An Algorithmic Framework for the Exact Solution of Tree-Star Problems

Markus Leitner*[1], Ivana Ljubić †[2], Markus Sinnl‡[1], and Juan-José Salazar-González§[3]

[1] *ISOR, University of Vienna, Austria.*
[2] *ESSEC Business School of Paris, France.*
[3] *DEIOC, Universidad de La Laguna, Tenerife, Spain.*

Abstract

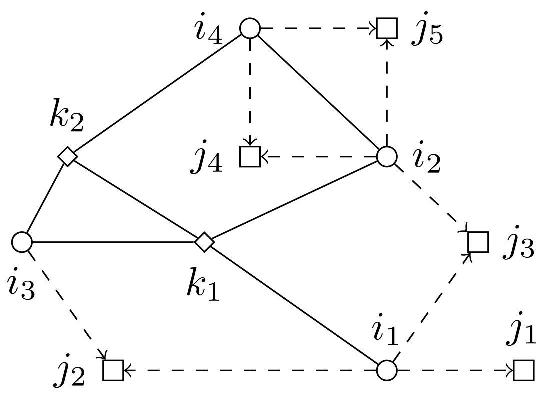
Many problems arising in the area of telecommunication ask for solutions with a tree-star topology. This paper proposes a general procedure for finding optimal solutions to a family of these problems. The family includes problems in the literature named as *connected facility location, rent-or-buy* and *generalized Steiner tree-star*. We propose a solution framework based on a branch-and-cut algorithm which also relies on sophisticated reduction and heuristic techniques. An important ingredient of this framework is a dual ascent procedure for asymmetric connected facility location. This paper shows how this procedure can be exploited in combination with various mixed integer programming formulations. Using the new framework, many benchmark instances in the literature for which only heuristic results were available so far, can be solved to provable optimality within seconds. To better assess the computational performance of the new approach, we additionally consider larger instances and provide optimal solutions for most of them too.

1 Introduction

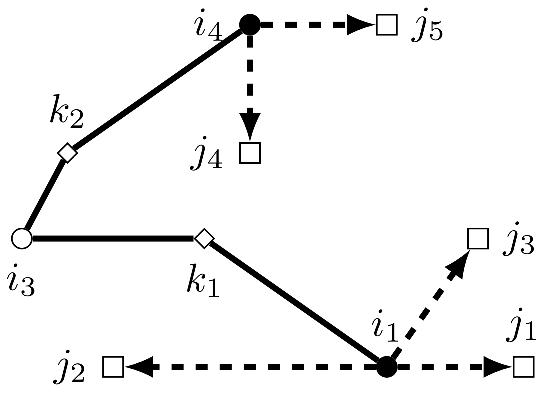
Many problems arising in the area of telecommunication ask for solutions possessing a tree-star topology. Consider a scenario in which, for example, servers have to be located on a network that will contain (or cache) information. Information on the servers is updated over time, and this incurs a fixed cost for every edge in the network along which this information is sent. Consequently, the cheapest way to update information over servers (once a choice of which servers to open has been made) is via a *tree* connecting all of them. End-clients requiring the information are served from their closest servers among a *star* topology. This family of tree-star problems includes several known problems in the literature as the *connected facility location, rent-or-buy* and *generalized Steiner tree-star problems in graphs*. The problems addressed in this paper are described in the following.

Let I be the set of potential locations where facilities can be opened, and J be the set of customers. Assume that $I \cap J = \emptyset$ and that each customer must be assigned to one open facility. Potential connections between facilities and customers are given by arc set $A_J \subseteq I \times J$. Open facilities must be connected to each other using the so-called *core-network* that consists of I and a set K of intermediate nodes. The set $S = I \cup K$ are called Steiner (or core) nodes. The edge set E_S defines potential connections between Steiner nodes, i.e., the core-network. In the following, we assume that the core-network is connected and that the optimal solution contains at least two facilities. Both assumptions are without loss of generality because,

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



(a) Instance of the ConFL.



(b) Feasible solution of the ConFL.

Figure 1: Example of the ConFL with $I = \{i_1, \ldots, i_4\}$, $J = \{j_1, \ldots, j_5\}$ and $K = \{k_1, k_2\}$. The solution is a tree-star structure. Open facilities i_1 and i_4 are indicated in black. Facility i_3 is used for connecting i_1 and i_4 but is not opened.

in the first case, we can solve our problem for each component individually and, in the second case, we can consider all single-facility solutions in a preprocessing phase.

The *connected facility location problem (ConFL)* consists of deciding which facilities from I to open, how to connect them using the edges in E_S and how to assign the customers to open facilities. There are costs associated to opening facilities, connecting them through the core network, and assigning customers to facilities. The aim of the ConFL is to find a minimum-cost solution. Figure 1a shows an instance of the problem and Figure 1b shows a feasible solution.

In the last decade, the ConFL and variants of it have received much attention from the scientific community, both in the areas of Operations Research and Computer Science (see e.g., [8, 9, 20, 28] and the references therein). The ConFL has been independently introduced by Karger and Minkoff [12], Krick et al. [17], Havet and Wennink [10] to model applications arising in information/content distribution networks. It has been also shown that the problem is of importance for applications in emergency management [31] where facilities correspond to distribution centers, demand nodes are cities and the facilities are to be connected over a transportation network. Gollowitzer and Ljubić [8] proposed several mathematical formulations based on direct graphs and compared their linear-programming relaxations. Recently, the problem has been studied from a polyhedral point of view in Leitner et al. [22] (undirected model) and Leitner et al. [21] (rooted directed model). The currently best known approximation ratio for ConFL is 3.19, see [9]. A directed version of ConFL is as hard to approximate as the directed Steiner tree problem (also known as the Steiner arborescene problem, see Section 3). Charikar et al. [3] provide an approximation algorithm for the latter problem with a ratio of $O(k^\epsilon)$, for any fixed $\epsilon > 0$, where k is the number of nodes that need to be connected.

The *Steiner tree-star problem (STS)* has been introduced by Lee et al. [18]. In this problem, we are searching for a tree-star network of minimum cost. All customers from J are required to be leaves, and, besides the costs incurred on the edges, non-negative costs need to paid for every node from S which is taken in the solution. Therefore, each node in S is considered as a potential facility, and the facility opening costs have to be paid even if no customer is attached directly to this node. The problem arises in the design of telecommunications networks for digital data service, where nodes from S correspond to locations where digital switching offices with bridging capabilities (called hubs) need to be installed. For the STS, Lee et al. [19] present a branch-and-cut algorithm based on an undirected graph representation, and provide some facet-defining inequalities. A tabu-search heuristic is given in Xu et al. [30] and a genetic algorithm in Chu et al. [4].

The *generalized Steiner tree-star problem (GSTS)* is an extension of the STS which has been introduced in Khuller and Zhu [13], where the authors present approximation algorithms. In this problem, in contrast to the STS, the facility set I and the customer set J do not need to be disjoint, i.e., $I \cap J \neq \emptyset$ is allowed. By introducing a new node j_i for every $i \in I \cap J$ and a zero-cost edge $\{i, j_i\}$, a GSTS instance can be transformed into a STS instance.

A special case of the GSTS has been addressed in [26, 28], under the name of *rent-or-buy problem (ROB)*. In the ROB, every node in the graph can act as facility node, i.e., $J \subseteq I$, and there are no opening costs for facilities. The current best approximation algorithm for the ROB has a factor of 2.92 and is given in [6].

Bardossy and Raghavan [1] introduced a variant of the ConFL in which the opening costs have to be paid for each facility, no matter if it serves a customer or not. The ConFL, STS, GSTS and ROB are transformed into this variant in Bardossy and Raghavan [1], which is solved heuristically using a primal-dual approach in combination with local search. Their transformation involves increasing the number of nodes and edges. Thus, in particular when larger graphs are considered, the performance of their approach is significantly deteriorated.

Our paper proposes a new approach for finding provably optimal solutions for the family of tree-star problems. The approach is based on a branch-and-cut framework using various Mixed-Integer Programming (MIP) formulations. First, we show that all the considered tree-star problems can be seen as special cases of the asymmetric version of the ConFL, denoted by *aConFL*. Then, we exploit a connection between the dual ascent approach for the Steiner arborescence problem (see, e.g., [27, 29]) and various cut-based formulations for the aConFL. In addition, the approach contains sophisticated reduction techniques and heuristic procedures. In this article we also demonstrate how Benders decomposition can be used to project out the “stars” from the underlying models.

We assess the efficiency of our approach with an extensive computational study using benchmark instances previously proposed in [1, 8]. The implementation solves to optimality all instances from [1] for which the optimal solutions were unknown, and it also gives a considerable speed-up on the instances from [8]. For that reason we also introduce a new set of larger benchmark instances, some of which are solved to optimality. Finally, the construction heuristic used in our approach turns out to be competitive in solution quality, and much faster in runtime, when compared to the heuristic approach in [1].

The rest of the paper is organized as follows. Section 2 defines the aConFL problem and shows how to transform each of the above-considered tree-star problems into the aConFL. In addition, the section also gives integer programming formulations for the aConFL. Section 3 points out the connection between the aConFL and the Steiner arborescence problem, and exploits this relation to build an efficient branch-and-cut algorithm. It details a dual ascent algorithm that provides strong cuts for our branch-and-cut algorithm. The section also describes reduction tests based on reduced costs obtained from the dual ascent algorithm, and introduces construction and primal heuristics. Further implementation details of the framework are also given in Section 3. Section 4 analyzes computational results. Section 5 summarizes the paper.

2 Mathematical Formulations

In this section we first define the aConFL and transform the four tree-star problems mentioned above into it. We then recall directed cut-set models that have been used in the literature (see [8, 21]) and introduce an alternative formulation with a reduced number of decision variables to model the "star"-part of the problem.

2.1 Transformation of the tree-star problems to aConFL

A main difference between the (undirected) ConFL and its asymmetric counterpart is in the structure of the *core graph*, i.e., the subgraph induced by $S = I \cup K$. Let $G = (V, A_c \cup A_J)$ be a directed graph defining an instance of the aConFL problem. Node set V is the disjoint union of core nodes $S = I \cup K$, customers J and an artificial root node 0. Besides arcs connecting two core nodes, the set A_c contains arcs from the root to each potential facility, i.e., $\{(0, i) \mid i \in I\} \subseteq A_J$. Arc set $A_J \subset I \times J$ defines the potential assignments of facilities to customers. Cost functions $c : A_c \cup A_J \rightarrow \mathbb{R}_0^+$ and $f : I \rightarrow \mathbb{R}_0^+$ define costs for connecting two nodes and assigning a customer to a facility, and for opening facilities, respectively.

Every feasible solution to the aConFL is an arborescence rooted at 0 that contains a path from 0 to all open facilities and such that the outdegree of the rootnode 0 is one, i.e., it is connected to precisely one open facility that is the root of the solution resulting from removing the artificial root node. As in the ConFL every customer must be assigned to one open facility. We now detail how to transform the ConFL and the STS to the aConFL. (Recall that the GSTS and the ROB are special cases of the STS.)

Connected Facility Location Problem Let us consider $G = (V, E_S, A_J)$, where $V = S \cup J$, and the cost functions $c' : E_S \cup A_J \rightarrow \mathbb{R}_0^+$ and $f' : I \rightarrow \mathbb{R}_0^+$. It is not difficult to see that the (undirected) ConFL can be straightforwardly translated into the aConFL by bidirecting every core edge $e \in E_S$. The arc costs are defined as

$$\begin{aligned} c_{ss'} = c_{s's} &= c'_e && \forall e = \{s, s'\} \in E_S \\ c_{ij} &= c'_{ij} && \forall (i,j) \in A_J \\ c_{0i} &= 0 && \forall i \in I, \end{aligned}$$

and the facility costs are simply adopted from the ConFL, i.e., $f_i = f'_i$, $\forall i \in I$. Now, it is easy to see that each feasible ConFL solution can be represented as an aConFL solution with the same costs by connecting the artificial root 0 with one arbitrarily chosen open facility and correspondingly directing the core graph solution edges.

Steiner Tree-Star Problem Recall that in the STS (which is defined on the same input as the ConFL), costs have to be paid for every node $i \in S$ in the solution. We therefore consider all nodes $i \in S$ as potential facilities ($S = I$) with associated costs f'_i that need to be paid whenever i belongs to a solution, regardless whether a customer is connected to it or not. Thus, a valid transformation is obtained from setting all facility opening costs f_i of the corresponding aConFL instance to zero and additionally adding the original opening costs f'_i to every arc (s, i) going into i (including the arc from the artificial root node), i.e.,

$$\begin{aligned} c_{si} &= c'_e + f'_i && \forall e = \{s, i\} \in E_S, i \in S \\ c_{0i} &= f'_i && \forall i \in I \\ f_i &= 0 && \forall i \in I. \end{aligned}$$

2.2 MIP Models

To model the aConFL, we use the following set of variables: a binary variable x_a indicates if an arc $a \in A_c$ is in the solution; a binary variable y_s indicates if node $s \in S$ is in the solution; a binary variable z_i indicates if facility $i \in I$ is opened; and a binary variable a_{ij} indicates if customer $j \in J$ is assigned to facility $i \in I$. Let $A = A_c \cup A_J$. For a set $H \subset V$, let $\delta^-(H) := \{(u,v) \in A : u \notin H, v \in H\}$ be the incoming cut-set and $\delta^+(H) := \{(u,v) \in A : u \in H, v \notin H\}$ and be the outgoing cut-set. Let $a(L) = \sum_{(u,v) \in L \cap A_J} a_{uv}$ and $x(L) = \sum_{(u,v) \in L \cap A_c} x_{uv}$ for $L \subseteq A$.

A generic MIP model for aConFL is given as follows:

$$\min \sum_{a \in A_c} c_a x_a + \sum_{i \in I} f_i z_i + \sum_{(i,j) \in A_J} c_{ij} a_{ij} \tag{1}$$

$$(a, z) \in STAR \tag{2}$$

$$(x, y) \in TREE \tag{3}$$

$$z_i \leq y_i \qquad \forall i \in I \tag{4}$$

$$(x, y, z, a) \in \{0,1\}^{|A_c|+|S|+|I|+|A_J|} \tag{5}$$

where (2) is a generic way to express that the variables (a, z) shall model “stars”, i.e., the facility location part of the solution, and (3) is a generic way to express that (x, y) variables model the “tree” part, i.e., the Steiner arboresence in the core network. The precise formulations of (2) and (3) are given below. Connection between the tree and the stars is given through the coupling constraints (4) that make sure that each open facility must be part of the arborescence.

Modeling the “Stars” (2) To force that (a, z) represents stars, one can consider the standard facility location model:

$$a(\delta^-(j)) = 1 \qquad \forall j \in J \tag{6}$$

$$a_{ij} \leq z_i \qquad \forall i \in I, \forall j \in J \tag{7}$$

which ensures that every customer is assigned to exactly one open facility.

Alternatively, one may reduce the number of decision variables by an order of magnitude. To this end, replace the $\sum_{(i,j)\in A_J} c_{ij}a_{ij}$ in the objective function by $\sum_{j\in J} w_j$, where w_j are decision variables modeling the allocation cost of customer $j \in J$. Thereby, the facility location part can be modeled as follows (see, e.g., [7]):

$$w_j + \sum_{k\in I}(c_{ij} - c_{kj})^+ z_i \geq c_{ij} \qquad \forall i \in I, \forall j \in J \tag{8}$$

$$\sum_{(i,j)\in A_J} z_i \geq 1 \qquad \forall j \in J \tag{9}$$

where $\alpha^+ = \max\{0, \alpha\}$. Constraints (9) ensure that for every customer, at least one facility, where it can be assigned to is opened. Note that these constraints reduce to $z(I) \geq 1$, for all $j \in J$ if A_J is a complete bipartite graph. This model can also be seen as a Benders reformulation of the facility location problem. In Benders' decomposition terminology, (8) are *optimality cuts* and (9) are *feasibility cuts*. See, e.g., [7] for more details. Observe that in this model we trade off a quadratic number of assignment variables by the quadratic number of constraints. The quality of linear programming (LP) relaxation bounds remains the same.

Modeling the “Trees” (3) To force that (x, y) represents a Steiner arborescence, one can use the following constraints:

$$x(\delta^-(s)) = y_s \qquad \forall s \in S \tag{10}$$

$$x(\delta^-(H)) \geq y_s \qquad \forall H \subseteq S, \forall s \in H \tag{11}$$

$$x(\delta^+(0)) = 1. \tag{12}$$

Inequalities (11) are directed cut-set constraints, ensuring that there is a path-from the root node to every node in the solution. The cut-set constraints (11) will be referred to as *ycuts* in the following. Constraint (12) ensures that only a single outgoing arc from the root node is selected. Due to the non-negative arc costs, there always exists an optimal solution where only opened facility nodes are leaf-nodes of the “tree” part. Thus it is enough to consider *ycuts* only for y_i, $i \in I$. Note that Gollowitzer and Ljubić [8] used z_i instead of y_i on the right-hand-side of (11), but it is easy to see that (11) dominate them due to (4). Furthermore, constraints (4), (7), (11) are facet-defining (see Leitner et al. [21]).

Aside from *ycuts*, there is also another family of cut-set constraints, which ensure connectivity of a solution. These constraints will be denoted by *acuts* and look as follows:

$$x(\delta^-(H)) + a(\delta^-(H)) \geq 1 \quad \forall H \subset V : |H \cap J| = 1. \tag{13}$$

Constraints (13) ensure that there is a directed path from the root to each customer, instead to the opened facilities only. Gollowitzer and Ljubić [8] showed that replacing *ycuts* with *acuts* leads to a stronger

LP-relaxation. Leitner et al. [21] showed that these inequalities are facet-defining. We refer to the model (1),(4)–(7),(10),(11) as a *Weaker Cut-Set Formulation*, and to the model (1),(4)–(7),(10),(13) with *acuts* instead of *ycuts* as *Strong Cut-Set Formulation.*

Valid Inequalities It is well-known that the so-called flow-balance inequalities

$$x(\delta^-(s)) \leq x(\delta^+(s)) \quad \forall s \in K \tag{14}$$

can improve the LP-bound for Steiner tree problems, see, e.g., [14, 24]. These inequalities use the fact that, due to non-negative arcs weights, there is always an optimal solution having no node from K as a leaf-node. The flow-balance inequalities are also valid for the (strong) directed cut-set formulation of aConFL. Moreover, one can also define such inequalities for facility nodes; in this case, they look as follows:

$$x(\delta^-(i)) \leq x(\delta^+(i)) + z_i \quad \forall i \in I. \tag{15}$$

Correctness of (15) follows the same arguments as for (14), i.e., only nodes where a facility is opened can be leaf-nodes in a solution.

3 Methods from the Steiner Arborescence Problem

Both *acuts* (13) and *ycuts* (11) presented above can be separated in polynomial time using a max-flow algorithm. In practice, however, separation of *acuts* is much more time-consuming than the separation of *ycuts* since *acuts* need to be separated for every customer. The separation of *ycuts* (and also *zcuts*) only needs to be done for every potential facility whose associated y-variable (z-variable) has a positive value in the current LP-solution (or an LP-value greater than some threshold). Moreover, the separation graph for *acuts* is the whole graph, while for *ycuts* it is only the core graph. Thus Gollowitzer and Ljubić [8] concluded that the *acuts* are not too useful in practice. Similar conclusions were drawn in [21], where for small scale instances using *acuts* is (considerably) faster, but for medium/large-scale instances the separation time becomes prohibitive.

In this section, we show how to overcome this drawback of *acuts* by exploiting the connection between the aConFL and the Steiner arborescence problem (SAP). A quick dual ascent procedure applied to the latter problem provides a useful source of strong cutting planes (*acuts*) that we use to initialize the branch-and-cut algorithm and to reduce the size of the input graph.

There is a connection between the strong cut-set formulation for the aConFL and the SAP. More precisely, the strong cut-set formulation can be transformed in an equivalent cut-set formulation of the SAP (with root-outgoing constraint (12)). The transformation works by splitting every facility node i into two nodes i and i'. Every assignment arc (i, j) is replaced by an arc (i', j) with the same cost. Moreover, an arc (i, i') with costs f_i is introduced. The set J becomes the terminal set of the SAP. It is easy to see that the terminal nodes can only be reached by using arcs from (i, i'), which encode the opening of facilities.

Let $G_T = (V_T, A_T)$ be the graph of an instance after transformation. We have $V_T = K \cup I' \cup J$, where I' are the split facility nodes, and $A_T = A_c \cup A'_J \cup A_{II'}$, where A'_J are A_J with nodes in I replaced by the new nodes in I' and $A_{II'}$ are the new arcs for the split facility nodes. Let x_a be a binary variable assuming value one iff arc $a \in A_T$ is in the solution. For a terminal $j \in J$, let $\mathcal{H}_j = \{H \subset V_T : H \cap J = \{j\}, 0 \notin H\}$ be the set of nodes inducing Steiner connectivity cuts, and let $\mathcal{H} = \bigcup_{j \in J} \mathcal{H}_j$. We obtain the following SAP model with an additional root-outgoing constraint. For easier exposition of the dual ascent algorithm, the model is stated in "$\geq$"-form.

$$\min \sum_{a \in A_T} c_a x_a \tag{16}$$

$$x(\delta^-(H)) \geq 1 \qquad \forall H \in \mathcal{H} \tag{17}$$

$$-x(\delta^+(0)) \geq -1 \tag{18}$$

$$x \in \{0,1\}^{|A_T|} \tag{19}$$

Connectivity cuts (17) ensure that every customer is connected to the root node, while constraint (18) ensures, that at most one arc going out from the root node is taken in a solution.

Connectivity cuts (17) can be easily transformed into *acuts*: For every arc a in a given constraint (17), which corresponds to an arc in A_c or A_J, use the appropriate variable x_a, resp. a_a. This leaves us with arcs from $A_{II'}$. Let (i,i') be such an arc. Note that by construction, this arc is the only incoming connection to i'. The only connection from i' to j is the arc (i',j). Thus, whenever (i,i') is in a solution, (i',j) must also be in a solution, and we can use the a-variable corresponding to this arc (i',j) for such arcs $(i,i') \in A_{II'}$.

3.1 Generating *acuts* using a Dual Ascent Algorithm

It is well-known that, for the SAP, the dual ascent (DA) algorithm first proposed by Wong [29] (using a multi-commodity flow formulation as the primal problem) usually provides good lower bounds. A basic idea of a DA algorithm is to create a feasible dual solution whose value provides a valid lower bound to the problem. Constraints of the dual problem that become saturated (tight) upon termination, provide a useful information about the arcs of the input graph that can be used to build a feasible solution. The graph induced by all saturated arcs is usually referred to as *support graph*.

Contrary to the approach proposed in [1] (which relies on a DA to derive heuristic solutions), our main goal is to exploit the connection to the DA for the SAP in the context of an efficient exact approach. More precisely, our aim is to (i) derive connectivity cuts *acuts* in an efficient way and (ii) make powerful reductions based on reduced costs. Note that our description of the DA algorithm follows [27] instead of [29] since [27] uses the directed cut model as the primal problem, which makes the connection to the obtained cuts more clear.

Let β_H be the dual variables of constraints (17) and λ be the dual variable of constraint (18). The dual problem of the LP-relaxation of (16)-(19) is

$$\max \sum_{H\in\mathcal{H}} \beta_H - \lambda \tag{20}$$

$$\sum_{H\in\mathcal{H}:(s,s')\in H} \beta_H \le c_{ss'} \qquad \forall (s,s') \in A_T, i \neq 0 \tag{21}$$

$$\sum_{H\in\mathcal{H}:(0,i)\in H} \beta_H - \lambda \le 0 \qquad \forall (0,i) \in A_T \tag{22}$$

$$\beta_H \ge 0 \qquad \forall H \in \mathcal{H} \tag{23}$$

$$\lambda \ge 0. \tag{24}$$

In the dual problem of the LP-relaxation of the classical SAP (see [27]) there would only be constraints of type (21). Due to the root-outgoing constraint, the dual constraints (22) of root-outgoing arcs (which have $c_{0i} = 0$ by definition) do, however, also contain the dual multiplier λ. For a fixed λ, say $\bar{\lambda}$, constraints (22) are similar to (21), with costs $\bar{\lambda}$ on the right hand side. Thus, the DA algorithm for tree-star problems needs an adaptation to handle the additional dual multiplier λ.

The DA algorithm for SAP in [27] works as follows: Let $\bar{\beta}_H$ be the values of the dual variables and $\bar{c}_{ss'} := c_{ss'} - \sum_{H\in\mathcal{H}:(s,s')\in H} \bar{\beta}_H$ be the reduced costs of the arcs. Let *Sup* be the arc-set of the support graph, which is defined as set of all *saturated arcs*, i.e., arcs with reduced costs equal to zero. Let *lower* be the lower bound value.

Step 1. Initialize *lower* and $\bar{\beta}_H$ with zero, consequently, we have $\bar{c}_{ss'} := c_{ss'}$ and $Sup = \emptyset$.

Step 2. As long as there is a terminal $j \in J$, for which there is no directed path from 0 to j in the subgraph induced by *Sup* do the following:

(a) Choose a terminal j not connected to the root in *Sup* and let H_j be the set of nodes, which are reachable from j by a reverse breadth-first search (bfs) in *Sup*. Note that H_j induces a connectivity cut (17) and we thus save it for later use in our branch-and-cut algorithm (after transforming it to an *acut*).

(b) Let $\Delta = \{\bar{c}_{sk} : (s,k) \in \delta^-(H_j)\}$ be the minimum reduced costs of all arcs in the connectivity cut induced by H_j.

(c) Set $\bar{c}_{sk} = \bar{c}_{sk} - \Delta$ for all $(s,k) \in \delta^-(H_j)$. By definition of Δ, some reduced costs decrease to zero and the associated arcs now enter *Sup*

(d) Increase *lower* by Δ and go back to (a).

Upon termination of the algorithm, the support-graph *Sup* contains a directed path from the root node to every terminal. However, for the aConFL, depending on the value of $\bar{\lambda}$, the support-graph may contain more than one of the arcs $(0,i)$, and would therefore not contain a feasible solution. For example, with $\bar{\lambda} = 0$, *Sup* would already contain all arcs $(0,i)$, and the dual ascent would only be concerned with the facility location part of the problem. On the other hand, setting $\bar{\lambda} = M$, where M is a big value, ensures that only one of the arcs $(0,i)$ is taken. This can be easily verified by modeling (18) with a big-M coefficient in the objective, instead of adding it explicitly as constraint. The obtained dual problem is exactly the same as setting $\bar{\lambda} = M$. Moreover, increasing the value of M does not deteriorate the quality of the lower bound obtained by the DA algorithm (i.e., the lower bound is monotonically non-decreasing in M). This is because both variables β_H and λ are contained in the objective function and in constraints (22). Therefore, some β_H can be increased for any increase of λ.

Bardossy and Raghavan [1] (who also provide a Lagrangian viewpoint on this topic) set M to $|S| \cdot \max_{a \in A_T} c_a$, since no path from the root to a terminal can be longer than this value. Whereas a too large value of M does not influence the final lower bound, it has a negative effect on (i) the runtime of the dual ascent, (ii) on the reduced costs and (iii) the overall number of detected connectivity cuts. Observe that for a too large value of M, the value of Δ in Step 2(b) will be determined by non-root arcs until the very end. Hence, the smaller (but still sufficiently large) the value of M is, the earlier a root arc becomes tight in Step 2, and the earlier the DA algorithm terminates. Consequently, fewer cuts are generated and the reduced costs $\bar{c}_{ss'}$ are higher.

To define the value of M in our case, we use the result of the following proposition.

Proposition 1. *The optimal dual multiplier λ in* (16)-(19) *is lower than or equal to the maximum length of a shortest path between two facility nodes $mSPF$.*

Proof. This follows from a sensitivity analysis of (16)-(19) in combination with the combinatorial aspects of the problem. Multiplier λ is the dual multiplier of the root-outgoing constraint (18), so it signalizes how much the solution could potentially be improved by having two root-outgoing arcs instead of one (i.e., decreasing the left hand side from -1 to -2). If in any feasible solution two facilities, instead of one, can be connected to the root-node (which does not induce any cost in the objective), the cost for connecting the facility with the most expensive connection cost could be saved. The maximum length of all shortest paths between two facility nodes gives an upper bound for this saving. □

The following result which will be used to construct a heuristic solution in Section 3.3 provides a value for $\bar{\lambda}$ that guarantees that the support graph contains a feasible solution.

Proposition 2. *Using $\bar{\lambda} \geq mSPF$ results in a support-graph containing a directed path from at least one $i \in I$ to all terminals $j \in J$.*

Proof. Upon the termination of the dual ascent, there is a directed path from the root 0 to all terminals $j \in J$ in the support-graph. If the support-graph only contains a single-arc $(0,i)$, then clearly, there is a directed path from i to every $j \in J$. Thus, consider the case that the support-graph contains two arcs $(0,i')$ and $(0,i'')$. Note that for each $j \in J$, this means there is a directed path to j from either i' or i''. By definition of the dual ascent, $(0,i)$ appears in any cut induced by H with $i \in H$. Thus, if $(0,i)$ is in the support-graph, an amount of $mSPF$ has been pushed into cuts going to i. This in turn means that the support-graph contains a path from any node k to i whose distance from i is at most $mSPF$. Hence, by definition of $mSPF$, when $(0,i)$ is contained in the support graph, it also contains a path from any other facility node to i. Consequently, there is a path from i' to i'' and a path from i'' to i' in the support-graph. Thus, from both i' and i'', all $j \in J$ can be reached. The proof for more than two root-arcs in the support-graph works analogously. $\square$

In view of the above propositions, we set $\bar{\lambda} = mSPF$ in our approach.

3.2 Bound-Based Reductions

Reductions using the lower bound and reduced costs provided by the DA algorithm have been successfully used in Polzin and Daneshmand [27] for the Steiner tree problem (modeled as SAP). They are based on the following observation.

Proposition 3. *[27] Let $G_T = (V_T, A_T)$ with cost function c be an instance of the SAP, let $\tilde{c} \leq c$ and let LB be a lower bound for the optimal solution of the SAP in $G_T = (V_T, A_T)$ with cost function $c' = c - \tilde{c}$. For any feasible Steiner arborescence $\bar{x}$, it holds $LB + \tilde{c}^T \bar{x} \leq c^T \bar{x}$.*

Since the dual solution obtained by the DA algorithm with cost function c stays feasible for cost function $c' = c - \tilde{c}$, one can exploit the result of Proposition 3 by using the lower bound *lower* as LB and the reduced costs $\tilde{c}$ produced by the DA algorithm. Consequently, Proposition (3) enables one to remove (resp., fix the associated variables to zero) nodes and arcs from an instance in the Steiner arborescence case. In particular, suppose we are given a feasible solution with objective value *obj*. Clearly, a solution $\bar{x}$ can only be optimal if $c^T \bar{x} \leq obj$, and from Proposition 3 and the discussion above it follows that $\tilde{c}^T \bar{x} \leq obj - lower$ must hold, i.e., the cost of Steiner arborescence $\bar{x}$ evaluated under the reduced costs $\tilde{c}$ must not be larger than $obj - lower$.

We now illustrate how an arc (s, s') can be fixed to zero using this result, the procedure for eliminating Steiner nodes follows similar arguments. Let $opt((s, s'))$ be the optimal value of a Steiner arborescence with cost function $\tilde{c}$ containing this arc. Clearly, if $opt((s, s')) > obj - lower$, (s, s') cannot be in an optimal solution, and thus can be fixed to zero. Finally, this check remains valid even when the value of $opt((s, s'))$ is underestimated by the shortest path cost from the root to s plus the shortest path cost from s' to any terminal and the cost of (s, s'). The correctness of this claim follows from the fact that there always exists an optimal solution such that the only leaves in this solution are terminal nodes. This fact has also been used in deriving valid inequalities (14) and (15).

For the elimination of potential facility nodes, recall that in our transformation facilities are modeled as arcs (i, i') in the SAP. Therefore, the reduction described above is also valid for the removal of facilities from an instance. Note that, if a facility i is "removed", the status of the underlying node i switches from I to K.

3.3 SAP-Based Construction and Primal Heuristic

The connection between aConFL and SAP can also be used in the design of construction and primal heuristics that are essential for the fast convergence of an efficient branch-and-cut framework. Moreover, the construction heuristic is also essential for the bound-based reductions of the previous sections, which need good upper bounds to be effective.

Let (x^*, y^*, z^*, a^*) be a solution of the LP-relaxation of the (strong) directed cut-set formulation. The primal heuristic of Gollowitzer and Ljubić [8] consists of choosing a threshold value z_T for z^* and considering every $i \in I : z_i^* \geq z_T$ as open facility. A solution is then constructed by first finding the minimum cost customer assignment possible for these opened facilities. Then, all open facilities are considered as terminals for the application of the MST-Steiner tree heuristic of [25]. Finally, a local search phase is done trying to close unnecessarily opened facilities and removing the no-longer needed connections to them. Thus, the whole procedure consists of three phases, a threshold value z_T must be chosen before and the LP-values of x_a^* as well as the facility opening costs are ignored.

We use a different heuristic obtained by modifying the PrimI heuristic described in [5] for the Steiner tree problem which is a combination of Dijkstra's shortest path algorithm and Prim's spanning tree algorithm. It starts with a partial tree T consisting only of the root node and shortest paths are calculated from T to all terminals (customers J in our case). A terminal with a cheapest shortest path is selected and added to

T, together with all nodes lying on the path. This process is repeated until all terminals are connected. To deal with the root-outgoing constraint one could simply run it for all facilities as root node.

There is, however, an undesired effect when directly using the PrimI heuristic on our transformed instances, due to their special structure. Suppose the first terminal (i.e., customer) j is found by the heuristic. Thus, the path from the root to j is added to T. Let i (in the form of the arc (i, i')) be the facility used in this connection. Since connection cost are calculated with respect to T, the costs for connecting any other customer j' to facility i is only $c_{ij'}$ while the cost to add it using another (still unopened) facility, say i'' consists of the connection cost of this other facility in the core, the opening cost of i'', and the assignment cost of arc (i'', j'). This leads to the undesirable effect that solutions constructed by the direct application of the PrimI heuristic on the transformed instances often contain only one open facility and have a rather poor quality. To deal with this problem, we propose a modification to obtain a construction heuristic and a primal heuristic which are explained below.

Construction Heuristic Algorithm 1 gives a pseudo-code for the construction of an initial heuristic solution; note that we try every $i \in I$ as root node.

```
Data: aConFL instance and root node $r \in I$
Result: Feasible solution to aConFL containing $r$
1  $T \leftarrow r$
2  $z^* \leftarrow \infty$
3  $\forall v \in V : distance(v) \leftarrow \infty$
4  $\forall v \in V : pred(v) \leftarrow null$
5  $distance(r) \leftarrow 0$
6  $pQ \leftarrow (r, distance(r))$ // priority queue, number is priority
7  while $pQ \neq \emptyset$ do
8      $v \rightarrow$ pop first element from $pQ$
9      $openFacility \leftarrow false$
10     if $v \in I$ then
11         $assignment \leftarrow$ best assignment of customers using open facilities in $T$ and $v$
12         if $cost(T, path\ to\ v, assignment) < z^*$ then
13             try to improve $assignment$ by closing a facility
14             $openFacility \leftarrow true$
15     if $openFacility = false$ or $v \in T$ then
16         for $(v, w) \in A_c$ do
17             if $distance(w) > distance(v) + c_{vw}$ then
18                 $distance(w) \leftarrow distance(v) + c_{vw}$
19                 $pred(w) \leftarrow v$
20                 push $(w, distance(w))$ to $pQ$
21     else
22         open $v$ and add $v$ and the path connecting to it to $T$ // path is recoverable using $pred$
23         set distance to zero for $v$ and all nodes on the path to it
24         push $v$ and all nodes on the path to it to $pQ$ with priority zero
25         update $z^*$ using $assignment$ and cost of the path to $v$
26 postprocesing to remove leaf-nodes in $T$, which are not opened facilities
```

Algorithm 1: Construction Heuristic

The undesired effect of PrimI on the transformed instances comes from the fact that it is "too greedy", as it only considers one customer at a time. When looking just at a single customer opening a new facility is most likely not done, since we would have to pay the connection cost to the new facility, opening cost and assignment cost. Instead for connecting it to the already opened facility, we only have to pay the assignment cost. However, when looking into a "cluster" of customers, this situation changes since the costs for opening a new facility are then shared among the customers in the cluster.

Our *construction heuristic* which attempts to incorporate this idea into the PrimI heuristic works as follows: We label the nodes in the core-network in which the facility nodes are considered as potential Steiner tree terminals. Whenever we encounter a new facility node i, we find the best possible assignment of customers using this facility and the open facilities from T. If this results in a cheaper solution than the current one, i is added to T, considered as a terminal and opened, and customers are respectively re-assigned. If adding i to T does not improve the current solution, i is considered to be a potential Steiner node and remains closed. Its shortest path distance label is updated and further exploited in the same fashion as in the PrimI heuristic algorithm. Observe that the re-assignment of customers may result in closing some of facilities being opened in previous iterations. Moreover, once a facility is added, we also check if we can get an improved solution by closing one of the already opened facilities. In that case, we close the facility which brings the largest improvement. We also experimented with closing more facilities in this step, but decided to close a single facility after preliminary computations. Closed facilities are kept in T and consequently, facilities from T may turn out unnecessary later. Hence we apply a postprocessing as in Gollowitzer and Ljubić [8] after termination of the heuristic, where all leaf-nodes from T, which are not opened facilities, are removed. If the assignment graph is not complete bipartite, we also add the encountered facility i, if opening it allows the connection of a previously unconnected customer j. In preliminary computational experiments, it turned out to be beneficial to run the construction heuristic not on the original instance, but on support-graph produced by the DA algorithm. In this case, after running the heuristic for an $i \in I$ as root, we check, if all customer are connected in the constructed solution, and discard infeasible solutions.

Primal Heuristic Similar to Gollowitzer and Ljubić [8], we first build the assignment part. However, we do not fix the threshold value z^T, but run the complete algorithm for all potential threshold values for given LP-solution values z^*. Once we have found the best assignment, we set the opened facilities in this assignment as terminals, and use the PrimI-heuristic, using $c_{ij}(1 - x^*_{ij})$ as weight. Thus, in contrast to Gollowitzer and Ljubić [8], our heuristic is influenced by the LP-values of the core-part and we avoid the need to apply a post-processing procedure closing unnecessarily opened facilities. We also tried our construction heuristic as primal heuristic, using the LP-values to set the weights. However, the current primal heuristic worked slightly better in our computational experiments.

3.4 Implementation Details

To conclude this section, we describe the implementation details of our solution framework.

Bound-based Reductions and Variable Fixing Before starting the branch-and-cut algorithm, we repeatedly call the DA algorithm and the construction heuristic to perform the bound-based reductions, until no more arcs/nodes/facilities can be removed. The construction heuristic is run on the support graph produced by the dual ascent.

We also use the dual ascent lower bounds and reduced costs to fix variables within the branch-and-cut framework. While modern branch-and-cut solvers do reduced cost based variable fixing on their own, in our case, the fixing is not based on the reduced cost of a single variable, but on more elaborate arguments as described in Section 3.1.

Separation of Inequalities Since there is an exponential number of *ycuts* and *acuts*, we separate them on-the-fly during the solution process, i.e., we use branch-and-cut. Benders optimality cuts (8) are also separated on-the-fly. Although they are of polynomial size, their separation has proven computationally advantageous in preliminary tests.

The separation of inequalities *ycuts* and *acuts* is done using a max-flow algorithm on a graph where the capacities of arcs are given by the current values of the LP-solution, i.e., for *ycuts*, we consider the support graph with capacities x^*, and for *acuts*, we consider the complete graph of an instance, where in addition to x^* for the core graph, a^* is used as capacity of the assignment arcs. We have implemented *backcuts, nested cuts* and *minimum cardinality cuts*, see, e.g., [14, 24] for more details. In case we encounter an integer solution during separation, we simply do a breadth-first-search to check connectivity instead of using max-flow.

A cut-pool is used to speed up the separation of inequalities *acuts*. The idea is the following: Given an LP-solution, the separation of *acuts* needs to be done for every customer (some preliminary experiments in which *acuts* are separated only for a few customers in case of fractional LP-solutions, resulted in a very weak performance). However, for large-scale-instances a significant slowdown of the branch-and-cut performance can be observed when *acuts* are separated instead of *ycuts*. This can be explained by the fact that *ycuts* are only separated for facilities with $z_i^* > 0$ (or some threshold value). The number of such facilities in an LP-solution is typically much smaller than the overall number of customers, and consequently much fewer max-flow iterations are required when considering *ycuts* instead of *acuts*. This small number of facilities with $z_i^* > 0$, however, also influences the structure of the *acuts* for the given LP-solution. They are likely to be very similar, since there are not many open facilities, where customers can be connected to. We try to exploit this in a heuristic scheme to generate *acuts*: For a given and violated *acut* to a customer j, we try to construct a violated *acut* for another customer j' by simply replacing all the a_{ij} in the cut with $a_{ij'}$ and leaving the remaining cut unchanged. In particular, we store all *acuts* separated by max-flow for a given LP-solution so far in a cut-pool. Before we call the (expensive) max-flow separation for the next customer j', we try to construct a violated *acut* using the described procedure applied to the cuts in the pool.

When separating *ycuts*, we check if all facilities are on the sink-side, and add a cut with one instead of y_i on the right-hand-side in that case. We take into account the variable fixing when checking, i.e., if a facility is fixed to zero, it does not matter, if it is on the source or sink side.

Tailing Off Preliminary computational tests have shown a slow convergence of the cut-loop, i.e., many violated cuts have been detected and inserted into the LP, with a very minor contribution in improving the quality of lower bound. This effect was especially pronounced for *acuts*. To tackle this issue, we used a two-fold strategy: We stop the cut-loop if for five consecutive iterations the absolute increase in lower bound value from one iteration to the next is < 0.001. Moreover, we use a parameter ρ to limit the overall number of cut-loop iterations within a branch-and-cut node. In our experiments, we have $\rho = 100$ for the root node and $\rho = 20$ for the other nodes. These settings only consider separation of fractional points, whereas for integer points encountered inside of the branch-and-cut tree, the separation procedures are always performed, in order to ensure the overall correctness of the algorithm.

Branching Priorities As already shown in [8], it is effective to set custom branching priorities taking into account the structure of the problem, instead of letting the MIP-solver decide. Similar to [8], we set the highest branching priority to z-variables succeeded by y-variables (note that [8] had no y-variables). The idea is that choosing which facility to open and which nodes to be in the solution is much more important than choosing a single arc. This is motivated by the observation that once the open facilities and nodes in the solution are selected, the problem reduces to a spanning tree problem in the core and to a cheapest assignment of the customers to open facilities.

4 Computational Results

The main purpose of our computational study was to compare the three basic formulations provided in Section 2, and to investigate the effects of the DA-based cut-generation, bound-based reductions and the role of heuristics to the overall performance of each of the basic settings. We compare our results against three state-of-the-art approaches from the literature:

- two (slighly improved) re-implementations of branch-and-cut approaches used in [8], based on *ycuts* and *acuts*, respectively (denoted as settings **Y** and **A**, respectively), and

- a heuristic of Bardossy and Raghavan [1], for which the authors have shown that it outperforms all other heuristic approaches available in the literature.

Settings Considered in the Computational Study In Table 1, all parameter settings considered in this computational study are detailed. The three basic settings are **Y**, **A** and **B**. The former two correspond to the branch-and-cut approaches used in [8], based on *ycuts* and *acuts*, respectively. The setting **B** is a Benders-like branch-and-cut in which *ycuts* are separated along with the Benders optimality cuts (8) that are used to model the "stars". Whenever a basic setting is enhanced by the generation of *acuts* or application of the bound-based reductions in combination with the construction heuristic (due to the DA procedure), this is denoted by **Da** and **R**, respectively.

In each of the settings, we use the branching priorities and the primal heuristic procedure described below. As initialization, we add $x_{ss'} + x_{s's} \leq y_s, \forall s, s' \in K$, which is a subset of (rewritten) *ycuts*. Moreover, we also add all inequalities (14) and (15).

Table 1: Overview on used settings in our computational study

name	model	DA-cuts	reductions	construction heur.
Y	Weak Cut-Set	×	×	×
A	Strong Cut-Set	×	×	×
B	Benders	×	×	×
YDaR	Weak Cut-Set	✓	✓	✓
ADaR	Strong Cut-Set	✓	✓	✓
BR	Benders	×	✓	✓
YDa	Weak Cut-Set	✓	×	✓
YR	Weak Cut-Set	×	✓	✓

While the settings **YDaR** and **ADaR** use the full power of our framework, we only consider **BR**, but not **BDaR**. The main reason for this distinction is because the DA-cuts are *acuts*, and thus they are not directly applicable in the Benders approach (in which a variables do not exist and are aggregated into customer allocation cost-variables w_j). Settings **YR** and **YDa** are used for demonstrating the influence of the single ingredients of our framework.

In addition, we also analyzed the performance of our construction heuristic and compared it with the heuristic of Bardossy and Raghavan [1]. In the following, setting **H** denotes the construction heuristic used in conjunction with the support-graph and bound-based reductions. Note that using **H**, we also provide valid lower bounds, due to the application of the DA algorithm.

4.1 Benchmark Instances

Bardossy and Raghavan [1] created instances for all four problems considered in this work, namely ConFL, STS, GSTS and ROB. We have evaluated our algorithms on these instances. Since these instances turned out to be easily solvable by our new algorithmic framework (see Table 3), we created two additional sets of larger instances following the procedure given in Bardossy and Raghavan [1]. These instances are available for download at `http://homepage.univie.ac.at/markus.sinnl/program-codes/confl/`.

The instance-generation procedure works as follows: Complete Euclidean graphs are obtained from randomly selecting nodes in a X by X grid, and rounding up the distance between them to the next integer to obtain the basic edge costs. Such calculated edge costs are considered as assignment-costs, whereas the edge-costs in the core-network are obtained by multiplying the distances by a factor M. Facility opening costs f are set to a constant value. The number of potential facilities and customers in any instance sums up to $|V|$, and different instances are created by taking $10\%, 20\%, \dots, 90\%$ of V as facility nodes. Each set contains ten instances for a fixed number of facility/customer nodes and combination of M and f. This results in 360 instances for each set and problem (Set1, Set2 for ConFL, STS, GSTS and Set3 for ROB). For ConFL, 20% intermediate nodes are added. Table 2 gives the detailed parameter settings used to obtain the instance sets. Note that, for ConFL, Bardossy and Raghavan [1] created two additional sets (Set4 and Set5) that comprise of sparser graphs. The graph density is specified for the edges of the core-graph and for the assignment edges (denoted by $\delta(E_S)$ and $\delta(A_J)$ in the table, respectively).

Table 2: Overview on instances from/based on Bardossy and Raghavan [1]. Set1 to Set5 are from Bardossy and Raghavan [1]; NewSet1 and NewSet2 are newly created larger version of Set1 and Set2. Note that there are 360 instances for each set and problem.

set	problems	X	$\|V\|$	M	f	$\delta(E_S)$ [%]	$\delta(A_J)$ [%]
Set1	ConFL, (G)STS	100	100	3	0, 10, 20, 30	100	100
Set2	ConFL, (G)STS	100	100	1, 3, 5, 7	30	100	100
Set3	ROB	100	100	1, 3, 5, 7	0	100	100
Set4	ConFL	100	100	7	30	25, 50, 75	25, 50, 75
Set5	ConFL	100	100	7	30	25, 50, 75	100
NewSet1	ConFL, (G)STS	150	500	3	0, 10, 20, 30	100	100
NewSet2	ConFL, (G)STS	150	500	1, 3, 5, 7	30	100	100

In addition, we consider ConFL instances proposed by Ljubić [23] and used in Gollowitzer and Ljubić [8]. They are constructed by combining Steiner tree instances with uncapacitated facility location (UFL) instances. The Steiner tree parts, which form the core graph, are from the sets `C, D` of ORLIB [2] and comprise instances with up 500 to 1000 nodes and up to 25000 edges. They are then combined with different instances from Ufllib [11], which form the assignment part to give the following sets of instances. Note that the results reported in Gollowitzer and Ljubić [8] are for a fixed root node, i.e., a node from the instance has been taken as root node and no artificial root was introduced. The selected root node consisted of the facility with the smallest index. We followed this procedure in our computational tests, by modifying our framework accordingly (e.g., since the fixed root-node is part of the instance, the root-outgoing constraint (12) needs to be removed). In these instances, the facility location part is dominating in the cost structure. The core edge costs are integer values from the interval $[1, 10]$.

- **Set MPQ:** The UFL instances are originally from [16]. Facility opening costs and assignment costs are rational numbers from $[100, 600]$ and $[2, 10]$, respectively. Two instances of the two classes `MP` and `MQ` with sizes $|I| = |J| = 200$, respectively $|I| = |J| = 300$ have been taken.
- **Set GS:** The UFL instances are originally from [15]. Facility opening costs are integers from $[100, 200]$ and assignment costs are integers from $[1000, 2000]$. Two instances of the two classes `gs250` and `gs500` with sizes $|I| = |J| = 250$ and $|I| = |J| = 500$ have been taken. Note that for the UFL, these instances are very hard/unsolved for state-of-the-art approaches, see, e.g., [7]. In [8], results are only reported for instances based on `C` Steiner tree instances.

4.2 Results for instances of/based on Bardossy and Raghavan [1]

In this and in the following section, we report the computational results obtained by our approaches. Each experiment has been performed on a single core of an Intel E5-2670v2 with 2.5GHz and at most 3GB RAM memory was given to each run. The algorithms were implemented in C++ and CPLEX 12.6.1 was used as branch-and-cut framework and all CPLEX parameters (except for restricting each run to a single-thread) were left at their default values. A time limit of 900 seconds was set for the runs whose results are reported in this section.

Table 3 contains results for the instances from [1] aggregated by set. In this table (and the following ones), we report the average runtime in seconds ($t[s]$), the average number of nodes in the branch-and-bound tree ($n.$), and the average optimality gap (gap), calculated as $(OBJ - LB)/OBJ \cdot 100$, where OBJ is the value of the best solution found and LB is the lower bound.

The obtained results confirm the importance of incorporating the two new ingredients, namely the dual ascent-based cuts and the bound-based reductions, in improving the algorithmic performance. The overall best performance is obtained by the setting **ADaR** (followed by **YDaR**): using *acuts* obtained by the dual ascent (in combination with bound-based reductions) enables us to solve all the instances of all five sets to provable optimality. In most of the cases, the optimal solution is found already at the root node of the branch-and-cut tree in under three seconds of average computing time. On the contrary, settings **Y** and **B**, which are both based on *ycuts*, fail to solve most of the instances within the given time limit, whereas setting **A** requires one to two orders of magnitude larger computing times.

The results also illustrate the difference in the strength of *acuts* against *ycuts*. Settings involving *acuts* (which are **A**, **ADaR**, but also **YDaR** – recall that **YDaR** separates *ycuts*, but it is initialized with the dual ascent cuts, which are *acuts*) solve all the instances to provable optimality, whereas the settings **Y**, **B** and **BR** fail to do so.

Comparing **YDaR** with **A**, we notice that **A** has longer computing times, but requires a smaller number of branch-and-bound nodes. This is an indicator that *acuts* provide much stronger bounds than *ycuts*, but are very expensive to separate, as long as their generation is not enhanced using the dual ascent. Moreover, setting **A** is much slower when solving the GSTS and the ROB than STS and ConFL. This can be explained by the fact that for the GSTS and the ROB, for each $i \in I \cap J$, an additional facility node i' is introduced together with assignment arcs of zero weight. This seems to have a negative effect on the time spent in separating *acuts*.

Our results reveal that all instances from [1] are solved to optimality within approximately the same time the approach from [1] needs to find a heuristic solution. Finally, we also observe that the construction heuristic **H** (combined with bound-based reductions) also works quite good on its own. Within a second, our heuristic provides solutions whose quality is on average only 1% worse than the optimal solution. Comparing computing times of **H** versus **ADaR** and **YDaR**, we notice that most of the time of the latter two settings is consumed by the initialization heuristic (recall that the heuristic at the same time collects the family of *acuts* and performs the bound based reductions).

Tables 4 to 6 report results on the created larger instances (`NewSet`). In these tables, the results are aggregated according to the number of facility and customer nodes. The obtained results indicate that increasing the number of potential facilities (even when reducing the number of customers) is in direct correlation with the difficulty of the instances (at least for our exact approaches).

The best performing settings are again **YDaR** and **ADaR**. They allow for finding provably optimal solution of nearly every instance within the given timelimit. A significant fraction of the computing times is consumed by the initialization heuristic (see the runtimes of **H**). This is due to the calculation of $mSPF$ to properly set $\bar{\lambda}$ (recall that this procedure requires an all-facility-pair shortest path calculation). We notice that further speed-ups could be achieved if an upper bound for $\bar{\lambda}$ is heuristically estimated instead.

Y and **A** perform much worse than **YDaR** and **ADaR** (e.g., for NewSet1-GSTS **Y** cannot even solve a single instance within the given timelimit). **B** performs equally bad as **Y** and **A**. A significant improvement is obtained by applying bound based reductions (i.e., **BR**), however, this approach is still not competitive with **YDaR** and **ADaR**. Note that despite our cut-loop, **Y** and **A** often only manage to solve the root-node within timelimit. This is in strong contrast to the results for the instances from `Set1` to `Set5` and can be explained by the fact that by going from 100 to 500 nodes, the size of the model drastically increases (recall that the core graph is a complete graph). Thus, solving the LP-relaxations and the separation problem becomes much more time-consuming.

Moreover, the size of the core graph also explains why setting **B** does not really give an improvement over **Y**: While the "star" part of the model gets much smaller using Benders decomposition, the size of the "tree" part remains identical. This again emphasizes the importance of using the cuts obtained by dual ascent and the reductions for solving these kind of instances. Having a closer look at the four individual problems (and considering both, instances from [1] and the `NewSet`), the GSTS appears as the most difficult one for our exact approaches (see, e.g., Table 6).

Table 3: Results for instances from Bardossy and Raghavan [1].

	Y			**YDaR**			**A**			**ADaR**			**B**			**BR**			**H**	
set	$t[s]$	n.	gap	$t[s]$	n.	gap	$t[s]$	n.	gap	$t[s]$	n.	gap	$t[s]$	n.	gap	$t[s]$	n.	gap	$t[s]$	gap
Set1 - ConFL	524	1006	9.31	2	3	0.00	12	1	0.00	1	0	0.00	528	1072	10.10	136	1226	0.85	1	1.69
Set2 - ConFL	417	926	4.56	2	2	0.00	10	1	0.00	1	0	0.00	426	907	5.19	64	418	0.22	1	1.44
Set4 - ConFL	187	974	0.99	1	1	0.00	6	1	0.00	1	0	0.00	184	989	1.26	1	27	0	1	0.40
Set5 - ConFL	153	665	0.86	1	7	0.00	6	1	0.00	1	1	0.00	149	662	1.10	5	126	0	1	3.31
Set1 - STS	452	1778	8.11	1	1	0.00	9	1	0.00	1	0	0.00	466	2342	8.53	17	453	0.1	1	0.57
Set2 - STS	310	1668	2.91	1	0	0.00	7	0	0.00	1	0	0.00	321	1900	3.21	3	17	0.02	1	0.44
Set1 - GSTS	870	512	32.33	4	1	0.00	227	0	0.00	3	0	0.00	870	556	35.41	442	1769	13.35	1	1.99
Set2 - GSTS	310	1668	2.91	1	0	0.00	7	0	0.00	1	0	0.00	321	1900	3.21	3	17	0.02	1	0.44
Set3 - ROB	755	511	31.15	3	1	0.00	99	1	0.00	2	0	0.00	826	646	35.31	318	3795	8.58	1	0.85

Table 4: Results for NewSet1 and NewSet2 - ConFL.

		Y			**YDaR**			**A**			**ADaR**			**B**			**BR**			**H**	
$\|I\|$	$\|J\|$	$t[s]$	n.	gap	$t[s]$	n.	gap	$t[s]$	n.	gap	$t[s]$	n.	gap	$t[s]$	n.	gap	$t[s]$	n.	gap	$t[s]$	gap
50	450	23	3	0.00	1	0	0.00	295	2	0.01	1	0	0.00	15	2	0.00	1	0	0.00	1	0.13
100	400	122	6	0.00	2	0	0.00	525	2	0.75	2	0	0.00	78	8	0.00	2	0	0.00	2	0.22
150	350	314	8	0.06	4	0	0.00	722	1	3.85	4	0	0.00	226	9	0.01	4	0	0.00	4	0.33
200	300	601	8	0.64	7	0	0.00	837	1	9.24	7	0	0.00	463	11	0.32	8	1	0.00	6	0.50
250	250	856	3	3.09	11	0	0.00	825	1	15.34	12	0	0.00	835	8	2.47	16	3	0.00	10	0.69
300	200	900	1	7.74	21	0	0.00	886	0	9.35	22	0	0.00	900	3	7.60	183	19	0.69	17	1.58
350	150	900	2	13.86	34	0	0.00	900	0	9.04	36	0	0.00	900	2	13.78	463	77	3.89	26	2.31
400	100	900	4	22.67	66	0	0.00	900	0	11.52	64	0	0.00	900	4	23.25	812	99	15.98	42	3.65
450	50	900	11	37.49	180	5	0.06	900	0	22.88	149	0	0.00	900	13	37.87	881	70	34.94	64	6.90
50	450	30	4	0.00	1	0	0.00	320	3	0.20	1	0	0.00	22	5	0.00	1	0	0.00	1	0.15
100	400	130	4	0.00	3	0	0.00	473	2	3.41	3	0	0.00	80	7	0.00	3	0	0.00	2	0.36
150	350	402	9	1.30	5	0	0.00	678	1	9.21	5	0	0.00	303	12	0.63	6	2	0.00	4	0.43
200	300	694	6	3.85	10	0	0.00	782	0	14.92	10	0	0.00	628	10	3.30	16	9	0.00	8	0.74
250	250	824	4	8.73	19	0	0.00	829	0	18.51	20	0	0.00	832	5	8.04	161	66	0.00	15	1.57
300	200	900	3	12.76	33	0	0.00	896	0	6.30	37	0	0.00	900	4	12.58	348	131	3.14	24	2.13
350	150	900	6	18.49	62	0	0.00	899	0	8.94	63	0	0.00	900	6	18.70	639	92	11.45	39	3.42
400	100	900	10	25.68	162	4	0.00	900	0	14.60	142	0	0.00	900	11	26.27	755	32	23.49	56	5.65
450	50	900	15	35.97	351	12	0.36	900	0	24.65	264	0	0.00	900	16	36.96	881	39	34.81	85	7.60

Table 5: Results for NewSet1 and NewSet2 - STS.

		Y			**YDaR**			**A**			**ADaR**			**B**			**BR**			**H**	
$\|I\|$	$\|J\|$	$t[s]$	n.	gap	$t[s]$	n.	gap	$t[s]$	n.	gap	$t[s]$	n.	gap	$t[s]$	n.	gap	$t[s]$	n.	gap	$t[s]$	gap
50	450	3	0	0.00	1	0	0.00	5	0	0.00	1	0	0.00	0	0	0.00	0	0	0.00	0	0.03
100	400	9	0	0.00	1	0	0.00	35	0	0.00	1	0	0.00	3	0	0.00	1	0	0.00	1	0.14
150	350	31	1	0.00	3	0	0.00	128	1	0.00	2	0	0.00	11	2	0.00	3	0	0.00	2	0.09
200	300	155	4	0.09	5	0	0.00	280	0	0.00	4	0	0.00	86	6	0.00	6	1	0.00	4	0.27
250	250	553	7	2.36	8	0	0.00	539	0	0.30	7	0	0.00	468	10	1.91	12	5	0.00	7	0.69
300	200	859	4	7.32	14	0	0.00	757	0	1.03	11	0	0.00	860	6	6.92	25	10	0.00	12	0.85
350	150	900	4	14.90	22	0	0.00	894	0	3.32	17	0	0.00	900	5	14.99	105	48	0.44	19	1.58
400	100	900	8	25.73	38	0	0.00	900	0	10.52	30	0	0.00	900	8	26.27	466	170	6.97	32	2.80
450	50	900	18	40.72	79	1	0.00	900	0	23.84	64	0	0.00	900	15	41.52	522	99	20.63	47	3.48
50	450	4	0	0.00	1	0	0.00	6	0	0.00	1	0	0.00	1	0	0.00	0	0	0.00	0	0.10
100	400	13	1	0.00	1	0	0.00	23	0	0.00	1	0	0.00	3	2	0.00	1	0	0.00	1	0.14
150	350	189	13	0.46	3	0	0.00	76	0	0.00	2	0	0.00	76	13	0.00	3	0	0.00	2	0.12
200	300	404	13	3.06	6	0	0.00	257	0	0.00	5	0	0.00	321	15	2.13	8	9	0.00	5	0.63
250	250	668	7	9.38	12	0	0.00	448	0	0.35	9	0	0.00	655	9	8.12	53	55	0.23	10	0.97
300	200	686	7	15.16	17	0	0.00	650	0	2.53	15	0	0.00	701	9	15.28	91	63	0.93	15	0.87
350	150	740	9	21.03	35	0	0.00	708	0	6.66	30	0	0.00	735	12	21.61	323	126	8.68	28	2.23
400	100	871	15	29.07	95	0	0.00	768	0	14.96	92	0	0.00	900	12	29.89	513	67	20.08	46	5.56
450	50	900	22	40.35	89	0	0.00	895	0	25.28	89	0	0.00	900	22	40.96	309	48	13.24	61	2.65

Table 6: Results for NewSet1 and NewSet2 - GSTS.

		Y			**YDaR**			**A**			**ADaR**			**B**			**BR**			**H**	
$\|I\|$	$\|J\|$	$t[s]$	n.	gap	$t[s]$	n.	gap	$t[s]$	n.	gap	$t[s]$	n.	gap	$t[s]$	n.	gap	$t[s]$	n.	gap	$t[s]$	gap
500	450	900	1	3.52	43	0	0.00	900	0	3.33	37	0	0.00	890	1	3.00	50	2	0.00	37	0.65
500	400	900	0	4.76	41	0	0.00	900	0	3.76	39	0	0.00	900	0	3.85	65	4	0.00	36	0.90
500	350	900	0	6.27	40	0	0.00	900	0	4.25	33	0	0.00	900	1	5.42	55	3	0.00	35	0.70
500	300	900	1	8.88	49	0	0.00	900	0	5.28	42	0	0.00	900	1	8.41	119	14	0.15	44	1.27
500	250	900	0	12.09	53	0	0.00	900	0	6.31	42	0	0.00	900	0	11.60	234	19	2.00	47	1.70
500	200	900	1	15.63	49	0	0.00	900	0	8.48	43	0	0.00	900	1	15.49	257	22	2.23	43	1.68
500	150	900	2	20.35	53	0	0.00	900	0	11.01	43	0	0.00	900	2	20.86	402	117	5.15	48	1.58
500	100	900	6	29.04	74	0	0.00	900	0	16.79	63	0	0.00	900	6	30.12	645	107	16.19	64	3.32
500	50	900	16	41.98	121	3	0.00	900	0	29.07	91	0	0.00	900	14	43.06	580	76	24.74	67	5.12
500	450	710	0	9.08	52	0	0.00	774	0	5.24	50	0	0.00	702	0	8.48	149	14	1.34	45	0.92
500	400	713	0	10.96	54	0	0.00	783	0	5.68	53	0	0.00	702	1	10.53	239	36	2.59	46	1.18
500	350	717	1	12.61	51	0	0.00	761	0	6.38	54	0	0.00	718	1	12.75	191	39	2.39	45	0.93
500	300	772	2	15.68	67	0	0.00	796	0	8.13	66	0	0.00	800	3	15.89	315	32	5.55	56	1.98
500	250	772	2	17.77	59	0	0.00	808	0	9.88	58	0	0.00	785	3	18.17	288	26	6.13	52	1.34
500	200	831	6	21.88	86	0	0.00	801	0	12.50	90	0	0.00	861	6	21.97	394	39	10.99	64	2.74
500	150	898	7	26.45	103	0	0.00	829	0	16.81	118	0	0.00	900	8	28.10	502	46	16.58	83	3.26
500	100	900	11	32.88	144	0	0.00	881	0	20.98	162	0	0.00	900	11	34.64	540	24	24.65	84	4.93
500	50	900	18	41.00	147	0	0.00	900	0	29.07	172	0	2.50	900	20	42.20	356	22	16.99	89	3.50

4.3 Results for Instances from Gollowitzer and Ljubić [8]

We used a timelimit of 7200 seconds for the experiments reported in this section. Interestingly, the results obtained for these instances are very different to the results reported in the previous section.

We first focus on the instances `MPQ` whose results are reported in Table 7. The results show that dual ascent cuts and reductions do not have a strong (and positive) effect on the overall computing time. This is due to the specific cost structure for `MPQ` instances in which the "star" cost significantly dominates the "tree" cost. Hence, *acuts* are much less contributing to the strength of the lower bound of the underlying models. This can be also observed by comparing the number of nodes in the branch-and-bound tree, which is typically a two-digit number for `MPQ` whereas it is zero for the instances from/based on [1].

Our results confirm the finding of [8] that for `MPQ` instances, the separation of *acuts* seems to be quite time consuming, and therefore the settings **Y*** significantly outperform their **A*** counterparts. The overall best performing approach for `MPQ` instances is Benders decomposition (settings **B** and **BR** perform equally well). The sparser the core graph, the larger is the speed-up obtained by the Benders decomposition, which is in line with the previous observation that the facility location structure of these instances dominates the costs, and consequently, influences the performance the most.

We also report the computing times and the gaps of our heuristic **H**. Instances `MPQ` were also tested in [1]. Comparing the quality of solutions of these two heuristic approaches, we notice that our heuristic gaps are comparable (or slightly worse) to those reported in [1]. However, computing times of **H** are less than ten seconds in most of the cases, whereas [1] report computing times between 60 and 660 seconds (obtained on an AMD Athlon 62 X2 Dual with 2.61 GHz and 3 GB of RAM).

Finally, the benchmark set `GS` remains the most difficult one for ConFL - not a single instance from `GS` could be solved to optimality (neither in the previous literature, nor with our new approaches). Some of our runs separating *acuts* also terminated due to the memorylimit. A cumulative plot of the remaining gaps at termination is given in Figure 2. The smallest gaps are again obtained with the settings **B** and **BR**. Compared to the results reported in [8], we manage to improve the final gaps by about 0.10-0.30%. We point out that these improvements are not insignificant, given that `GS` instances are known to posses a very large number of local optima, most of them lying within the 1% of the root lower bound.

Concerning the performance of our heuristic **H**, the obtained gaps are between 0.3% and 0.5% (again, they are comparable to those reported in [1]). However, it is worth mentioning that our heuristic requires between one and four seconds, whereas the runtimes reported in [1] are between 100 and 5500 seconds for the same instances.

Figure 2: Results for instance-set GS from Gollowitzer and Ljubić [8]

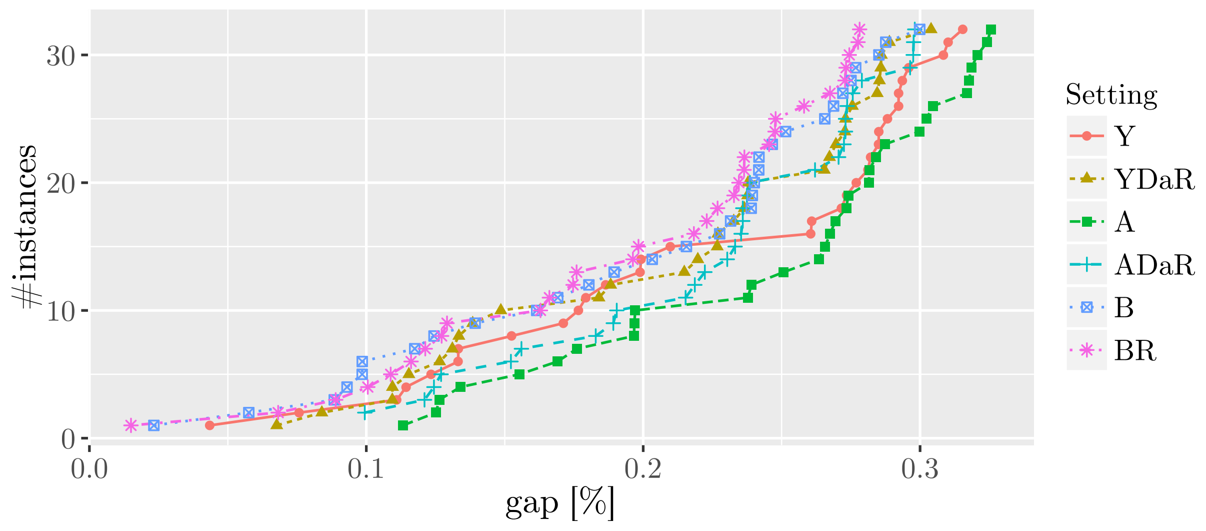


Table 7: Results for instances MPQ from Gollowitzer and Ljubić [8].

	Y			**YDaR**			**A**			**ADaR**			**B**			**BR**			**H**	
name	$t[s]$	n.	gap	$t[s]$	n.	gap	$t[s]$	n.	gap	$t[s]$	n.	gap	$t[s]$	n.	gap	$t[s]$	n.	gap	$t[s]$	gap
c5-mp1	21	51	0.00	33	51	0.00	72	15	0.00	115	31	0.00	3	29	0.00	4	23	0.00	2	4.18
c5-mp2	11	39	0.00	66	17	0.00	25	7	0.00	194	9	0.00	2	16	0.00	6	12	0.00	4	10.22
c5-mq1	69	57	0.00	112	83	0.00	431	53	0.00	467	61	0.00	5	35	0.00	15	36	0.00	10	5.65
c5-mq2	78	74	0.00	765	75	0.00	981	63	0.00	3186	59	0.00	13	64	0.00	42	53	0.00	22	8.50
c10-mp1	20	17	0.00	51	31	0.00	343	33	0.00	845	29	0.00	5	15	0.00	7	18	0.00	2	6.72
c10-mp2	12	19	0.00	9	11	0.00	115	9	0.00	143	7	0.00	2	9	0.00	4	8	0.00	2	2.85
c10-mq1	46	59	0.00	83	41	0.00	501	39	0.00	1274	49	0.00	6	31	0.00	16	40	0.00	10	5.65
c10-mq2	69	81	0.00	80	77	0.00	1425	65	0.00	2945	85	0.00	12	42	0.00	22	48	0.00	13	4.35
c15-mp1	18	21	0.00	25	23	0.00	1025	21	0.00	1263	39	0.00	5	11	0.00	7	14	0.00	2	3.72
c15-mp2	14	9	0.00	11	7	0.00	123	5	0.00	223	13	0.00	3	5	0.00	5	9	0.00	2	2.59
c15-mq1	51	53	0.00	64	65	0.00	1092	45	0.00	1893	85	0.00	9	35	0.00	20	32	0.00	9	5.68
c15-mq2	55	35	0.00	54	69	0.00	2298	81	0.00	4215	55	0.00	11	22	0.00	23	18	0.00	7	3.95
c20-mp1	48	21	0.00	54	21	0.00	2427	13	0.00	4642	47	0.00	28	11	0.00	24	9	0.00	2	3.83
c20-mp2	23	5	0.00	19	5	0.00	319	13	0.00	274	11	0.00	10	3	0.00	8	0	0.00	2	2.19
c20-mq1	72	39	0.00	81	41	0.00	2148	49	0.00	2249	37	0.00	42	32	0.00	43	27	0.00	9	5.55
c20-mq2	179	39	0.00	130	27	0.00	7200	26	2.40	7200	28	2.41	87	33	0.00	75	40	0.00	9	4.07
d5-mp1	18	25	0.00	17	55	0.00	114	35	0.00	132	7	0.00	3	12	0.00	4	9	0.00	2	3.65
d5-mp2	15	31	0.00	23	37	0.00	72	21	0.00	144	23	0.00	3	17	0.00	5	17	0.00	3	3.46
d5-mq1	44	39	0.00	50	35	0.00	219	27	0.00	272	39	0.00	4	27	0.00	14	28	0.00	10	5.06
d5-mq2	45	39	0.00	52	69	0.00	296	19	0.00	408	21	0.00	7	28	0.00	18	26	0.00	9	4.27
d10-mp1	21	21	0.00	45	31	0.00	1872	23	0.00	1329	33	0.00	9	15	0.00	11	17	0.00	3	3.89
d10-mp2	18	17	0.00	50	30	0.00	1586	33	0.00	2146	35	0.00	6	9	0.00	9	12	0.00	3	3.54
d10-mq1	51	37	0.00	198	73	0.00	1813	57	0.00	1810	51	0.00	8	26	0.00	19	27	0.00	11	5.74
d10-mq2	64	39	0.00	74	41	0.00	3020	29	0.00	5532	67	0.00	14	30	0.00	21	25	0.00	9	4.13
d15-mp1	26	29	0.00	28	11	0.00	826	55	0.00	1441	59	0.00	9	12	0.00	10	13	0.00	2	4.46
d15-mp2	17	19	0.00	20	9	0.00	314	9	0.00	529	33	0.00	6	12	0.00	9	5	0.00	2	3.95
d15-mq1	44	35	0.00	49	38	0.00	1303	81	0.00	899	41	0.00	13	35	0.00	21	33	0.00	8	5.83
d15-mq2	65	29	0.00	64	45	0.00	5346	67	0.00	5647	59	0.00	16	36	0.00	25	35	0.00	8	4.12
d20-mp1	68	19	0.00	80	37	0.00	4461	19	0.00	5667	47	0.00	50	11	0.00	60	12	0.00	2	4.19
d20-mp2	71	8	0.00	84	5	0.00	2689	5	0.00	2733	11	0.00	31	5	0.00	51	5	0.00	3	2.26
d20-mq1	166	47	0.00	163	37	0.00	7200	20	2.59	7200	20	2.62	98	29	0.00	128	35	0.00	9	5.54
d20-mq2	233	69	0.00	155	39	0.00	7200	39	2.43	7200	26	2.36	143	25	0.00	131	33	0.00	8	4.23

5 Conclusion

Tree-star structures appear in many combinatorial optimization problems, and they are particularly important for applications arising in the design of telecommunication or data management networks. In this paper, we present an unified algorithmic framework to deal with problems from this family, namely the connected facility location problem, the (generalized) Steiner tree-star problem and the rent-or-buy problem. We investigate three important aspects for enhancing the state-of-the art branch-and-cut methods from the literature: (i) utilization of strong cuts derived from a dual ascent procedure, (ii) application of dual-based reductions to reduce the size of the input graphs, and (iii) incorporation of construction/primal heuristic heuristic in these frameworks.

Our computational study reveals that these new ingredients are crucial for the solution of very large classes of instances. As a matter of fact, the open (generalized) Steiner tree-star instances and the rent-or-buy instances in the literature for which the optimal values were unknown, are now solved to optimality (often within a few seconds) using our implementation. Additionally, the construction heuristic also performs very well on its own, being competitive in solution quality with the state-of-the-art, while often outperforming it in runtime.

Acknowledgements

The work of M. Leitner was supported by the Austrian Research Fund (FWF, Project I 892-N23). The work of I. Ljubić and M. Sinnl was supported by the Austrian Research Fund (FWF, Project P 26755-N19). The work of J.J. Salazar-González was supported by the Spanish Research Project MTM2015-63680-R (MINECO/FEDER, UE).

References

[1] M. G. Bardossy and S. Raghavan. Dual-based local search for the connected facility location and related problems. *INFORMS Journal on Computing*, 22(4):584–602, 2010.

[2] J. E. Beasley. OR-Library: Distributing Test problems by Electronic Mail. 1990.

[3] M. Charikar, C. Chekuri, T.-Y. Cheung, Z. Dai, A. Goel, S. Guha, and M. Li. Approximation algorithms for directed Steiner problems. *Journal of Algorithms*, 33(1):73–91, 1999.

[4] C.-H. Chu, G. Premkumar, and H. Chou. Digital data networks design using genetic algorithms. *European Journal of Operational Research*, 127(1):140–158, 2000.

[5] M. P. de Aragão and R. F. Werneck. On the implementation of MST-based heuristics for the Steiner problem in graphs. In *Algorithm engineering and experiments*, pages 1–15. Springer, 2002.

[6] F. Eisenbrand, F. Grandoni, T. Rothvoß, and G. Schäfer. Connected facility location via random facility sampling and core detouring. *Journal of Computer and System Sciences*, 76(8):709–726, 2010.

[7] M. Fischetti, I. Ljubić, and M. Sinnl. Redesigning Benders Decomposition for Large Scale Facility Location. *Management Science*, 2016. online first.

[8] S. Gollowitzer and I. Ljubić. MIP models for connected facility location: A theoretical and computational study. *Comput. & Oper. Res.*, 38(2):435–449, 2011.

[9] F. Grandoni and T. Rothvoß. Approximation algorithms for single and multi-commodity connected facility location. In O. Günlük and G. Woeginger, editors, *IPCO 2011, New York, NY, USA, Proceedings*, volume 6655 of *LNCS*, pages 248–260. Springer, 2011.

[10] F. Havet and M. Wennink. The push tree problem. *Networks*, 44(4):281–291, 2004.

[11] M. Hoefer. UflLib, Benchmark Instances for the Uncapacitated Facility Location Problem, 2003.

[12] D. Karger and M. Minkoff. Building Steiner trees with incomplete global knowledge. In *FOCS*, pages 613–623. IEEE Computer Society, 2000.

[13] S. Khuller and A. Zhu. The general Steiner tree-star problem. *Information Processing Letters*, 84(4): 215–220, 2002.

[14] T. Koch and A. Martin. Solving Steiner tree problems in graphs to optimality. *Networks*, 32:207–232, 1998.

[15] M. Körkel. On the exact solution of large-scale simple plant location problems. *European Journal of Operational Research*, 39(2):157–173, 1989.

[16] J. Kratica, D. Tošic, V. Filipović, and I. Ljubić. Solving the simple plant location problem by genetic algorithm. *RAIRO-Operations Research*, 35(01):127–142, 2001.

[17] C. Krick, H. Räcke, and M. Westermann. Approximation algorithms for data management in networks. *Theory Comput. Syst.*, 36(5):497–519, 2003.

[18] Y. Lee, L. Lu, Y. Qiu, and F. Glover. Strong formulations and cutting planes for designing digital data service networks. *Telecommunication Systems*, 2(1):261–274, 1993.

[19] Y. Lee, S. Y. Chiu, and J. Ryan. A branch and cut algorithm for a Steiner tree-star problem. *INFORMS Journal on Computing*, 8(3):194–201, 1996.

[20] M. Leitner and G. R. Raidl. Branch-and-cut-and-price for capacitated connected facility location. *J. of Math. Modelling and Algorithms*, 10(3):245–267, 2011.

[21] M. Leitner, I. Ljubić, J. J. Salazar-González, and M. Sinnl. On the asymmetric connected facility location polytope. *Lecture Nodes in Computer Science*, 8596:371–383, 2014.

[22] M. Leitner, I. Ljubić, J. J. Salazar-González, and M. Sinnl. The Connected Facility Location Polytope. *Discrete Applied Mathematics*, 2016. to appear.

[23] I. Ljubić. A hybrid VNS for connected facility location. In T. Bartz-Beielstein, M. J. B. Aguilera, C. Blum, B. Naujoks, A. Roli, G. Rudolph, and M. Sampels, editors, *Hybrid Metaheuristics, Proceedings*, volume 4771 of *Lecture Notes in Computer Science*, pages 157–169. Springer, 2007.

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

[25] K. Mehlhorn. A faster approximation algorithm for the Steiner problem in graphs. *Information Processing Letters*, 27(3):125–128, 1988.

[26] P. Nuggehalli, V. Srinivasan, and C.-F. Chiasserini. Energy-efficient caching strategies in ad hoc wireless networks. In *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, pages 25–34. ACM, 2003.

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

[28] C. Swamy and A. Kumar. Primal–dual algorithms for connected facility location problems. *Algorithmica*, 40(4):245–269, 2004.

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

[30] J. Xu, S. Y. Chiu, and F. Glover. Using tabu search to solve the Steiner tree-star problem in telecommunications network design. *Telecommunication Systems*, 6(1):117–125, 1996.

[31] J. Zhu, J. Huang, D. Liu, and J. Han. Connected distribution center location problem under traffic network in emergency management. In *CSO 2012*, pages 160–163, 2012.