An Algorithmic Framework for the Exact Solution of the Prize-Collecting Steiner Tree Problem*

Ivana Ljubić[1], René Weiskircher[1], Ulrich Pferschy[2], Gunnar W. Klau[1],

Petra Mutzel[1], and Matteo Fischetti[3]

[1] Vienna University of Technology, Favoritenstr. 9-11, A-1040 Vienna, Austria
[2] University of Graz, Universitätsstr. 15, A-8010 Graz, Austria
[3] DEI, University of Padova, via Gradenigo 6/a, I-35131 Padova, Italy

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 and implementation of a branch-and-cut algorithm based on a directed graph model where we combine several state-of-the-art methods previously used for the Steiner tree problem. Our method outperforms the previously published results on the standard benchmark set of problems.

We can solve all benchmark instances from the literature to optimality, including some of them for which the optimum was not known. Compared to a recent algorithm by Lucena and Resende, our new method is faster by more than two orders of magnitude. We also introduce a new class of more challenging instances and present computational results for them. Finally, for a set of large-scale real-world instances arising in the design of fiber optic networks, we also obtain optimal solution values.

Keywords: Branch-and-Cut – Steiner Arborescence – Prize Collecting – Network Design

1 Introduction

The recent deregulation of public utilities such as electricity and gas in Europe 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

* This work has been partly supported by the RTN ADONET, 504438, by the Doctoral Scholarship Program of the Austrian Academy of Sciences (DOC) and by CNR and MIUR, Italy. A preliminary version of this paper appeared as [21].

companies in this field [17]. Another area that has recently gained in importance is the extension of fiber optic networks to cover the "last mile" reaching private households.

In a typical planning scenario the input is a set of potential customers with known or estimated demands (represented by discounted future profits), and a potential network for laying the pipes or cables (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 putting the connections into the ground.

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 existing infrastructure. 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 or cable on the corresponding street segment.

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

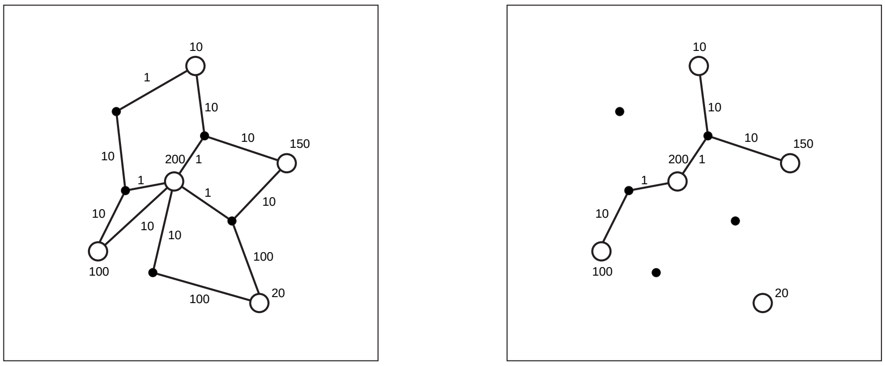
Definition 1 (Prize-Collecting Steiner Tree Problem, PCST). *Let* $G = (V, E, c, p)$ *be an undirected vertex- and edge-weighted graph as defined above. The* Linear Prize-Collecting Steiner Tree problem *(PCST) consists of finding a connected subgraph* $T = (V_T, E_T)$ *of* G*,* $V_T \subseteq V$*,*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



(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) [18]. 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:

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

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, 22, 5]).

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. This approach can also be used to guarantee the connection to existing infrastructure by shrinking all the corresponding vertices into a single vertex that forms the source vertex for the rooted problem.

In the next section we give a short overview of previous work on PCST and some of its relatives. Some of the results we report have their roots in the study of the symmetric and asymmetric traveling salesman problem and of some of its "non-spanning" variants, and are addressed, e.g., in the recent book [16]. Preprocessing, which helps to significantly reduce the size of many instances, is treated in Section 3. In Section 4 we propose a transformation of the input graph into a rooted digraph. We also introduce the cut-based ILP model and describe how it can be solved in a branch-and-cut framework in Section 5. Extensive computational experiments on instances from the literature and on new instances are reported in Section 6. 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 [24] 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 of the remaining vertices are non-positive profit values, while non-negative weights of the edges reflect the costs incurred for 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. [4], 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. [18], 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. [5] proposed a multi-start local-search-based algorithm with perturbations; Klau et al. [19] developed a memetic algorithm with incorporated local improvement that used a previous version of the exact algorithm described in this paper as a subroutine.

2.1 Lower Bounds and Polyhedral Studies

In [24], Segev presented single- and multi-commodity flow formulations for SPWST. 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 [3] 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. 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 [22] presented a cutting plane algorithm for the PCST based on generalized subtour elimination constraints. Their algorithm contains basic reduction steps similar to those given by Duin and Volgenant [10], and was tested on two groups of benchmark instances proposed in [18, 5]. The proposed algorithm solved many of the considered instances to optimality, but not all of them (cf. Section 6).

3 Preprocessing

In this section, we briefly describe reduction techniques adopted from the work of Duin and Volgenant [10] for the NWST, which have been partially used also in [22]. 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

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

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 $l = 2$. In [25] Uchoa generalized the least cost test and the degree-l test for the PCST such that both tests can also be used for customer vertices. This is not implemented in the framework described here.

Minimum Adjacency Test This test is also known as $V \setminus K$ *reduction test* from [10]. 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 6. The space complexity of preprocessing does not exceed $O(|E|^2)$.

4 Cut Formulation

In this section we show the integer linear program formulation we use. It is defined on a directed graph model and uses connectivity inequalities corresponding to minimum weight cuts in the graph to guarantee connectivity of the solution.

4.1 Transformation into the Steiner Arborescence Problem

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

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).

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 variable vectors $x \in \{0,1\}^{|A_{\text{SA}}|}$ and $y \in \{0,1\}^{|V_{\text{SA}}|-1}$ with the following interpretation:

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

Our ILP-formulation 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. Analogous formulations were used by Wong [26] and Fischetti in [13] for the NWST. An undirected ILP-formulation using cuts was already introduced by Aneja [1] who modelled the collection of all cut constraints as a set covering problem and studied the resulting branch and cut approach.

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:

$$(CUT) \qquad \min \sum_{ij \in A_{\text{SA}}} c''_{ij} x_{ij} + \sum_{i \in V_{\text{SA}}} p'_i \tag{4}$$

$$\text{subject to} \quad \sum_{ji \in A_{\text{SA}}} x_{ji} = y_i \qquad \forall i \in V_{\text{SA}} \setminus \{r\} \tag{5}$$

$$x(\delta^-(S)) \geq y_k \qquad k \in S, r \notin S, \forall S \subset V_{\text{SA}} \tag{6}$$

$$\sum_{ri \in A_{\text{SA}}} x_{ri} = 1 \tag{7}$$

$$x_{ij}, y_i \in \{0,1\} \qquad \forall (i,j) \in A_{\text{SA}}, \forall i \in V_{\text{SA}} \setminus \{r\} \tag{8}$$

The cut constraints (6) 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 (6) can be put in an LP equivalent form by adding together $-x(\delta^-(S)) \leq -y_k$ and the in-degree equations $\sum_{ji \in A_{\mathrm{SA}}} x_{ji} = y_i$ for all $i \in S$, to produce the generalized subtour elimination constraint (GSEC):

$$\sum_{i,j \in S} x_{ij} \leq \sum_{i \in S \setminus \{k\}} y_i \ .$$

Note that the lower bounding procedure for PCST presented in [22] is based on undirected GSECs. However, Chopra and Rao [8] 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 GSEC (and also min-cut) formulation is preferable in practice.

4.2 Asymmetry Constraints

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

$$x_{rj} \leq 1 - y_i, \ \forall\, i < j, \ i \in R \tag{9}$$

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. Although we insert $\binom{R}{2}$ new constraints, computational results show that they significantly reduce the computation time, because they exclude separation of many cuts that correspond to symmetric solutions.

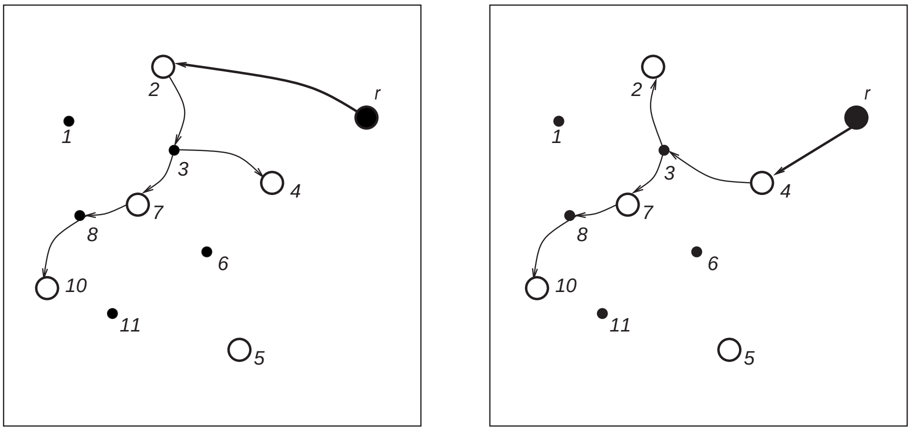


Fig. 2. Two feasible Steiner arborescences representing the same PCST solution. Using the asymmetry inequalities only the solution on the right-hand side is considered as feasible.

4.3 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:

$$\sum_{ji \in A_{\text{SA}}} x_{ji} \leq \sum_{ij \in A_{\text{SA}}} x_{ij} \quad , \quad \forall i \notin R, \quad i \neq r \ . \tag{10}$$

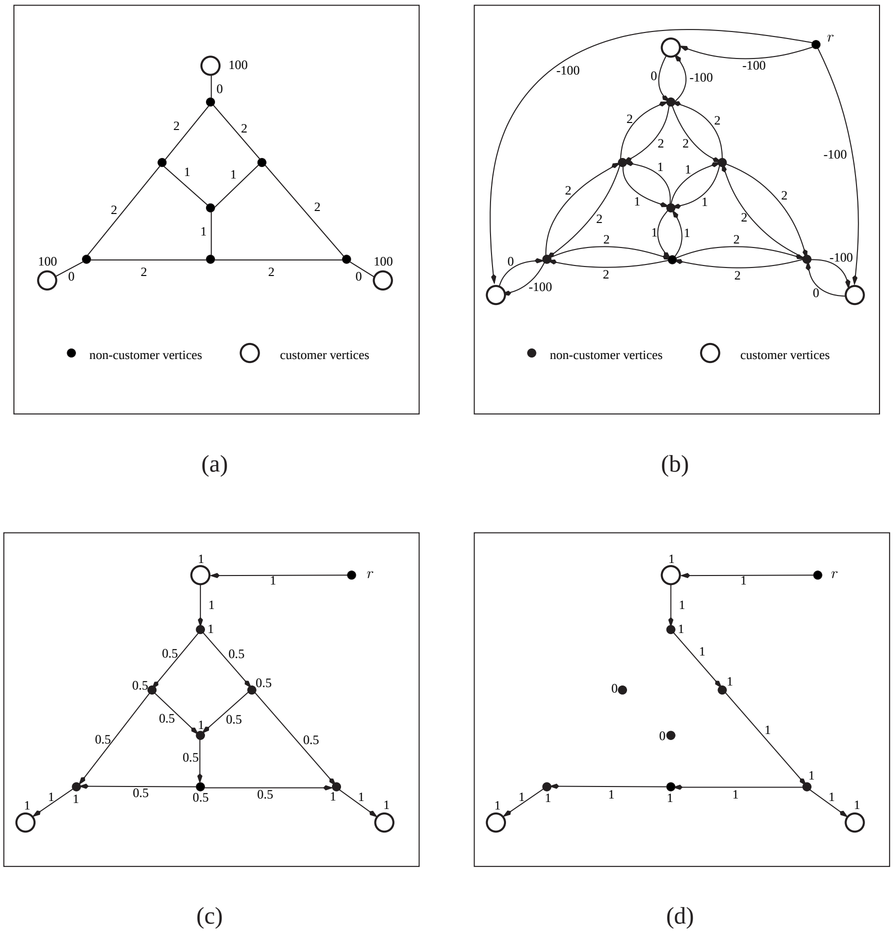


Fig. 3. (a) an input graph G; (b) after transformation into the Steiner arborescence problem; (c) solution of (CUT) LP-relaxation, $c(LP_{CUT}) = 7.5$; LP-values of x and y variables are shown; (d) the solution of (CUT) LP-relaxation augmented with flow-balance constraints has cost 8 and corresponds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

These so-called *flow-balance constraints* were introduced by Koch and Martin in [20] for the Steiner tree problem. In contrast to the Steiner tree problem where flow-balance constraints are added for all Steiner vertices, in PCST we can only apply them to non-customer vertices. They indeed represent a strengthening of the LP-relaxation of (5)-(9), as can be shown by an example in Figures 3 (c) and (d).

5 Branch-and-Cut Algorithm

To solve the proposed ILP formulation we use a branch-and-cut algorithm, whose non-standard ingredients are outlined next. At each node of the branch-and-bound tree we solve the LP-relaxation (CUT), obtained by replacing the integrality requirements (8) by $0 \leq y_i \leq 1, \forall i \in V_{\text{SA}} \setminus \{r\}$ 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 8.1.

5.1 Initialization

There are exponentially many constraints of type (6), 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:

$$x_{ij} + x_{ji} \leq y_i, \quad \forall i \in V_{\text{SA}} \setminus \{r\}, \quad (i,j) \in A_{\text{SA}} \tag{11}$$

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. 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 6.

5.2 Separation

During the separation phase which is applied at each node of the branch-and-bound tree, we add constraints of type (6) 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 LP solution. For finding the maximum flow in a directed graph, we used an adaptation of Cherkassky and Goldberg's maximum flow algorithm [6][4].

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 ∈ R_SA, y_i > 0 do
    x' = x + EPS;
    repeat
        f = MaxFlow(G, x', r, i, S_r, S_i);
        Detect the cut δ+(S_r) such that x'(δ+(S_r)) = f, r ∈ S_r;
        if f < y_i then
            Insert the violated cut x(δ+(S_r)) ≥ y_i into the LP;
        end
        x'_ij = 1, ∀(i, j) ∈ δ+(S_r);
        if BACKCUTS then
            Detect the cut δ−(S_i) such that x'(δ−(S_i)) = f, i ∈ S_i;
            if S_i ≠ S̄_r then
                Insert the violated cut x(δ−(S_i)) ≥ y_i into the LP;
                x'_ij = 1, ∀(i, j) ∈ δ−(S_i);
            end
        end
    until f ≥ y_i or MAXCUTS constraints added;
end
```

Algorithm 1: Separation procedure.

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 $f = \mathit{MaxFlow}(G, x', r, i, S_r, S_i)$ 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 performed as long as the total number of the detected violated cuts is less than $MAXCUTS$ (100, in the default 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).

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

Chopra et al. [7] 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 6) confirm that for most of our instances setting EPS to a positive value increases the CPU time.

6 Computational Results

We tested our new approach outlined in Section 5 extensively on the following groups of instances:

- Johnson et al. [18] 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 11 instances of group `P` and 23 instances of group `K` with up to 400 vertices and 1 576 edges that have also been tested by Lucena and Resende [22] and Canuto et al. [5].
- Canuto et al. [5] generated a set of 80 test problems derived from the Steiner problem instances of the well-known OR-Library[5]. 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 scheme, 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.

All 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`.

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

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 [22] (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 on the original istances without running 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. [5]. 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 [22] (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 [9] 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.

Both algorithms, LR and our new ILP approach, solved all P and K instances to optimality. Thus, we omit the corresponding tables here and refer to Table 3 and the next subsection where we analyze the running time of our algorithm. All the instances of K, P, C and D groups are solved in the root node of the branch-and-cut tree.

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	0.1	0	0	0.1
C1-B	625	125	226	1.2	85	1.7	85	0.0	6	6	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	0.1
C3-A	625	160	277	1.1	414	1.2	414	0.1	0	0	0.3
C3-B	625	185	304	1.3	737	26.1	737	0.1	4	1	0.4
C4-A	625	178	300	1.2	618	1.7	618	0.2	2	6	0.9
C4-B	625	218	341	1.3	1063	287.4	1063	0.2	4	0	0.6
C5-A	625	163	274	1.2	1080	80.4	1080	0.2	0	0	8.6
C5-B	625	199	314	1.7	1528	3487.1	1528	0.2	0	0	2.0
C6-A	1000	355	822	2.1	18	0.9	18	0.1	0	0	0.1
C6-B	1000	356	823	2.1	55	57.5	55	0.3	8	12	0.6
C7-A	1000	365	842	2.6	50	1.3	50	0.1	0	0	0.1
C7-B	1000	365	842	2.5	102	4.7	102	0.2	4	9	0.2
C8-A	1000	367	849	2.7	361	33.4	361	0.2	0	0	0.6
C8-B	1000	369	850	3.0	500	215.0	500	0.3	4	1	0.6
C9-A	1000	387	877	2.4	533	84.1	533	1.2	12	16	0.9
C9-B	1000	389	879	2.8	694	1912.6	694	1.0	14	23	1.1
C10-A	1000	359	841	3.3	859	160.3	859	0.8	8	0	2.5
C10-B	1000	323	798	3.4	1069	3502.3	1069	0.6	10	0	4.3
C11-A	2500	489	2143	9.4	18	3.4	18	0.2	2	0	0.2
C11-B	2500	489	2143	9.5	32	68.5	32	4.2	48	25	3.2
C12-A	2500	484	2186	6.8	38	37.0	38	0.3	4	0	0.3
C12-B	2500	484	2186	6.8	46	126.7	46	0.6	12	8	0.6
C13-A	2500	472	2113	9.8	236	332.5	236	0.7	4	1	2.1
C13-B	2500	471	2112	9.8	258	3092.7	258	2.9	56	22	5.4
C14-A	2500	466	2081	7.5	293	1749.8	293	0.6	2	0	0.6
C14-B	2500	459	2048	7.5	318	1142.3	318	0.6	4	0	0.6
C15-A	2500	406	1871	6.5	501	54223.3	501	1.1	12	1	6.4
C15-B	2500	370	1753	6.0	551	—	*551	0.6	4	0	4.4
C16-A	12500	500	4740	2.4	11	204.5	11	1.4	6	9	3.7
C16-B	12500	500	4740	2.4	11	205.1	11	1.4	6	9	3.6
C17-A	12500	498	4694	2.4	18	250.3	18	2.4	12	30	2.8
C17-B	12500	498	4694	2.3	18	388.2	18	1.7	8	15	2.8
C18-A	12500	469	4569	2.6	111	20031.8	111	1.6	2	2	5.3
C18-B	12500	465	4538	2.9	113	—	*113	3.3	14	3	25.7
C19-A	12500	430	3982	2.9	146	152217.1	146	0.7	2	0	3.9
C19-B	12500	416	3867	2.8	146	18999.6	146	0.6	2	0	4.1
C20-A	12500	241	1222	6.1	265	—	**266**	0.2	2	0	22.6
C20-B	12500	133	563	5.0	267	—	*267	0.1	2	0	51.8
C-AVG	4 156.3	348.5	1 733.4	3.8	334.3	—	334.3	0.7	7.0	5.0	4.4

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	1250	231	440	4.9	18	0.4	18	0.1	0	0	0.2
D1-B	1250	233	443	4.9	106	5.5	106	0.1	8	8	0.5
D2-A	1250	257	481	4.9	50	0.7	50	0.1	0	0	0.2
D2-B	1250	264	488	4.9	218	2.2	218	0.1	0	0	0.3
D3-A	1250	301	529	5.5	807	12.2	807	0.2	0	0	1.1
D3-B	1250	372	606	6.3	1509	331.5	1509	0.4	2	0	0.9
D4-A	1250	311	541	5.6	1203	51.5	1203	0.4	0	0	3.0
D4-B	1250	387	621	7.2	1881	1551.3	1881	0.8	8	0	2.5
D5-A	1250	348	588	7.6	2157	597.5	2157	1.0	6	0	93.8
D5-B	1250	411	649	11.5	3135	—	*3135	1.4	6	0	7.3
D6-A	2000	740	1707	14.4	18	2.6	18	0.2	0	0	0.2
D6-B	2000	741	1708	14.7	67	225.8	67	1.2	20	4	3.0
D7-A	2000	734	1705	11.3	50	4.3	50	0.2	0	0	0.2
D7-B	2000	736	1707	11.4	103	154.1	103	0.3	2	1	0.4
D8-A	2000	764	1738	11.7	755	170.7	755	3.6	24	0	9.5
D8-B	2000	778	1757	12.3	1036	3267.9	1036	0.9	2	1	3.1
D9-A	2000	752	1716	17.9	1070	1346.7	1070	14.3	24	15	30.7
D9-B	2000	761	1724	20.9	1420	25052.5	1420	4.2	104	0	3.5
D10-A	2000	694	1661	14.6	1671	62590.0	1671	4.2	8	0	30.4
D10-B	2000	629	1586	18.5	2079	—	*2079	2.4	8	0	25.5
D11-A	5000	986	4658	27.7	18	33.9	18	1.4	2	4	0.5
D11-B	5000	986	4658	23.6	29	870.6	29	4.5	14	41	4.8
D12-A	5000	991	4639	23.1	42	281.7	42	1.4	10	2	1.6
D12-B	5000	991	4639	22.3	42	297.1	42	1.0	4	3	1.3
D13-A	5000	966	4572	27.7	445	24689.7	445	6.8	26	4	18.9
D13-B	5000	961	4566	28.0	486	4464.2	486	3.4	12	0	3.3
D14-A	5000	946	4500	35.5	602	—	*602	3.7	12	0	60.2
D14-B	5000	931	4469	37.2	665	—	*665	4.7	14	1	22.4
D15-A	5000	832	4175	47.1	1040	—	*1042	16.9	24	14	146.8
D15-B	5000	747	3896	49.2	1107	1691918.5	**1108**	4.9	10	1	24.2
D16-A	25000	1000	10595	10.8	13	9957.8	13	3.5	8	4	19.3
D16-B	25000	1000	10595	10.8	13	6129.7	13	4.5	2	3	10.1
D17-A	25000	999	10534	10.8	23	16939.8	23	4.5	6	27	38.0
D17-B	25000	999	10534	10.7	23	13742.4	23	4.6	6	35	27.0
D18-A	25000	944	9949	11.7	218	—	*218	49.4	60	2	16.1
D18-B	25000	929	9816	12.0	223	—	**223**	5.7	10	0	15.8
D19-A	25000	897	9532	12.4	306	—	**306**	11.6	20	42	49.3
D19-B	25000	862	9131	13.1	310	—	**310**	11.4	28	2	105.5
D20-A	25000	488	2511	37.3	529	—	**536**	0.7	0	0	48.3
D20-B	25000	307	1383	32.9	530	—	**537**	0.3	2	0	122.1
D-AVG	8 312.5	705.2	3 793.7	17.4	650.4	—	650.9	4.6	12.3	5.4	23.8

6.1 Summarizing Results on K, P, C and D Groups

In Table 3 we summarize experimental results of the branch-and-cut algorithm without running the reduction procedures proposed in Section 3. We represent the following values for each group K, P, C and D: for each instance where the LR algorithm proved optimality, and whose $t_{ILP} > 0.2$, we calculate the *speed-up factor* t_{LR}/t_{ILP}. We then present the average (AVG), minimum (MIN) and maximum (MAX) value of this factor per group. We also count the number of instances where we proved optimality for values that were not guaranteed to be optimal by LR, and also the number of instances where optimal solution were not known before. The last two columns show the number of instances for which our ILP approach needed not more than 0.2 seconds to solve them and the total number of instances per group, respectively.

If we assume the conservative hardware speed-up factor of 20, the results of Table 3 show that:

- our algorithm is on average about 30 times slower for the instances of group K. However, all of them could be solved to optimality by both LR and our algorithm within a short running time (within 1000 seconds, in the worst case).
- on the remaining instances that could be solved to optimality by the LR algorithm, our new approach is significantly faster, and the average running time speed-up factor lies between 11 and 156.
- our new approach is able to solve all instances from groups K, P, C and D to optimality within a very short time, even if preprocessing (used also by Lucena and Resende [22]) is turned off.

Table 3. Comparison of running-time speed-up factors over all instances of a group: average, minimal and maximal factors for each group are given.

Group	$t_{LR}/(20 \cdot t_{noprep})$			new status of optimality			
	AVG	MIN	MAX	proven	new value	$t_{noprep} \leq 0.2$	# of inst.
K	0.1	0.03	0.2	-	-	7	23
P	11.9	3.3	28.7	-	-	6	11
C	116.9	0.1	1961.6	3	1	11	40
D	155.7	0.2	3498.6	5	7	12	40

6.2 The Advantage of Preprocessing for the Branch-and-Cut Algorithm

The preprocessing techniques still play an important role in solving larger instances to optimality. This can be seen in Table 4 where the results of our new approach on the set E of benchmark instances with and without preprocessing are compared. 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. For these more challenging instances both variants of the algorithm are terminated by setting the `cplexTimeLimit` parameter to 2 000 seconds. The results indicate the advantage of preprocessing when the size of the LPs increases. The results document that preprocessing can reduce the number of vertices and edges of the original graph by about 30%, respectively, 50%, on average. For 5 out of 40 instances the algorithm did not find the optimal solution within the given time limit when preprocessing was turned off. Owing to Gomory cuts, the instances of this group are also solved without branching.

6.3 Tuning the Separation Strategy

In our default implementation we used nested cuts and back-cuts. The parameter EPS was set to 0.0, which means that we did not calculate the smallest cardinality cuts. Our computational experiments have shown that the usage of back-cuts is crucial for our implementation. When we omitted the back-cuts, some of the larger instances (even of groups `C` and `D`) could not be solved to proven optimality - the algorithm terminated because of excessive memory consumption.

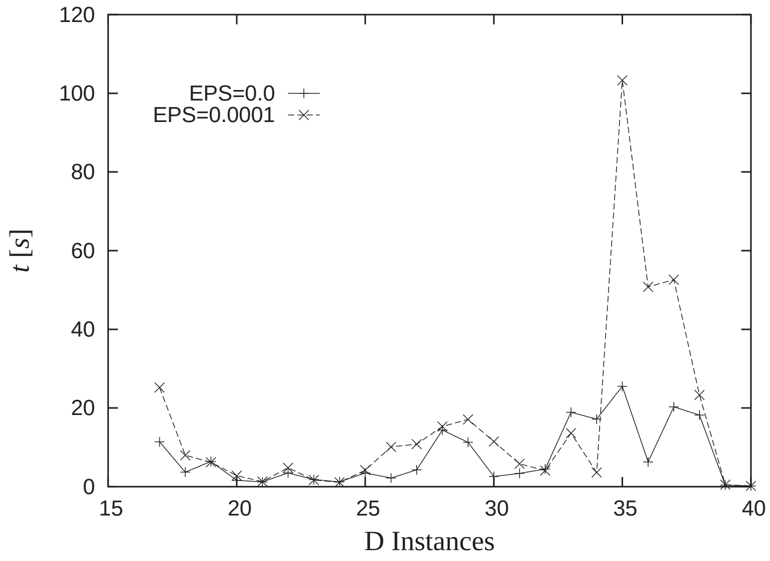
The role of the parameter EPS within the separation is studied in Table 5 where we show CPU times in seconds averaged over each group of instances. All the runs were limited to 1 000 seconds (except for group `E`, where we set the limit to 2 000). As the first two columns in Table 5 document, the usage of minimum cardinality cuts within the separation plays the most important role for solving the instances in the `K` group. On the other hand, when 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 problem and the time for solving a single LP-relaxation. Figures 4(a) and (b) depict the running time performance of the branch-and-cut algorithm with and without smallest cardinality cuts for the largest 24 instances of groups `D` and `E`, respectively. Although there are some instances where the usage of smallest cardinality cuts may reduce the total running time, the overall performance is getting worse. For two largest groups, `D` and `E`, the usage of smallest cardinality cuts increases the running time by more than a factor of two.

Table 5 also documents the crucial role of using inequalities (11) within the initialization procedure. These inequalities say that each edge can only be used in one direction in any feasible solution (see Section 5.1) and only if one of its incident vertices is in the solution. The last two columns represent results obtained by omitting inequalities (11) from the initialization.

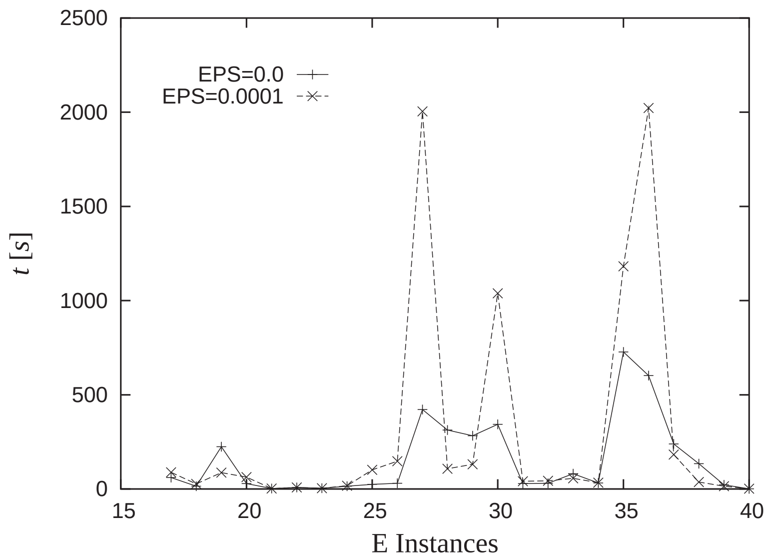
These inequalities that forbid subtours of size two play the most important role in our implementation. By separating these inequalities instead of inserting them in the initialization phase already, the algorithm is not able to solve many of the instances within a given time limit. This negative effect can not be compensated by the –

Table 4. 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	3125	651	1246	21.5	13	0.1	0	0	0.5	0	0
E01-B	3125	655	1250	21.8	109	0.9	10	21	1.6	6	9
E02-A	3125	694	1304	20.7	30	0.1	0	0	0.5	0	0
E02-B	3125	697	1307	20.7	170	0.4	4	4	1.6	6	5
E03-A	3125	813	1414	29.4	2231	5.3	6	18	25.7	26	5
E03-B	3125	962	1572	30.9	3806	4.0	6	0	16.6	14	6
E04-A	3125	829	1425	24.9	3151	14.3	8	2	288.9	18	12
E04-B	3125	980	1588	26.2	4888	7.1	8	0	20.2	6	0
E05-A	3125	893	1502	36.9	5657	41.6	14	2	—	—	—
E05-B	3125	1029	1644	45.0	7998	16.6	2	0	133.8	16	0
E06-A	5000	1821	4283	37.9	19	0.5	0	0	0.7	0	0
E06-B	5000	1821	4283	37.6	70	1.4	2	5	3.6	4	11
E07-A	5000	1863	4339	39.3	40	0.5	0	0	0.7	0	0
E07-B	5000	1865	4341	39.4	136	2.6	6	15	3.8	4	12
E08-A	5000	1902	4379	40.1	1878	56.5	10	19	79.2	10	8
E08-B	5000	1911	4387	50.4	2555	10.5	14	2	22.4	22	2
E09-A	5000	1909	4388	50.5	2787	60.2	10	5	587.5	20	3
E09-B	5000	1918	4397	54.7	3541	14.7	6	2	23.8	10	3
E10-A	5000	1716	4181	60.6	4586	221.8	96	2	538.7	12	0
E10-B	5000	1594	4045	84.6	5502	27.7	10	0	351.0	145	2
E11-A	12500	2491	12063	145.1	21	2.6	0	0	5.8	0	0
E11-B	12500	2491	12063	146.2	34	8.1	2	8	8.8	2	16
E12-A	12500	2490	12090	82.5	49	4.2	2	4	2.1	0	0
E12-B	12500	2490	12090	85.5	67	14.7	22	21	22.9	32	24
E13-A	12500	2430	11949	148.2	1169	24.6	12	3	97.6	20	2
E13-B	12500	2407	11915	146.7	1269	30.1	14	7	27.3	10	1
E14-A	12500	2366	11872	144.2	1579	420.3	200	0	71.2	32	1
E14-B	12500	2311	11737	145.7	1716	305.9	166	6	139.8	160	1
E15-A	12500	2044	10845	207.8	2610	276.4	150	0	313.0	54	1
E15-B	12500	1864	10264	234.3	2767	342.4	136	4	694.1	112	5
E16-A	62500	2500	29332	82.2	15	28.4	8	17	80.2	16	27
E16-B	62500	2500	29332	81.9	15	30.6	8	23	103.3	18	22
E17-A	62500	2500	29090	81.6	25	80.7	20	19	145.9	26	27
E17-B	62500	2500	29090	81.8	25	31.8	10	18	244.3	44	23
E18-A	62500	2378	28454	85.7	555	725.6	170	0	—	—	—
E18-B	62500	2347	28269	86.9	564	600.0	143	0	—	—	—
E19-A	62500	2156	25011	92.2	747	234.9	96	5	—	—	—
E19-B	62500	2085	23641	94.0	758	134.7	70	4	1145.9	84	7
E20-A	62500	1525	12770	107.3	1331	21.6	16	1	1232.2	48	3
E20-B	62500	861	3881	231.5	1342	1.1	2	0	—	—	—
E-AVG	20 781.3	1 781.5	10 325.8	82.1	1 645.6	95.1	36.5	5.9	—	—	—



(a)



(b)

Fig. 4. Running time comparison of separation with and without smallest cardinality cuts, $EPS = 0.0001$ and $EPS = 0.0$, respectively. The largest 24 instances of the group `D` (a) and `E` (b) are shown: from `D09-A` to `D20-B`, and from `E09-A` to `E20-B`, respectively. The total running time of the branch-and-cut algorithm (without preprocessing time) is shown.

Table 5. 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 constraints (11). Preprocessing times are omitted.

Group	Init. with ineq. (11)		Init. without ineq. (11)	
	$EPS = 0$	$EPS = 10^{-4}$	$EPS = 0$	$EPS = 10^{-4}$
K	48.8	7.6	88.8	2.9
P	0.2	0.3	21.4	2.2
C	0.8	0.9	—	0.8
D	4.5	9.6	—	—
E	95.1	199.1	—	—

sometimes considerable – improvement in running time caused by the separation with smallest cardinality cuts (in particular for K and P groups).

The difficulties with subtours of size two usually arise when a customer vertex i is connected to a non-customer vertex j by an edge with cost $c_{ij} < p_i$. 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 the inequalities stating $x_{ij} \leq y_i, \forall i \in V_{\text{SA}} \setminus \{r\}$, instead of (11), solutions of the initial LP contain subtours of size two on such pairs (i, j) of vertices. Figure 5 illustrates examples of fractional LP-solutions immediately after initialization, i.e. before any separation is called. Similarly, the problems appear also if two customer vertices i and j are connected by an edge whose $c_{ij} < \min\{p_i, p_j\}$.

Although the number of these inequalities is linear in number of edges $O(|A_{\text{SA}}|)$, they usually do not slow down the performance of solving a single LP-relaxation. On the other side, they may save a huge number of separation calls.

6.4 Testing Real-World Instances

In this section we consider a set of 35 instances based on real-world examples that have been used in the design of fiber optic networks for some German cities [2][6]. The instances are generated according to GIS data bases: connections and positions of vertices are based on real infrastructures, but, for reasons of data protection, the choice of customers and their prizes are slightly changed. Our instances are divided in two groups: `Cologne1` and `Cologne2`. Basic properties of these instances, like the number of vertices $|V|$, the number of edges $|E|$ and the number of customers $|R|$ are shown in Table 6. For each of the two groups there are 5 subgroups, each of which contains a number of equal-sized instances. The number of instances of each subgroup is also shown in the table.

[6] Instances are available at `http://www.ads.tuwien.ac.at/pcst`.

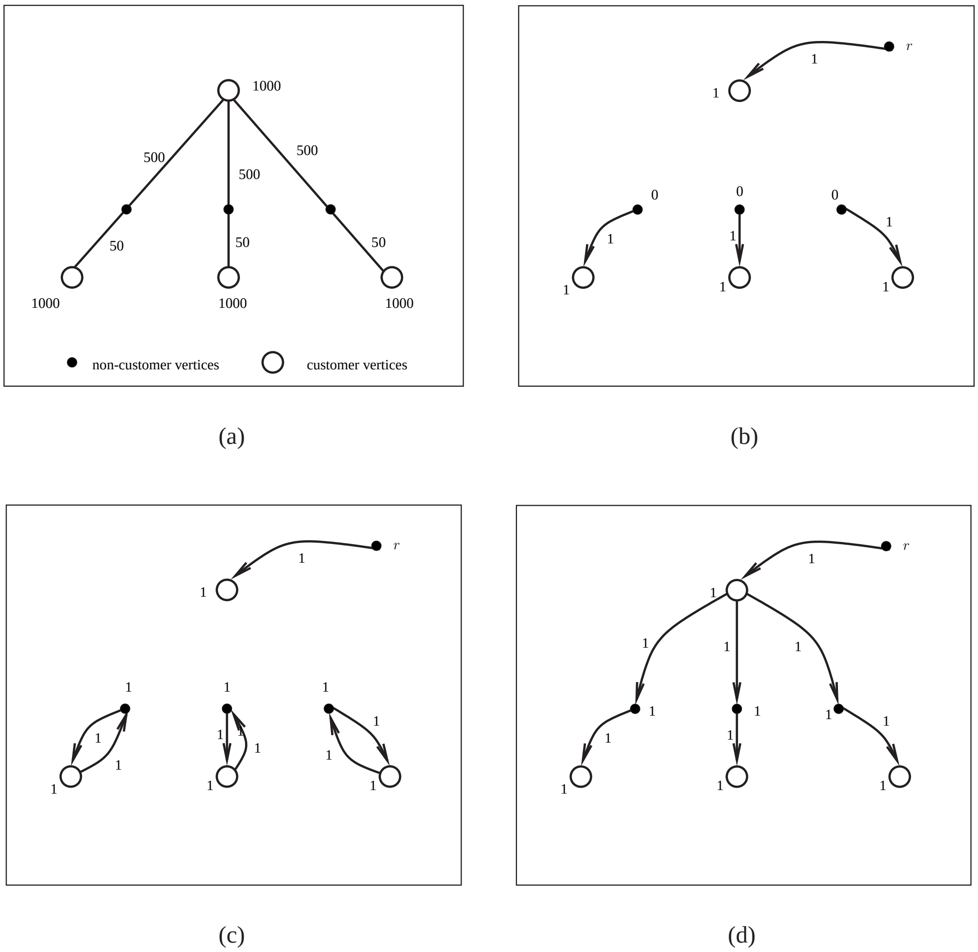


Fig. 5. The advantage of explicit initialization with constraints (11) can be seen when comparing LP-solutions obtained immediately after initialization: (a) Given instance with edge costs and vertex prizes; (b) LP-solution when we omit inequalities (11). LP-values of edges and vertices are shown, $c(LP) = 150$; (c) LP-solution if $x_{ij} \leq y_i, \forall i \in V_{\text{SA}} \setminus \{r\}$ are used in initialization. $c(LP) = 300$; (d) LP-solution when (11) are added explicitly in the initialization. $c(LP) = 1650$, the solution is already optimal, no separation needed.

These instances contain an existing subnetwork that needs to be augmented to serve new customers. This subnetwork can be shrunk into a single vertex, by replacing multiple edges by cheapest connections and discarding self-loops. After this transformation, we only need to consider the rooted PCST problem, with the shrunk vertex as a root. Two examples of these instances are shown in Figure 6.

(a) (b)

Fig. 6. Examples of two instances from `Cologne1` group. Optimal solution of (a) `i02M2`; (b) `i04M3`. Shadowed vertices represent customers, while bold lines belong to the existing network. Grid lines in the background determine customers' prizes.

Since the instances are typically very dense (i.e. they are almost complete graphs), it does not pay to apply degree-n test. Our experiments have shown that because the number of customers is typically very small, the minimum adjacency test does not reduce the instances at all. Hence, we applied only the least-cost test, and, as shown in column $|E'|$ of Table 6, the savings obtained in that way are greater than 90%.

Table 6. Properties of two groups of real-world instances, `Cologne1` and `Cologne2`.

Cologne1						Cologne2					
Group	$\|V\|$	$\|E\|$	$\|R\|$	$\|E'\|$	# of inst.	Group	$\|V\|$	$\|E\|$	$\|R\|$	$\|E'\|$	# of inst.
i01	768	69077	10	6332	3	i01	1819	213973	9	16743	4
i02	769	69140	11	6343	3	i02	1820	213915	7	16740	4
i03	771	69100	13	6343	3	i03	1825	214095	12	16762	4
i04	761	68907	3	6293	3	i04	1817	213859	4	16719	4
i05	761	68934	3	6296	3	i05	1826	214013	13	16794	4

Table 7 shows the performance of our ILP approach on the real-world instances. We compare two approaches based on the initialization of the LP with and without (11), i.e. generalized subtour elimination constraints of size two. The results document that all instances of the `Cologne1` group could be solved to optimality in less than 40 minutes. For instances of the `Cologne2` group running times of up to 12 hours may occur. However, as it is usually the case for branch and cut algorithms, the optimal solution was found much earlier, usually in less than 2 hours. Finally, we also provide optimal values in the OPT columns.

While constraints (11) were shown to be highly advantageous for the previous instances from the literature, Table 7 documents that for these specific real-world instances there is a trade-off between the size of the underlying LP and the number of separation calls that can be saved. The reason is the very small percentage of customer vertices, which is less than 2% and 1%, for the `Cologne1` and `Cologne2` groups, respectively. The number of subtours of size two usually depends on the number of negative edges which directly corresponds to the number of customer vertices. On the other side, using (11), we insert $2 \cdot |A_{\text{SA}}|$ inequalities in the initialization phase, which represents a disadvantage for these very large instances with only few customer vertices. All `*M1` instances of the `Cologne2` group could be solved to optimality in less than 30 seconds, and they always represent single-vertex solutions, thus we omit them from Table 7.

We can conclude that our ILP approach shows that it can be used within real-world applications to solve instances that appear in practice. The approach is able to solve very large graphs (with up to 1 825 vertices and 214 095 edges) to proven optimality within less than 12 hours, in the worst case. Since we are dealing with off-line network design problems, such a running time is still reasonable.

7 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 district heating and fiber optic networks. Selecting the most profitable customers and connecting them by a least-cost network immediately leads to the problem of computing a Steiner tree,

Table 7. Results on two groups of real-world instances. We compare two ILP approaches, where the initialization is done with and without constraints (11) (faster running times are shown in boldface). All instances are solved to optimality.

Cologne1				Cologne2			
Instance	Without (11) $t\ [s]$	With (11) $t\ [s]$	OPT	Instance	Without (11) $t\ [s]$	With (11) $t\ [s]$	OPT
i01M1	**0.5**	2.9	109271.5	i01M2	**2.1**	5.1	355467.7
i01M2	**252.3**	487.8	315925.3	i01M3	31025.7	**27331.9**	628833.6
i01M3	1371.4	**1195.8**	355625.4	i01M4	45002.1	**40927.5**	773398.3
i02M1	**0.5**	2.9	104065.8	i02M2	**107.0**	110.7	288946.8
i02M2	**431.8**	598.2	352538.8	i02M3	**9034.4**	14173.6	419184.2
i02M3	2353.6	**1810.9**	454365.9	i02M4	**13322.0**	19124.3	430034.3
i03M1	**0.5**	3.1	139749.4	i03M2	907.1	**855.9**	459918.9
i03M2	362.6	**326.8**	407834.2	i03M3	**37416.1**	42150.0	643062.0
i03M3	1140.2	**755.9**	456125.5	i03M4	42752.0	**42237.7**	677733.1
i04M1	**0.5**	2.8	25282.6	i04M2	**2.5**	5.4	161700.5
i04M2	**20.0**	22.6	89920.8	i04M3	**5095.1**	13259.2	245287.2
i04M3	**42.1**	77.7	97148.8	i04M4	**4298.1**	8700.1	245287.2
i05M1	**0.5**	2.8	26717.2	i05M2	**2107.0**	2568.7	571031.4
i05M2	**94.7**	122.9	100269.6	i05M3	**11852.9**	19655.4	672403.1
i05M3	443.8	**399.4**	110351.2	i05M4	**16203.8**	16343.5	713973.6

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 finding the optimum without branching. Finally, we also solve to optimality a number of large real-world instances arising in the design of urban fiber optic networks.

Acknowledgments

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

References

1. Y. P. Aneja. An integer linear programming approach to the Steiner problem in graphs. *Networks*, 10:167–178, 1980.
2. P. Bachhiesl, M. Prossegger, G. Paulus, J. Werner, and H. Stögner. Simulation and optimization of the implementation costs for the last mile of fiber optic networks. *Networks and Spatial Economics*, 3(4):467–482, 2003.
3. J. E. Beasley. An SST-based algorithm for the Steiner problem in graphs. *Networks*, 19:1–16, 1989.
4. 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.
5. 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.
6. B. V. Cherkassky and A. V. Goldberg. On implementing push-relabel method for the maximum flow problem. *Algorithmica*, 19:390–410, 1997.
7. 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.
8. S. Chopra and M. R. Rao. The Steiner tree problem I: Formulations, compositions and extension of facets. *Mathematical Programming*, 64:209–229, 1994.
9. J. J. Dongarra. Performance of various computers using standard linear equations software (linpack benchmark report). Technical Report CS-89-85, University of Tennessee, 2004.
10. C. W. Duin and A. Volgenant. Some generalizations of the Steiner problem in graphs. *Networks*, 17(2):353–364, 1987.
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. G. Gutin and A. Punnen, editors. *The Traveling Salesman Problem and its Variations*. Kluwer, 2002.
17. J. Hackner. *Energiewirtschaftlich optimale Ausbauplanung kommunaler Fernwärmesysteme*. PhD thesis, Vienna University of Technology, Austria, 2004.
18. 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.
19. 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.
20. T. Koch and A. Martin. Solving Steiner tree problems in graphs to optimality. *Networks*, 32:207–232, 1998.
21. I. Ljubić, R. Weiskircher, U. Pferschy, G. W. Klau, P. Mutzel, and M. Fischetti. Solving the prize-collecting Steiner tree problem to optimality. In *Proc. of the Seventh Workshop on Algorithm Engineering and Experiments (ALENEX 05)*. SIAM, 2005. To appear.
22. 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.
23. M. Minkoff. The prize-collecting Steiner tree problem. Master's thesis, MIT, May, 2000.
24. A. Segev. The node-weighted Steiner tree problem. *Networks*, 17:1–17, 1987.
25. E. Uchoa. Reduction tests for the prize-collecting Steiner problem. Technical Report RPEP Vol.4 no.18, Universidade Federal Fluminense, Engenharia de Produção, Niterói, Brazil, 2004.
26. R. T. Wong. A dual ascent based approach for the Steiner tree problem in directed graphs. *Mathematical Programming*, 28:271–287, 1984.