Solving the Prize-Collecting Steiner Tree Problem to Optimality*

Ivana Ljubić[1], René Weiskircher[1], Ulrich Pferschy[2], Gunnar 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] 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 in the design of utility networks (eg. fiber optics or district heating) where profit generating customers and the network connecting them have to be chosen in the most profitable way.
Our main contribution is an efficient implementation using a directed graph model and connectivity inequalities corresponding to cuts in the graph. This enables us to efficiently separate sets of violated inequalities using a maximum flow algorithm.
Our implementation solves all benchmark instances from the literature to optimality, including eight for which the optimum was not known. We also present optimal results on very large real world instances that represent fiber optic networks in a German city.

1 Introduction

Recent advances in technology have made fiber optic connections for households economically feasible. In a typical planning scenario the input is a set of potential customers together with the discounted future profits they would generate, and a potential network for laying the fiber. Costs of the network are dominated by labor and right-of-way charges for laying the fiber. The situation is similar for other utilities like gas or district heating.

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 network. 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 mathematically as follows:

Definition 1 (Prize-Collecting Steiner Tree Problem, PCST). *Let $G = (V, E, c, p)$ be an undirected graph with vertex- and edge-weights as defined above. The* 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 minimizes the objective function $c(T) = \sum_{v \notin V_T} p(v) + \sum_{e \in E_T} c(e)$.*

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(a) illustrates an example of a PCST instance and Figure 1(d) a feasible solution for that instance.

* This work has been partly supported by the RTN ADONET 504438. 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.

When planning an augmentation of a network by a set of new customers, we model the problem as a *rooted* PCST: We shrink the existing network (connected or not) into a single vertex that must be contained in the solution.

The primary focus of this paper is a construction of an efficient algorithmic implementation rather than a theoretical study of the problem. In the next section we give a short overview of previous work on PCST and some of its relatives. In Section 3 we introduce our cut-based ILP model and describe how it can be solved in a branch-and-cut framework in Section 4. Extensive computational experiments on instances known from the literature are reported in Section 5. It turns out that the new approach can solve all of them to proven optimality (including eight for which optimum was not known) within a short running time. We also present results on fiber optic networks of a German city that show that our exact method is applicable in practice as well.

2 Previous Work

In 1987, Segev [21] 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 that must be included in the solution.

The PCST has been introduced by Bienstock et al. [3], where a factor 3 approximation algorithm has been proposed. Several other approximation algorithms have been developed. Goemans and Williamson presented in [13] 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. [14], 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. [11] achieves a ratio of $2 - \frac{2}{n}$ within the same time.

Recently, two meta-heuristic approaches for PCST have been developed: Canuto et al. [4] proposed a multi-start local-search-based algorithm with perturbations; Klau et al. [16] developed an evolutionary algorithm with incorporated local improvement for the problem.

Lower Bounds and Polyhedral Studies Fischetti [12] 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. Engevall et al. [10] proposed another ILP formulation for the NWST, based on the *shortest spanning tree* problem formulation, introduced originally by Beasley [2] for the Steiner tree problem.

Lucena and Resende [18] presented a cutting plane algorithm for the PCST based on generalized subtour elimination constraints. Their algorithm contains basic reduction steps similar to those already given by Duin and Volgenant [9], and was tested on 114 benchmark instances – all but 16 of them have been solved to proven optimality.

3 ILP Formulation of the Problem

In order to achieve a tighter LP-relaxation, we transform the original problem defined on an undirected graph into a problem on a directed graph thus obtaining a Steiner arborescence problem. Chopra and Rao showed that the LP-relaxation for the directed graph is superior to the formulation on the undirected graph (see [7]).

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: Each arc (r, j) has costs $-p_j$ while all other arcs (i, j) with $i \neq r$ have costs $c_{ij} - p_j$. An example of this transformation can be found in Figures 1 (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 a variable vector $x \in \{0,1\}^{|A_{\text{SA}}|}$ where the component for an arc in A_{SA} is one if and only if it is in T_{SA} and zero otherwise. Furthermore, to indicate which of the vertices from $V_{\text{SA}} \setminus \{r\}$ belong to the solution, we use a variable vector $y \in \{0,1\}^{|V_{\text{SA}}|-1}$.

Our ILP-formulation (introduced in [12] 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 the directed cut $\delta^-(S) = \{(i,j) \mid i \in \overline{S}, j \in S\}$. 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:

$$(CUT) \qquad \min \sum_{ij \in A_{\text{SA}}} c'_{ij} x_{ij} + \sum_{i \in V_{\text{SA}}} p_i \tag{1}$$

$$\text{subject to} \quad \sum_{ji \in A_{\text{SA}}} x_{ji} = y_i \qquad \forall i \in V_{\text{SA}} \setminus \{r\} \tag{2}$$

$$x(\delta^-(S)) \geq y_k \qquad k \in S, r \notin S, \forall S \subset V_{\text{SA}} \tag{3}$$

$$\sum_{ri \in A_{\text{SA}}} x_{ri} = 1 \tag{4}$$

$$x_{ij}, y_i \in \{0,1\} \qquad \forall (i,j) \in A_{\text{SA}}, \forall i \in V_{\text{SA}} \setminus \{r\} \tag{5}$$

The cut constraints (3) 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. The so-called *in-degree* equation (2) guarantees that every selected vertex has exactly one predecessor on its path from the root. Finally, the so-called *root-degree* constraint (4) is relevant only for the unrooted PCST and it makes sure that the artificial root r is connected only to a single vertex which is crucial for the connectedness of the solution.

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

$$y_i \leq x_{rj}, \ \forall i < j, \ i \in R \tag{6}$$

For the unrooted PCST, these inequalities assure that for each PCST solution the customer vertex adjacent to the artificial root is the one with the smallest index. Without this constraint, the same solution with n customers can be represented by n different solution vectors that just differ in the choice of the customer connected to the artificial root r. Computational results have shown that these inequalities significantly reduce the computation time, because they exclude many redundant solutions.

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, in-degree and out-degree must be equal, whereas in a branching non-customer vertex, the in-degree is always less than the outgoing degree. Thus, we have:

$$\sum_{ji \in A_{\text{SA}}} x_{ji} \leq \sum_{ij \in A_{\text{SA}}} x_{ij} \ , \quad \forall i \notin R, \quad i \neq r \ . \tag{7}$$

These so-called *flow-balance constraints* were introduced by Koch and Martin in [17] for the Steiner tree problem. They indeed represent a strengthening of the LP-relaxation of (2)-(6), as can be shown by an example in Figures 1 (d) and (e). For the classical Steiner tree problem, an analogous example can be found in [20].

4 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 (5) 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 (version 8.1) and ILOG Concert Technology (version 1.3).

Initialization There are exponentially many constraints of type (3), 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:

$$x_{ij} + x_{ji} \leq y_i, \quad \forall i \in V_{\text{SA}} \setminus \{r\}, \quad (i,j) \in A_{\text{SA}} \tag{8}$$

These constraints express the trivial fact that every arc incident to a vertex in the solution tree can be oriented only in one way. Although the LP may become large by adding all of these inequalities at once they offer a tremendous speedup for some instances since they do not have to be separated implicitly during the branch-and-cut algorithm. Further details are discussed in Section 5.

Separation During the separation phase which is applied at each node of the branch-and-bound tree, we add constraints of type (3) that are violated by the current solution of the LP-relaxation.

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 [5][4].

We outline the separation procedure 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 pairs of vertices (r, i), with $i \in R_{\text{SA}}, y_i > 0$. 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:

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

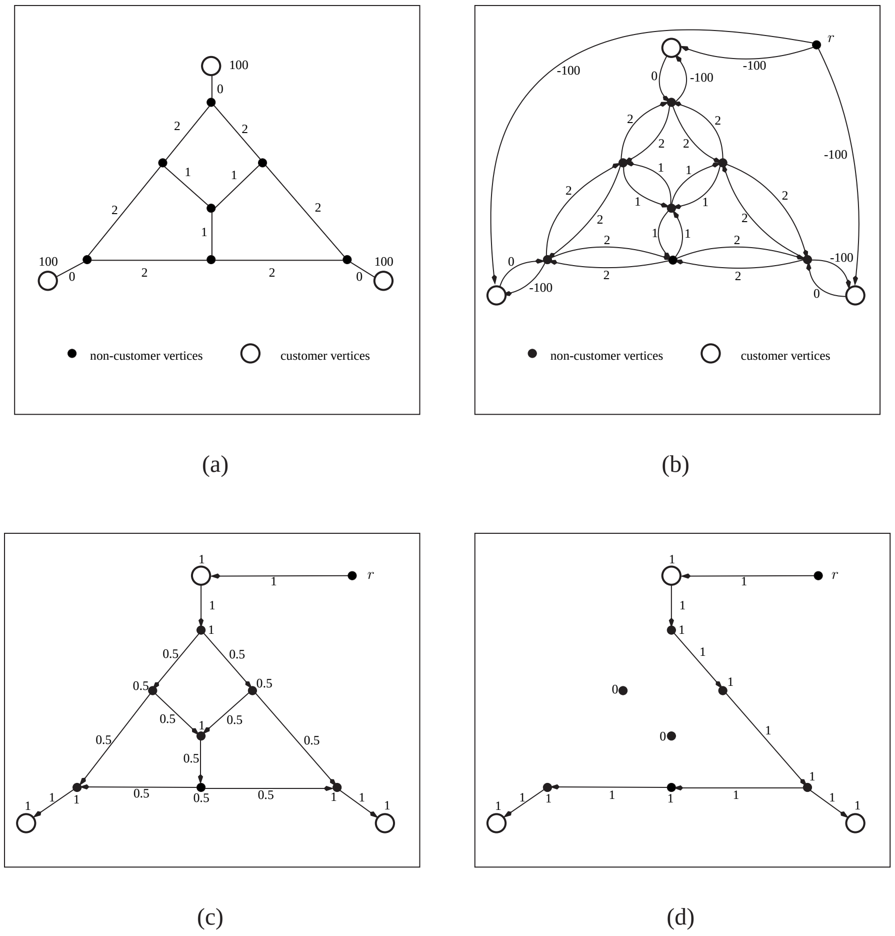


Fig. 1. (a) An input graph G. Each connection has fixed costs, hollow circles and filled circles represent customer and non-customer vertices, respectively. (b) Graph after transformation into the Steiner arborescence problem; (c) solution of (CUT) LP-relaxation, $c(LP_{CUT}) = 7.5$. LP-values of x and y variables are shown; (d) solution of (CUT) LP-relaxation augmented with flow-balance constraints has cost 8 and corresponds to an optimal solution.

- Subset $S_r \subset V_{\text{SA}}$ contains root vertex r and induces a minimum cut $\delta^+(S) = \{(i,j) \mid i \in S, j \in \overline{S}\}$ 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 $\delta^-(S) = \{(i,j) \mid i \in \overline{S}, j \in S\}$ 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* [17]: 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 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).

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

for $i \in R_{SA}, y_i > 0$ do
    $x' = x$;
    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;
            $x'_{ij} = 1, \forall (i,j) \in \delta^+(S_r)$;
            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
        end
    until $f \geq y_i$ or $MAXCUTS$ constraints added;
end
```

Algorithm 1: Separation procedure.

Chopra et al. [6] proposed the so-called *back-cuts*, also used in [17], 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.

5 Computational Results

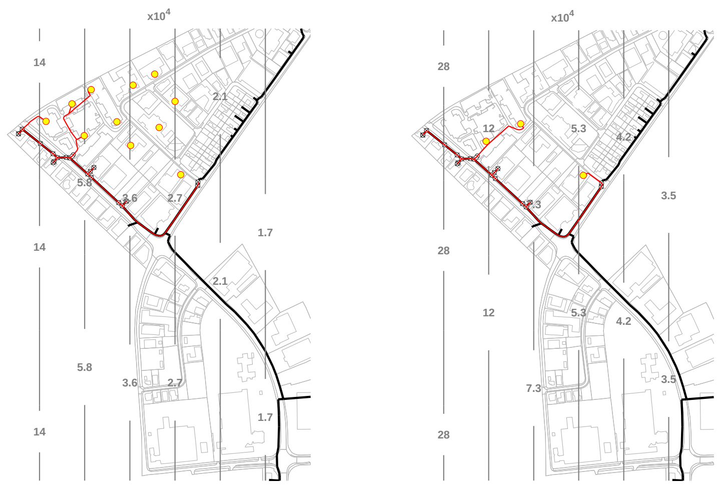
In this section we show our computational study on two sets of problem instances:

- The instances of the first set[5] have been used by Lucena and Resende in [18] to test their lower bounding procedure denoted in the sequel by LR. These instances can be divided into four groups: `K`, `P`, `C` and `D`. Groups `K` and `P` have been generated by Johnson et al. [14]. A detailed description of the generators for these instances can be found in [19]. We consider instances with up to 400 vertices and 1 576 edges that have also been tested in [4]. Canuto et al. [4] generated a set of 80 test problems derived from the Steiner problem instances of groups `C` and `D` from well-known OR-Library[6].
- The second set of 35 instances is based on real-world examples that have been used in the design of fiber optic networks for a German city [1][7]. 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`, and their basic properties, like the number of vertices $|V|$, the number of edges $|E|$ and the number of customers $|R|$, are shown in Table 5. The instances are divided in small

[5] Instances are available at `http://www.research.att.com/~mgcr/data/index.html`.
[6] OR-library: J. E. Beasley, `http://mscmga.ms.ic.ac.uk/info.html`.
[7] Instances are available at `http://www.ads.tuwien.ac.at/pcst`.

Table 1. Properties of two groups of real-world instances, `Cologne1` and `Cologne2`.

`Cologne1`						`Cologne2`					
Group	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert R\rvert$	$\lvert E'\rvert$	# of inst.	Group	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert R\rvert$	$\lvert E'\rvert$	# 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



(a) (b)

Fig. 2. Examples of two instances from `Cologne1` group. Optimal solution of (a) `i02M2`; (b) `i04M3`. Shadowed vertices represent customers, while non-shadowed vertices belong to the existing network. Grid lines in the background determine customers' prizes.

subgroups containing graphs of the same size. The number of instances of each subgroup is also shown in the table.

These real-world instances contain an existing network that needs to be augmented by new customers. The existing network can be shrunken into a single vertex, by replacing multiple edges by a cheapest connection and discarding self-loops. The problem is than considered as the rooted PCST problem, with the shrunk vertex as a root. Two examples of these instances are shown in Figure 2.

To remove redundant edges from these very dense graphs, we applied the least-cost test [9, 18] and, as Table 5 in column $|E'|$ documents, savings in the number of edges are greater than 90%.

Comparing LR with our results Both algorithms, LR and our new ILP approach, solved all `P` and `K` instances to optimality. LR approach did not prove optimality for 16 (out of 80) instances of groups `C` and `D`. Improving upon their results, our new ILP approach solved all instances were known from the literature to proven optimality. All instances were solved in the root node of the branch-and-cut tree.

In our default implementation we used nested cuts and back-cuts. Our computational experiments have shown that the initialization with constraints (8) and the usage of back-cuts are crucial for our implementa-

Table 2. Comparison of running-time speed-up factors over all instances of a group: average, minimal and maximal factor are given.

Group	$t_{LR}/(20 \cdot t_{ILP})$			new status of optimality		
	AVG	MIN	MAX	proven	new value	$t_{ILP} < 0.2$
K	0.1	0.1	0.3	-	-	4
P	11.0	3.0	27.2	-	-	6
C	218.7	0.1	5073.9	3	1	6
D	83.8	0.3	1516.1	5	7	2

tion. If we did not use both, some of the larger instances (of groups C and D) could not be solved to proven optimality. More details of our experiments can be found in our technical report [15].

Comparing our running time data (achieved on a Pentium IV with 2.8 GHz, 2 GB RAM, SPECint2000 = 1204) with the LR results (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 with 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 [8] the SGI machine is assigned a factor of 114 and 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.

Table 2 summarizes our experimental results and contains the following values for each group K,P,C and D: for each instance where the LR algorithm proved optimality, and whose running time in seconds $t_{ILP} \geq 0.2$, 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 (but the upper bounds found in [4] were equal to lower bounds obtained by LR), and also the number of instances where optimal solution were not known before. The last column shows the number of instances for which our ILP approach needed less than 0.2 seconds to solve them.

If we assume the conservative hardware speed-up factor of 20, the results of Table 2 show that:

– our algorithm is about 14 times on average 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 218.
– our new approach is able to solve all the instances to optimality within a very short time, even if preprocessing (used by Lucena and Resende [18]) is turned off. For a more exhaustive study on results with and without preprocessing, see our technical report [15].

Testing real-world instances Table 3 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 (8), i.e. generalized subtour elimination constraints of size two. The results document that all the instances of Cologne1 group could be solved to optimality in less than 2 400 seconds. For instances of Cologne2 group, we present the percentage of the gap between lower bound and optimum after two hours computation time: %-*gap* $= (OPT - LB)/OPT$. We also show total running times in seconds ($t\ [s]$) needed to prove optimality (the time limit was set to 45 000 seconds). Finally, we also provide optimal values in OPT columns.

Table 3. Results on two groups of real-world instances. We compare two ILP approaches, where the initialization is done with and without constraints (8). All instances of `Cologne1` group are solved to optimality, while for the `Cologne2` group, we show the gap in percent obtained after the time limit of 2 hours was exceeded.

`Cologne1`				`Cologne2`					
Instance	Without (8)	With (8)		Instance	Without (8)		With (8)		
	$t\ [s]$	$t\ [s]$	OPT		%-*gap*	$t\ [s]$	%-*gap*	$t\ [s]$	OPT
`i01M1`	**0.5**	2.9	109271.5	`i01M2`	0.0	2.1	0.0	5.1	355467.7
`i01M2`	**252.3**	487.8	315925.3	`i01M3`	1.9	31025.7	**1.7**	27331.9	628833.6
`i01M3`	1371.4	**1195.8**	355625.4	`i01M4`	**2.1**	45002.1	3.9	40927.5	773398.3
`i02M1`	**0.5**	2.9	104065.8	`i02M2`	0.0	107.0	0.0	110.7	288946.8
`i02M2`	**431.8**	598.2	352538.8	`i02M3`	**0.5**	9034.4	2.4	14173.6	419184.2
`i02M3`	2353.6	**1810.9**	454365.9	`i02M4`	**2.1**	13322.0	4.7	19124.3	430034.3
`i03M1`	**0.5**	3.1	139749.4	`i03M2`	0.0	907.1	0.0	855.9	459918.9
`i03M2`	362.6	**326.8**	407834.2	`i03M3`	**3.3**	37416.1	5.7	42150.0	643062.0
`i03M3`	1140.2	**755.9**	456125.5	`i03M4`	**3.7**	42752.0	5.6	42237.7	677733.1
`i04M1`	**0.5**	2.8	25282.6	`i04M2`	0.0	2.5	0.0	5.4	161700.5
`i04M2`	**20.0**	22.6	89920.8	`i04M3`	**0.0**	5095.1	2.1	13259.2	245287.2
`i04M3`	**42.1**	77.7	97148.8	`i04M4`	**0.0**	4298.1	0.1	8700.1	245287.2
`i05M1`	**0.5**	2.8	26717.2	`i05M2`	0.0	2107.0	0.0	2568.7	571031.4
`i05M2`	**94.7**	122.9	100269.6	`i05M3`	**0.1**	11852.9	1.0	19655.4	672403.1
`i05M3`	443.8	**399.4**	110351.2	`i05M4`	**0.5**	16203.8	0.8	16343.5	713973.6

While constraints (8) have shown to be very advantageous for the previous instances known from the literature, Table 3 documents that for the 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. This can be explained by a very small percentage of customer vertices, which is less than 2% and 1%, for `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 (8), we insert $2 \cdot |A_{\mathrm{SA}}|$ inequalities in the initialization phase, which obviously represents a disadvantage for such very large instances with only few customer vertices. All `*M1` instances of `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 3.

6 Conclusions

The prize-collecting Steiner tree problem (PCST) formalizes the planning problem encountered in the design of utility networks such as fiber optic, gas or district heating. Selecting the most profitable customers and connecting them by a least-cost network 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 our contribution is the construction of an algorithmic framework to solve large and difficult instances, known from the literature and also those appearing in practice, 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 all instances from the literature in a few seconds, deriving new optimal solution values and new certificates of optimality for a number of problems previously attacked. We also show that our exact method is applicable in praxis as well: for a set of 35 very large real-world instances used in the design of fiber optic networks (with up to 1 825 vertices and 214 095 edges) we derive provably optimal solutions.

Acknowledgments

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

References

1. 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.
2. J. E. Beasley. An SST-based algorithm for the Steiner problem in graphs. *Networks*, 19:1–16, 1989.
3. 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.
4. 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.
5. B. V. Cherkassky and A. V. Goldberg. On implementing push-relabel method for the maximum flow problem. *Algorithmica*, 19:390–410, 1997.
6. 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.
7. S. Chopra and M. R. Rao. The Steiner tree problem I: Formulations, compositions and extension of facets. *Mathematical Programming*, 64:209–229, 1994.
8. J. J. Dongarra. Performance of various computers using standard linear equations software (linpack benchmark report). Technical Report CS-89-85, University of Tennessee, 2004.
9. C. W. Duin and A. Volgenant. Some generalizations of the Steiner problem in graphs. *Networks*, 17(2):353–364, 1987.
10. 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.
11. 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.
12. M. Fischetti. Facets of two Steiner arborescence polyhedra. *Mathematical Programming*, 51:401–419, 1991.
13. 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.
14. 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.
15. G. Klau, I. Ljubić, A. Moser, P. Mutzel, P. Neuner, U. Pferschy, and R. Weiskircher. Solving the prize-collecting Steiner tree problem to optimality. Technical Report TR-186-1-04-01, Vienna University of Technology, 2004.
16. 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)*, LNCS. Springer-Verlag, 2004. to appear.
17. T. Koch and A. Martin. Solving Steiner tree problems in graphs to optimality. *Networks*, 32:207–232, 1998.
18. A. Lucena and M. G. C. Resende. Strong lower bounds for the prize-collecting Steiner problem in graphs. *Discrete Applied Mathematics*, 2003. to appear.
19. M. Minkoff. The prize-collecting Steiner tree problem. Master's thesis, MIT, May, 2000.
20. T. Polzin and S. V. Daneshmand. A comparison of Steiner tree relaxations. *Discrete Applied Mathematics*, 112:241–261, 2001.
21. A. Segev. The node-weighted Steiner tree problem. *Networks*, 17:1–17, 1987.