_{ij \in A_{\mathrm{SA}}} x_{ij} \ , \quad \forall i \notin R, \quad i \neq r \ . \tag{30}$$

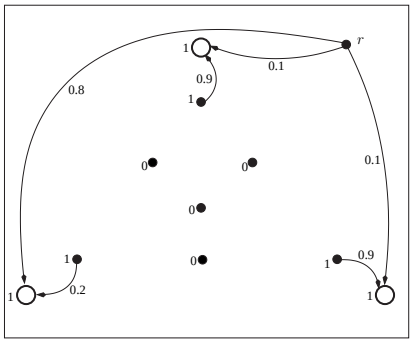
These so-called *flow-balance constraints* were introduced by Koch and Martin in [20] for the Steiner tree problem. They indeed represent a strengthening of the LP-relaxation of (25)-(29), as can be shown by an example in Figures 3 (d) and (e). For the classical Steiner tree problem, an analogous example can be found in [24].



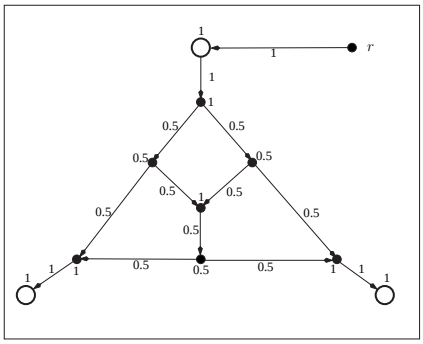
(a)



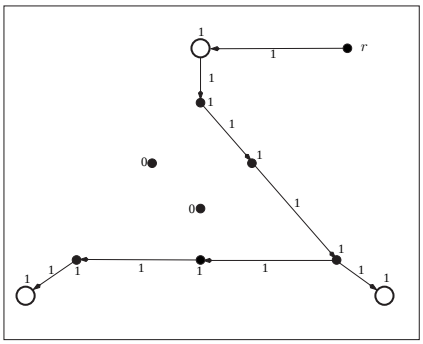
(b)



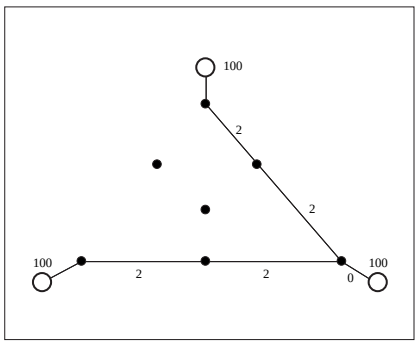
(c)



(d)



(e)



(f)

Fig. 3. (a) an input graph G; (b) after transformation into the Steiner arborescence problem; (c) solution of (SF) LP-relaxation, $c(LP_{SF}) = 0$. LP-values of x and y variables are shown; (d) solution of (MCF) LP-relaxation, $c(LP_{MCF}) = c(LP_{CUT}) = 7.5$; (e) solution of (CUT) LP-relaxation augmented with flow-balance constraints has cost 8 and corresponds to the optimal solution (f).

6. Branch-and-Cut Algorithm

To solve the proposed ILP formulation we use a branch-and-cut algorithm: At each node of the branch-and-bound tree we solve the LP-relaxation (CUT), obtained by replacing the integrality requirements (28) by the simple bounds: $0 \leq y_i \leq 1, \forall i \in V_{\text{SA}} \setminus \{r\}$ and $0 \leq x_{ij} \leq 1, \forall (i,j) \in A_{\text{SA}}$. For solving the LP-relaxations and as a generic implementation of the branch-and-cut approach, we used the commercial packages ILOG CPLEX and ILOG Concert Technology (version 8.1).

6.1. Initialization

There are exponentially many constraints of type (26), so we do not insert them at the beginning but rather *separate* them during the optimization process using the separation procedure described below.

At the root node of the branch-and-bound tree, we start with in-degree, root-degree, flow-balance and asymmetry constraints. Furthermore, we add the following group of inequalities:

$$x_{ij} + x_{ji} \leq y_i, \quad \forall i \in V_{\text{SA}} \setminus \{r\}, \quad (i,j) \in A_{\text{SA}} \tag{31}$$

These constraints express the trivial fact that every arc adjacent to a vertex in the solution tree can be oriented only in one way. They are also a special case of the connectivity constraints written in their equivalent GSEC form, a directed counterpart of (6), with $S := \{i,j\}$. Although the LP may become large by adding all of these inequalities at once they offer a tremendous speedup since they do not have to be separated implicitly during the branch-and-cut algorithm. Further details are discussed in Section 7.

6.2. Separation

During the separation phase which is applied at each node of the branch-and-bound tree, we add constraints of type (26) that are violated by the current solution of the LP-relaxation. Usually, this model is less dense than the equivalent directed (GSEC) model, so it may be computationally preferable within the branch-and-cut implementation.

These violated cut constraints can be found in polynomial time using a maximum flow algorithm on the *support graph* with arc-capacities given by the current solution. For finding the maximum flow in a directed graph, we used an adaptation of Goldberg's maximum flow algorithm [4][2].

The outline of the separation procedure is given in Algorithm 1. Given a support graph $G_s = (V_{\text{SA}}, A_{\text{SA}}, x)$, we search for violated inequalities by calculating the maximum flow for all (r, i) pairs of vertices, $i \in R_{\text{SA}}, y_i > 0$. The maximum flow algorithm $f = MaxFlow(G, x', r, i, S_r, S_i)$ returns the flow value f and two sets of vertices:

- Subset $S_r \subset V_{\text{SA}}$ contains root vertex r and induces a minimum cut closest to r, in other words, $x(\delta^+(S_r)) = f$;
- Subset $S_i \subset V_{\text{SA}}$ contains vertex i and induces a minimum cut closest to i, i.e., $x(\delta^-(S_i)) = f$.

If $f < y_i$, we insert the violated cut $x(\delta^+(S_r)) \geq y_i$ into the LP. We then follow the idea of the so-called *nested cuts* [20]: we iteratively add further violated constraints induced by the minimum (r, i)-cut in the support graph in which the capacities of all the arcs $(u, v) \in \delta^+(S_r)$ are set to one. This iterative process is done as long as the total number of the detected violated cuts is less than $MAXCUTS$ (100, in the default

[2] Available at `http://www.avglab.com/andrew/CATS/maxflow_solvers.htm`

Data : A support graph $G_s = (V_{\text{SA}}, A_{\text{SA}}, x)$.
Result : A set of violated inequalities incorporated into the current LP.

for $i \in R_{SA}, y_i > 0$ **do**
  $x' = x + EPS$;
  repeat
    $f = MaxFlow(G, x', r, i, S_r, S_i)$;
    Detect the cut $\delta^+(S_r)$ such that $x'(\delta^+(S_r)) = f, r \in S_r$;
    if $f < y_i$ **then**
      Insert the violated cut $x(\delta^+(S_r)) \geq y_i$ into the LP;
    end
    $x'_{ij} = 1, \forall (i,j) \in \delta^+(S_r)$;
    if $BACKCUTS$ **then**
      Detect the cut $\delta^-(S_i)$ such that $x'(\delta^-(S_i)) = f, i \in S_i$;
      if $S_i \neq \overline{S_r}$ **then**
        Insert the violated cut $x(\delta^-(S_i)) \geq y_i$ into the LP;
        $x'_{ij} = 1, \forall (i,j) \in \delta^-(S_i)$;
      end
    end
  until $f \geq y_i$ *or* $MAXCUTS$ *constraints added*;
end

Algorithm 1: Separation procedure.

implementation), or there are no more such cuts. By setting the capacities of the edges in a cut to one, we are able to increase the number of violated inequalities found within one cutting plane iteration. Note that the cuts are inserted only if they are violated by at least some ϵ (which was set to 10^{-4} in the default implementation).

Chopra et al. [5] proposed the so-called *back-cuts*, also used in [20], for the Steiner tree problem. To speed up the process of detecting more violated cuts within the same separation phase, we consider the reversal flow in order to find the cut "closest" to i, for some $i \in R, y_i > 0$. The advantage of Goldberg's implementation is that only one

maximum flow calculation is needed in order to find both sets $S_r, r \in S_r$ and $S_i, i \in S_i$ defining the minimum cut of value f. Note that back-cuts (controlled by $BACKCUTS$ parameter) are combined with nested cuts in our implementation.

Finally, we considered the possibility of adding the smallest cardinality cut by increasing all x_{ij} values by some value EPS. The smallest cardinality cuts may have a great influence on the density of the underlying LP, however the running time of the maximum flow calculations may also increase. Indeed, our computational results (cf. Section 7) confirm that for most of our instances setting EPS to a positive value increases the CPU time.

6.3. Primal Heuristic

The branch-and-cut framework of CPLEX calls the primal heuristic when the linear program in a node of the tree is solved and no more violated inequalities are found just before a branch is performed.

The basic idea of our primal heuristic is that we first fix a set S of vertices that will be contained in the heuristic solution. Then we apply the standard minimum spanning tree heuristic for the Steiner tree problem to the graph $G = (V, E, c)$ with terminal set S. Let T be the resulting tree. We solve the PCST on T optimally by the linear time algorithm also used in [19].

For choosing the set S of terminal vertices, we use the values of the y-variables in the LP-solution of the current node in the branch-and-cut tree. We tried several strategies: We used the y-values as probabilities for inserting a vertex into S or computed for each vertex the average of its LP-value in the fractional solution and the best known

feasible solution and used this value as the probability for choosing the vertex. But the following non-randomized method produced the best results: Choose all vertices v_i where the value of y_i in the current fractional solution is at least $1/2$.

Having chosen the vertices in S, we compute the *distance network* G_S for S where $G_S = (S, S \times S, d_S)$. The length d_S of an edge in G_S is the length of the shortest path connecting the two corresponding vertices in G. The length of a path is determined by the x-variables of the edges on the path. We assign to each edge (i, j) in the problem graph the cost $1 - \max\{x_{ij}, x_{ji}\}$ where x_{ij} is the value of the corresponding edge-variable in the fractional solution of the current branch-and-bound node. Thus, a path is short if its edges have high LP-values.

We compute a minimum spanning tree $T = (S, E_T)$ in G_S and define the set S' of vertices in G as the union of S and the set of all vertices on the shortest paths that correspond to edges in E_T. Let $G_H = (S', E_H, c)$ be the subgraph of G induced by the vertex set S'. In this graph, the cost of each edge is again the original cost in the problem instance.

This graph is connected and therefore we can compute a minimum spanning tree $T' = (S', E_{T'})$ for it. Finally, we use the linear time algorithm to solve the PCST optimally on the tree T'. The resulting graph is our heuristic solution. The computational results in Section 7 show that this heuristic can significantly improve the gap between the lower bound and the best known feasible solution for our most challenging problem instances.

7. Computational Results

We tested our new approach outlined in Section 6 extensively on the following groups of instances:

– Johnson et al. [17] tested their approximation algorithm on two sets of randomly generated instances. In the so-called P class, instances are unstructured and designed to have constant expected degree and profit to weight ratio. The K group comprises random geometric instances designed to have a structure somewhat similar to street maps. A detailed description of the generators for these instances can be found in [23]. In our tests, we considered a part of these instances with up to 400 vertices and 1 576 edges that have also been tested by Lucena and Resende [21] and Canuto et al. [3].

– Canuto et al. [3] generated a set of 80 test problems derived from the Steiner problem instances of the well-known OR-Library[3]. For each of the 40 problems from series C and D, two sets of instances were generated by assigning zero profits to non-terminal vertices and randomly generated profits in the interval $[1, maxprize]$ to terminal vertices. Here, $maxprize = 10$ for problems in set A, and $maxprize = 100$ for problems in set B. Instances of group C contain 500 vertices, and between 625 and 12 500 edges, while instances of group D contain 1 000 vertices and between 1 250 and 25 000 edges.

  Following this schema, we generated an additional set of 40 larger benchmark instances derived from series E of the Steiner problem instances in the OR-Library. The new instances contain 2 500 vertices and between 3 125 and 62 500 edges.

[3] OR-library: J. E. Beasley, http://mscmga.ms.ic.ac.uk/info.html

– Rosseti et al. [25] proposed three new sets of artificially generated and very difficult instances for the Steiner tree problem. We used the most difficult of them, the so-called *hypercubes*, to derive new test problems for the PCST.

  Graphs in this series for the Steiner tree problem are d-dimensional hypercubes with $d \in \{6, \ldots, 12\}$. For each value of d, the corresponding graph has 2^d vertices and $d \cdot 2^{d-1}$ edges. These graphs are bipartite and the terminals are defined as one set of the bipartition. The so-called *unperturbed* instances receive unit edge costs whereas edges of the so-called *perturbed* instances receive random integral costs distributed uniformly in the interval $[100, 110]$.

  We derived the PCST hypercubes by assigning zero profits to non-terminal vertices and an integer profit randomly chosen from a uniform distribution over the interval $[1, 2]$ and $[100, 220]$ for unperturbed and perturbed instances, respectively. Our naming convention is as proposed in [25], thus `hc`d`[u|p]` denotes an unperturbed (`u`) or perturbed (`p`) d-dimensional hypercube instance.

Instance sets `K`, `P`, `C` and `D` of instances are available at `http://www.research.att.com/~mgcr/data/index.html`. All other problem instances used in this paper are available in our online database for PCST instances and solutions at the following URL: `http://www.ads.tuwien.ac.at/pcst`.

For groups `C` and `D`, Tables 1 and 2 list the instance name, its number of edges $|E|$, the size of the graph after the reductions described in Section 3 ($|V'|$, $|E'|$) and the time spent on this preprocessing procedure (t_{prep} $[s]$). We compare our results against those recently obtained by Lucena and Resende [21] (denoted by LR). For their approach, we show the best obtained lower bounds (*L. Bound*) and the CPU times in seconds required

to prove optimality ($t\ [s]$). If the CPU time is not given, it means that the LR algorithm terminated because of excessive memory consumption. For our new ILP approach, we provide the following values: the provably optimal solution value (OPT), the total running time in seconds ($t\ [s]$) (not including preprocessing), the number of violated cuts found by our separation procedure (*#Cuts*), the number of violated Gomory fractional cuts automatically added by CPLEX (*#G. Cuts*) and the total running time of the same algorithm without preprocessing ($t_{noprep}\ [s]$).

On all but 8 instances of groups `C` and `D` lower bounds obtained by LR were equal to known upper bounds obtained by Canuto et al. [3]. However, on 16 instances from `C` and `D`, the LR algorithm did not prove the optimality. Improving upon their results, our new ILP approach solved all instances known from the literature to proven optimality. The optimal solution values, that were not guaranteed to be optimal before, are marked with an asterisk while new optimal values are given in bold face.

Comparing our running time data (achieved on a Pentium IV with 2.8 GHz, 2 GB RAM, SPECint2000=1 204) with the results of Lucena and Resende [21] (done on SGI Challenge Computer 28 196 MHz MIPS R10000 processors with 7.6 GB RAM, each run used a single processor), the widely used SPEC© performance evaluation (`www.spec.org`) does not provide a direct scaling factor. However, taking a comparison to the respective benchmark machines both for SPEC 95 and SPEC 2000 into account, we obtain a scaling factor of 17.2. On the other side, in [7] the SGI machine is assigned a factor of 114 and to our machine the factor 1 414, which gives 12.4 as a scaling factor. Thus, we can argue by a conservative estimate that dividing the LR running times by a factor of 20 gives a very reasonable basis of comparison to our data.

The running time comparison for those instances where LR running times are known, shows that our new approach is significantly faster (on average, by more than two orders of magnitude!).

Tables 1 and 2 document also that our new approach is able to solve all the instances to optimality within a very short time, even if preprocessing is turned off. However, preprocessing is important for larger instances, like those of the group E (see Table 3).

Both algorithms, LR and our new ILP approach, solved all P and K instances to optimality. The LR algorithm needed 53 (369) seconds on average for P (K) instances, while our new approach solved them in 0.2 (69.1) seconds to optimality. All the instances of K, P, C and D groups are solved in the root node of the branch-and-cut tree.

Table 3 presents the results of our new approach on the set E of benchmark instances. As before, the instance name, the number of edges of the original graph and the size of the instance after preprocessing as well as the preprocessing CPU time are listed. On these more challenging instances we compare the performance of the algorithm with and without preprocessing in more detail. The algorithm without preprocessing is terminated by setting the `cplexTimeLimit` parameter to 2 000 seconds. The results document that preprocessing can reduce the number of vertices and edges of the original graph by about 30%, respectively, 50%, on average. Furthermore, the results also show that preprocessing plays an important role when the size of the LPs increases. For 3 out of 40 instances the algorithm did not find the optimal solution within the given time limit if preprocessing is turned off. As before, the instances of this group are also solved without branching.

Table 1. Results obtained by Lucena and Resende (LR) and our new results, on the instances from Steiner series C. Running times in (LR) to be divided by 20 for comparison (cf. text above). Asterisk marks new certificates of optimality for known values. New optimal solution values are given in bold face.

	Orig.	Preprocessing			LR		ILP				
Instance	$\|E\|$	$\|V'\|$	$\|E'\|$	t_{prep} $[s]$	L. Bound	t $[s]$	OPT	t $[s]$	# Cuts	# G. Cuts	t_{noprep} $[s]$
C1-A	625	116	214	1.2	18	0.1	18	0.0	0	0	0.1
C1-B	625	125	226	1.2	85	1.7	85	0.1	6	5	0.2
C2-A	625	109	207	1.1	50	0.1	50	0.0	0	0	0.1
C2-B	625	111	209	1.1	141	1.0	141	0.0	0	0	0.2
C3-A	625	160	277	1.1	414	1.2	414	0.0	0	0	0.1
C3-B	625	185	304	1.3	737	26.1	737	0.1	2	1	0.6
C4-A	625	178	300	1.2	618	1.7	618	0.1	2	12	1.0
C4-B	625	218	341	1.3	1 063	287.4	1 063	0.1	4	0	0.5
C5-A	625	163	274	1.2	1 080	80.4	1 080	0.3	0	0	11.4
C5-B	625	199	314	1.7	1 528	3 487.1	1 528	0.2	0	0	1.6
C6-A	1 000	355	822	2.1	18	0.9	18	0.1	0	0	0.1
C6-B	1 000	356	823	2.1	55	57.5	55	0.4	12	11	0.5
C7-A	1 000	365	842	2.6	50	1.3	50	0.1	0	0	0.1
C7-B	1 000	365	842	2.5	102	4.7	102	0.1	4	4	0.1
C8-A	1 000	367	849	2.7	361	33.4	361	0.1	0	0	0.6
C8-B	1 000	369	850	3.0	500	215.0	500	0.2	4	1	0.6
C9-A	1 000	387	877	2.4	533	84.1	533	2.7	20	30	1.8
C9-B	1 000	389	879	2.8	694	1 912.6	694	1.2	24	4	0.7
C10-A	1 000	359	841	3.3	859	160.3	859	0.8	8	0	2.3
C10-B	1 000	323	798	3.4	1 069	3 502.3	1 069	0.4	8	0	3.7
C11-A	2 500	489	2 143	9.4	18	3.4	18	0.2	0	0	0.2
C11-B	2 500	489	2 143	9.5	32	68.5	32	3.5	48	18	2.2
C12-A	2 500	484	2 186	6.8	38	37.0	38	0.2	4	1	0.2
C12-B	2 500	484	2 186	6.8	46	126.7	46	0.3	6	3	0.4
C13-A	2 500	472	2 113	9.8	236	332.5	236	0.6	2	1	2.6
C13-B	2 500	471	2 112	9.8	258	3 092.7	258	2.1	30	23	5.9
C14-A	2 500	466	2 081	7.5	293	1 749.8	293	0.4	2	0	0.5
C14-B	2 500	459	2 048	7.5	318	1 142.3	318	0.4	4	0	0.4
C15-A	2 500	406	1 871	6.5	501	54 223.3	501	1.1	4	2	9.7
C15-B	2 500	370	1 753	6.0	551	—	*551	0.5	2	1	5.6
C16-A	12 500	500	4 740	2.4	11	204.5	11	1.6	10	14	2.3
C16-B	12 500	500	4 740	2.4	11	205.1	11	1.3	6	21	2.3
C17-A	12 500	498	4 694	2.4	18	250.3	18	1.2	6	13	3.0
C17-B	12 500	498	4 694	2.3	18	388.2	18	1.2	6	16	2.9
C18-A	12 500	469	4 569	2.6	111	20 031.8	111	1.5	4	15	18.0
C18-B	12 500	465	4 538	2.9	113	—	*113	18.3	95	6	38.9
C19-A	12 500	430	3 982	2.9	146	152 217.1	146	0.6	4	0	1.5
C19-B	12 500	416	3 867	2.8	146	18 999.6	146	0.5	4	0	2.4
C20-A	12 500	241	1 222	6.1	265	—	**266**	0.1	2	0	236.0
C20-B	12 500	133	563	5.0	267	—	*267	0.1	4	0	125.0
C-AVG	4 156.3	348.5	1 733.4	3.8	334.3	—	334.3	1.1	8.4	5.1	12.2

Table 2. Results obtained by Lucena and Resende (LR) and our new results, on the instances from Steiner series D. Running times in (LR) to be divided by 20 for comparison (cf. text above). Asterisk marks new certificates of optimality for known values. New optimal solution values are given in bold face.

	Orig.	Preprocessing			LR		ILP				
Instance	$\|E\|$	$\|V'\|$	$\|E'\|$	t_{prep} $[s]$	L. Bound	t $[s]$	OPT	t $[s]$	# Cuts	# G. Cuts	t_{noprep} $[s]$
D1-A	1 250	231	440	4.9	18	0.4	18	0.1	0	0	0.1
D1-B	1 250	233	443	4.9	106	5.5	106	0.1	8	15	0.7
D2-A	1 250	257	481	4.9	50	0.7	50	0.0	0	0	0.1
D2-B	1 250	264	488	4.9	218	2.2	218	0.0	0	0	0.3
D3-A	1 250	301	529	5.5	807	12.2	807	0.1	0	0	0.8
D3-B	1 250	372	606	6.3	1 509	331.5	1 509	0.3	2	0	0.9
D4-A	1 250	311	541	5.6	1 203	51.5	1 203	0.5	2	0	3.6
D4-B	1 250	387	621	7.2	1 881	1 551.3	1 881	0.8	12	0	2.8
D5-A	1 250	348	588	7.6	2 157	597.5	2 157	0.9	6	0	115.5
D5-B	1 250	411	649	11.5	3 135	—	*3 135	1.3	6	0	5.0
D6-A	2 000	740	1 707	14.4	18	2.6	18	0.1	0	0	0.2
D6-B	2 000	741	1 708	14.7	67	225.8	67	1.5	14	12	1.9
D7-A	2 000	734	1 705	11.3	50	4.3	50	0.1	0	0	0.2
D7-B	2 000	736	1 707	11.4	103	154.1	103	0.3	2	0	0.3
D8-A	2 000	764	1 738	11.7	755	170.7	755	5.8	20	8	8.2
D8-B	2 000	778	1 757	12.3	1 036	3 267.9	1 036	0.7	2	1	3.5
D9-A	2 000	752	1 716	17.9	1 070	1 346.7	1 070	11.4	8	10	92.8
D9-B	2 000	761	1 724	20.9	1 420	25 052.5	1 420	3.7	50	0	4.0
D10-A	2 000	694	1 661	14.6	1 671	62 590.0	1 671	6.4	8	2	62.6
D10-B	2 000	629	1 586	18.5	2 079	—	*2 079	1.6	4	0	13.5
D11-A	5 000	986	4 658	27.7	18	33.9	18	1.2	2	0	0.4
D11-B	5 000	986	4 658	23.6	29	870.6	29	3.5	8	29	4.1
D12-A	5 000	991	4 639	23.1	42	281.7	42	1.8	10	10	1.9
D12-B	5 000	991	4 639	22.3	42	297.1	42	1.2	4	1	0.9
D13-A	5 000	966	4 572	27.7	445	24 689.7	445	3.5	16	3	23.4
D13-B	5 000	961	4 566	28.0	486	4 464.2	486	2.2	8	0	7.3
D14-A	5 000	946	4 500	35.5	602	—	*602	4.3	14	0	80.4
D14-B	5 000	931	4 469	37.2	665	—	*665	14.4	40	5	34.6
D15-A	5 000	832	4 175	47.1	1 040	—	**1 042**	11.3	12	1	18.9
D15-B	5 000	747	3 896	49.2	1 107	1691 918.5	**1 108**	2.6	6	0	55.8
D16-A	25 000	1 000	10 595	10.8	13	9 957.8	13	3.4	6	2	9.7
D16-B	25 000	1 000	10 595	10.8	13	6 129.7	13	4.5	8	3	7.4
D17-A	25 000	999	10 534	10.8	23	16 939.8	23	18.9	38	36	86.2
D17-B	25 000	999	10 534	10.7	23	13 742.4	23	17.2	44	29	23.3
D18-A	25 000	944	9 949	11.7	218	—	*218	25.5	52	23	60.4
D18-B	25 000	929	9 816	12.0	223	—	**223**	6.3	12	0	16.1
D19-A	25 000	897	9 532	12.4	306	—	**306**	20.3	38	28	62.1
D19-B	25 000	862	9 131	13.1	310	—	**310**	18.2	40	78	19.8
D20-A	25 000	488	2 511	37.3	529	—	**536**	0.4	0	0	18.5
D20-B	25 000	307	1 383	32.9	530	—	**537**	0.2	2	0	18.3
D-AVG	8 312.5	705.2	3 793.7	17.4	650.4	—	650.9	4.9	12.6	7.4	21.7

In our default implementation we used nested cuts and back-cuts. The parameter EPS was set to 0.0, which means that we refrained from the calculation of smallest cardinality cuts. Our computational experiments have shown that the usage of back-cuts is crucial for our implementation. By omitting the back-cuts, some of the larger instances (even of groups C and D) could not be solved to proven optimality.

The role of parameter EPS within the separation is studied in Table 4 where we show CPU times in seconds taken on average over each group of instances. All the runs were limited to 1 000 seconds (except for group E, where we set the limit to 2 000). As the first two columns in Table 4 document, the usage of minimum cardinality cuts within the separation plays the most important role for solving the instances in the K group. In contrary, for solving the instances in the C, D and E groups, computing the smallest cardinality cuts seems to be too expensive, i.e., there is a trade-off between the time needed to solve the maximum-flow algorithm and the time for solving a single LP-relaxation.

Table 3. Results obtained on the instances derived from Steiner series E.

	Orig.	Preprocessing				With preprocessing			Without preprocessing		
Instance	$\|E\|$	$\|V'\|$	$\|E'\|$	t_{prep} $[s]$	OPT	t $[s]$	# Cuts	# G. Cuts	t $[s]$	# Cuts	# G. Cuts
E01-A	3 125	651	1 246	21.5	13	0.1	0	0	0.2	0	0
E01-B	3 125	655	1 250	21.8	109	0.4	12	38	1.9	12	8
E02-A	3 125	694	1 304	20.7	30	0.1	0	0	0.3	0	0
E02-B	3 125	697	1 307	20.7	170	0.3	2	0	2.1	6	5
E03-A	3 125	813	1 414	29.4	2 231	5.6	6	37	63.7	26	38
E03-B	3 125	962	1 572	30.9	3 806	3.6	6	0	14.3	116	1
E04-A	3 125	829	1 425	24.9	3 151	23.9	12	2	237.1	8	4
E04-B	3 125	980	1 588	26.2	4 888	7.3	8	0	17.7	6	0
E05-A	3 125	893	1 502	36.9	5 657	81.6	12	6	438.4	16	0
E05-B	3 125	1 029	1 644	45.0	7 998	16.0	6	0	794.0	132	2
E06-A	5 000	1 821	4 283	37.9	19	0.3	0	0	0.4	0	0
E06-B	5 000	1 821	4 283	37.6	70	1.6	6	14	3.5	4	12
E07-A	5 000	1 863	4 339	39.3	40	0.3	0	0	0.4	0	0
E07-B	5 000	1 865	4 341	39.4	136	2.4	6	8	3.4	4	8
E08-A	5 000	1 902	4 379	40.1	1 878	192.6	69	18	397.9	20	14
E08-B	5 000	1 911	4 387	50.4	2 555	6.8	8	0	15.3	14	2
E09-A	5 000	1 909	4 388	50.5	2 787	55.5	8	0	700.5	22	9
E09-B	5 000	1 918	4 397	54.7	3 541	13.5	4	2	25.0	8	3
E10-A	5 000	1 716	4 181	60.6	4 586	159.8	22	0	608.1	10	0
E10-B	5 000	1 594	4 045	84.6	5 502	51.9	34	1	381.2	78	2
E11-A	12 500	2 491	12 063	145.1	21	2.3	0	0	5.9	4	8
E11-B	12 500	2 491	12 063	146.2	34	7.1	2	4	6.3	2	0
E12-A	12 500	2 490	12 090	82.5	49	6.0	6	14	2.3	0	0
E12-B	12 500	2 490	12 090	85.5	67	13.5	14	2	16.0	16	20
E13-A	12 500	2 430	11 949	148.2	1 169	85.5	34	2	95.7	24	29
E13-B	12 500	2 407	11 915	146.7	1 269	37.0	18	11	25.9	8	0
E14-A	12 500	2 366	11 872	144.2	1 579	1 903.8	440	12	94.6	14	3
E14-B	12 500	2 311	11 737	145.7	1 716	60.9	36	1	357.4	269	4
E15-A	12 500	2 044	10 845	207.8	2 610	149.2	34	1	430.8	111	1
E15-B	12 500	1 864	10 264	234.3	2 767	1 375.3	652	1	243.4	42	9
E16-A	62 500	2 500	29 332	82.2	15	39.3	8	17	56.5	8	9
E16-B	62 500	2 500	29 332	81.9	15	31.6	8	22	53.7	10	16
E17-A	62 500	2 500	29 090	81.6	25	32.8	4	5	53.0	6	19
E17-B	62 500	2 500	29 090	81.8	25	32.0	10	19	60.3	10	13
E18-A	62 500	2 378	28 454	85.7	555	629.4	162	0	—	—	—
E18-B	62 500	2 347	28 269	86.9	564	1 746.1	388	1	2 025.0	192	2
E19-A	62 500	2 156	25 011	92.2	747	51.4	14	3	830.2	56	24
E19-B	62 500	2 085	23 641	94.0	758	19.1	8	3	594.2	42	2
E20-A	62 500	1 525	12 770	107.3	1 331	4.3	0	0	—	—	—
E20-B	62 500	861	3 881	231.5	1 342	1.2	2	0	—	—	—
E-AVG	20 781.3	1 781.5	10 325.8	82.1	1 645.6	171.3	51.5	6.1	—	—	—

Table 4 also documents the crucial role of using inequalities (31) within the initialization procedure. This inequality says that each edge can only be used in one direction in any feasible solution (see Section 6.1). The last two columns represent results obtained by omitting inequalities (31) from the initialization.

It can be observed that the number of directed subtours of size two drastically increases with the input size such that the algorithm is not able to solve many of the instances within a given time limit. This negative effect can not be compensated by the – sometimes considerable – improvement of running time brought about by the separation with smallest cardinality cuts (in particular for K and P groups). The difficulties usually arise when a customer vertex i is connected to a non-customer vertex j by an edge with cost $c(i,j) < p_i$. In this case, the initial LP-solution contains a directed arborescence rooted at j (note that $y_j = 0$, i.e., the in-degree of j is zero), with an arc (j,i) of negative cost. By adding $x_{ij} \leq y_i, \forall i \in V_{\text{SA}} \setminus \{r\}$ inequalities, instead of (31), solutions of the initial LP contain subtours of size two on such pairs (i,j) of vertices.

We also applied our program to the hypercube instances mentioned earlier with dimensions from 6 to 12 generated from the graphs used in [25]. The running time of the program was limited to 1 800 seconds. For each instance, we provided an initial feasible solution computed by the heuristic algorithm described in [18]. As primal heuristic for the branch-and-bound algorithm, we used the simple minimum spanning tree heuristic described in Section 6.3 that uses the fractional solution at the current branch-and-bound node to compute a feasible solution. Note that our preprocessing method described in Section 3 did not manage to reduce these instances at all.

Table 4. Comparison of average CPU times over all instances of a group for separation with or without smallest cardinality cuts ($EPS = 10^{-4}$ or $EPS = 0$, respectively) and for initialization with or without (31) constraints.

Group	Init. with (31) ineq.		Init. without (31) ineq.	
	$EPS = 0$	$EPS = 10^{-4}$	$EPS = 0$	$EPS = 10^{-4}$
K	69.1	10.0	100.5	4.7
P	0.2	0.3	25.5	2.9
C	1.1	0.8	—	22.6
D	4.9	10.0	—	—
E	171.3	279.3	—	—

Table 5 shows the results for our experiments with the hypercube instances. The three instances hc6u, hc6p and hc7u could be solved to optimality within the time bound. For the instances hc7p to hc10p, the gap between the best feasible solution found and the best lower bound is between 1.44 and 10.70 percent. For the hypercubes with dimensions 11 and 12, no lower bound could be found because the initial linear program could not be solved before the end of the time limit.

The number of cuts separated grows with the size of the problem but than goes down again when the linear programs become so big that the number of simplex iterations in the time limit drops. Using our primal heuristic produced different results than the version of the program without the heuristic only for two instances: hc8p and hc9p.

For instance hc8p, the version with the primal heuristic performs slightly worse than the version without. The lower bound was the same in both cases but the best feasible solution found was slightly worse than in the version without the primal heuris-

Table 5. Computational results for the hypercube instances with a time limit of 1 800 seconds.

Instance Name	Number of G. cuts	Number of cuts	Number of B&B Nodes	Best feasible	Best bound	Gap (%)	Gap (%) w/o heuristic
hc6p	4	1547	125	3908	3908	0.00	0.00
hc6u	3	2	0	36	36	0.00	0.00
hc7p	0	8003	289	7739	7628.99	1.44	1.44
hc7u	0	2135	12	72	72	0.00	0.00
hc8p	0	5122	44	15274	14995.9	**2.15**	**1.85**
hc8u	87	486	0	150	141	6.38	6.38
hc9p	0	1720	10	32401	29661.3	**3.96**	**9.24**
hc9u	86	451	0	301	279.356	7.75	7.75
hc10p	41	104	0	65186	58884.7	10.70	10.70
hc10u	96	174	0	594	552.655	7.48	7.48

tic. For this instance, the built-in heuristic of CPLEX found a better solution than our heuristic.

For the instance hc9p, the gap between the best known feasible solution and the lower bound is more than twice as large without our primal heuristic. The lower bound was the same in both runs but our heuristic found a solution that CPLEX alone could not find.

Our heuristic can only improve the result if the branch-and-bound tree has more than one node, otherwise it is never called. It follows that it is not useful for easy instances (where the problem can be solved in the root node) or very large instances (where CPLEX cannot finish solving the linear program of the root node of the tree before the time limit). So we conclude that the heuristic is of limited use and should only be tried for instances where CPLEX without the heuristic has to branch.

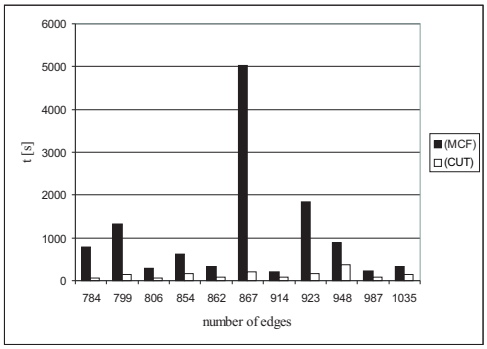


Fig. 4. CPU times of (CUT) and (MCF) formulations for 11 `K400` instances. The instances are sorted according to their number of edges after preprocessing.

Finally, Figure 4 shows that the multi-commodity flow formulation is very ineffective in practice. The running times for the (MCF) formulation vary widely and consistently exceed the running time for the (CUT) model by a huge margin.

8. Conclusions

The prize-collecting Steiner tree problem (PCST) formalizes in an intuitive way the planning problem encountered in the design of utility networks such as gas and district heating. Selecting the most profitable customers and connecting them by a least-cost network immediately leads to the problem of computing a Steiner tree, where the terminals are not fixed but can be chosen arbitrarily from a given set of vertices each one contributing a certain profit.

The aim of this paper is the construction of an algorithmic framework to solve large and difficult instances of PCST to optimality within reasonable running time. The method of choice is a branch-and-cut approach based on an ILP formulation depending

on connectivity inequalities which can be written as cuts between an artificial root and every selected customer vertex.

While the choice of the ILP model is essential for the success of our method, it should also be pointed out that solving the basic ILP model by a default algorithm is by no means sufficient to reach reasonable results. Indeed, our experiments show that a satisfying performance can be achieved only by appropriate initialization and strengthening of the original ILP formulation and in particular by a careful analysis of the separation procedure.

Combining all these efforts, we manage to solve to optimality (even without the usual preprocessing) all instances from the literature in a few seconds thereby deriving new optimal solution values and new certificates of optimality for a number of problems previously attacked.

For a number of new large instances constructed from Steiner tree instances, we also derive optimal solutions within reasonable running time. For these instances with more than 60 000 edges, our advanced preprocessing procedure proves to be an indispensable tool for still finding the optimum without branching.

The so-called hypercube instances are the final performance test for our algorithm. The built-in difficulty of these instances for the standard Steiner tree problem carries over in a natural way to PCST. For these cases, we add a primal heuristic to our framework to improve the upper bound in each node of the branch-and-cut tree. This heuristic can improve the best feasible solutions found dramatically but can in other cases make the solution slightly worse.

Acknowledgments

The authors thank Andreas Moser and Philipp Neuner for their help in implementing parts of the algorithmic framework.

References

1. J. E. Beasley. An SST-based algorithm for the Steiner problem in graphs. *Networks*, 19:1–16, 1989.
2. D. Bienstock, M. X. Goemans, D. Simchi-Levi, and D. Williamson. A note on the prize-collecting traveling salesman problem. *Mathematical Programming*, 59:413–420, 1993.
3. 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*, 38:50–58, 2001.
4. B. V. Cherkassky and A. V. Goldberg. On implementing push-relabel method for the maximum flow problem. *Algorithmica*, 19:390–410, 1997.
5. S. Chopra, E. Gorres, and M. R. Rao. Solving a Steiner tree problem on a graph using a branch and cut. *ORSA Journal on Computing*, 4:320–335, 1992.
6. S. Chopra and M. R. Rao. The Steiner tree problem I: Formulations, compositions and extension of facets. *Mathematical Programming*, 64:209–229, 1994.
7. J. J. Dongarra. Performance of various computers using standard linear equations software (linpack benchmark report). Technical Report CS-89-85, University of Tennessee, 2004.
8. C. Duin. *Steiner's Problem in Graphs*. PhD thesis, University of Amsterdam, 1993.
9. C. W. Duin and A. Volgenant. Some generalizations of the Steiner problem in graphs. *Networks*, 17(2):353–364, 1987.
10. J. Edmonds. Submodular functions, matroids and certain polyhedra. In R. Guy, H. Hanani, N. Sauer, and J. Schönheim, editors, *Combinatorial Structures and Their Application*, pages 69–87. Gordon and Breach, 1970.
11. S. Engevall, M. Göthe-Lundgren, and P. Värbrand. A strong lower bound for the node weighted Steiner tree problem. *Networks*, 31(1):11–17, 1998.
12. P. Feofiloff, C.G. Fernandes, C.E. Ferreira, and J.C. Pina. Primal-dual approximation algorithms for the prize-collecting Steiner tree problem. 2003. submitted.

13. M. Fischetti. Facets of two Steiner arborescence polyhedra. *Mathematical Programming*, 51:401–419, 1991.
14. M. X. Goemans. The Steiner tree polytope and related polyhedra. *Mathematical Programming*, 63:157–182, 1994.
15. M. X. Goemans and D. P. Williamson. The primal-dual method for approximation algorithms and its application to network design problems. In D. S. Hochbaum, editor, *Approximation algorithms for NP-hard problems*, pages 144–191. P. W. S. Publishing Co., 1996.
16. J. Hackner. *Energiewirtschaftlich optimale Ausbauplanung kommunaler Fernwärmesysteme*. PhD thesis, Vienna University of Technology, Austria, 2004.
17. D. S. Johnson, M. Minkoff, and S. Phillips. The prize-collecting Steiner tree problem: Theory and practice. In *Proceedings of 11th ACM-SIAM Symposium on Discrete Algorithms*, pages 760–769, San Francisco, CA, 2000.
18. G.W. Klau, I. Ljubić, A. Moser, P. Mutzel, P. Neuner, U. Pferschy, and R. Weiskircher. Combining a memetic algorithm with integer programming to solve the prize-collecting Steiner tree problem. In K. Deb, editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2004)*, volume 3102 of *LNCS*, pages 1304–1315. Springer-Verlag, 2004.
19. G.W. Klau, I. Ljubić, P. Mutzel, U. Pferschy, and R. Weiskircher. The fractional prize-collecting Steiner tree problem on trees. In G. Di Battista and U. Zwick, editors, *ESA 2003*, volume 2832 of *LNCS*, pages 691–702. Springer-Verlag, 2003.
20. T. Koch and A. Martin. Solving Steiner tree problems in graphs to optimality. *Networks*, 32:207–232, 1998.
21. A. Lucena and M. G. C. Resende. Strong lower bounds for the prize-collecting Steiner problem in graphs. *Discrete Applied Mathematics*, 141:277–294, 2004.
22. F. Margot, A. Prodon, and Th. M. Liebling. Tree polyhedron on 2-tree. *Mathematical Programming*, 63:183–192, 1994.
23. M. Minkoff. The prize-collecting Steiner tree problem. Master's thesis, MIT, May, 2000.
24. T. Polzin and S. V. Daneshmand. A comparison of Steiner tree relaxations. *Discrete Applied Mathematics*, 112:241–261, 2001.
25. I. Rosseti, M. Poggi de Arago, 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*, pages 557–561, 2001.

26. A. Segev. The node-weighted Steiner tree problem. *Networks*, 17:1–17, 1987.