A Hybrid VNS for Connected Facility Location

Ivana Ljubić[*]

Department of Statistics and Decision Support Systems
University of Vienna
Austria
ivana.ljubic@univie.ac.at

Abstract. The connected facility location (ConFL) problem generalizes the facility location problem and the Steiner tree problem in graphs. Given a graph $G = (V, E)$, a set of customers $\mathcal{D} \subseteq V$, a set of potential facility locations $\mathcal{F} \subseteq V$ (including a root r), and a set of Steiner nodes in the graph $G = (V, E)$, a solution (F, T) of ConFL represents a set of open facilities $F \subseteq \mathcal{F}$, such that each customer is assigned to an open facility and the open facilities are connected to the root via a Steiner Tree T. The total cost of the solution (F, T) is the sum of the cost for opening the facilities, the cost of assigning customers to the open facilities and the cost of the Steiner tree that interconnects the facilities.
We show how to combine a variable neighborhood search method with a reactive tabu-search, in order to find sub-optimal solutions for large scale instances. We also propose a branch-and-cut approach for solving the ConFL to provable optimality. In our computational study, we test the quality of the proposed hybrid strategy by comparing its values to lower and upper bounds obtained within a branch-and-cut framework.

1 The Connected Facility Location Problem

Due to increasing customer demands regarding broadband connections, telecommunication companies search for solutions that "push" rapid and high-capacity fiber-optic networks closer to the subscribers, thus replacing the outdated copper twisted cable connections. The *Connected Facility Location Problem* (ConFL) models the next-generation of telecommunication networks: In the so-called *tree-star* networks, the core (fiber-optic) network represents a tree. This tree interconnects multiplexers that switch between fiber optic and copper connections. Each selected multiplexer is the center of the star-network of copper connections to its customers.

ConFL represents a generalization of two prominent combinatorial optimization problems: the *facility location problem* and the *Steiner tree problem* in graphs. More formally, ConFL is defined as follows: We are given an undirected graph $G = (V, E)$ with a set of *facilities* $\mathcal{F} \subseteq V$ and a set of *customer nodes* $\mathcal{D} \subseteq V$. We assign opening costs $f_i \geq 0$ to each facility $i \in \mathcal{F}$, edge costs $c_e \geq 0$

[*] Supported by the Hertha-Firnberg Fellowship of the Austrian Science Foundation (FWF)

to each edge $e \in E$, and demands d_j to each customer $j \in \mathcal{D}$. We also assume that there is the set of Steiner nodes $\mathcal{S} = V \setminus (\mathcal{F} \cup \mathcal{D})$ and a root node $r \in \mathcal{F}$. Costs of assigning a customer $j \in \mathcal{D}$ to a facility $i \in \mathcal{F}$ are given as $a_{ij} \geq 0$. A solution (F, T) of ConFL represents a set of open facilities $F \subseteq \mathcal{F}$, such that each customer $j \in \mathcal{D}$ is assigned to an open facility $i(j) \in F$ and the open facilities are connected to the root $r \in F$ by a Steiner Tree T. The total cost of the solution (F, T) is the sum of the cost for opening the facilities, the cost of assigning customers to the open facilities and the cost of the Steiner tree that interconnects the facilities:

$$\sum_{i \in F} f_i + \sum_{j \in \mathcal{D}} d_j a_{i(j)j} + \sum_{e \in T} c_e.$$

In this constellation, some facility nodes may be used as pure Steiner nodes, in which case no opening costs for them will be paid.

As mentioned in [17], we can assume without loss of generality that the root r represents an open facility and hence belongs to the Steiner tree. To solve the unrooted version of the problem, we simply need to run the algorithm for all facility nodes chosen as the root r. Without loss of generality, we can also assume that customer demands are all equal to one. Otherwise, we can set $a_{ij} \leftarrow d_j a_{ij}$ for all pairs (i, j).

In Section 2, we propose a hybrid approach that combines Variable Neighborhood Search (VNS) with a reactive tabu search method. Section 3 describes a branch-and-cut (B&C) approach to solve the problem to provable optimality. In the last Section, our computational results show the comparison of the proposed hybrid approach with lower and upper bounds obtained within the B&C framework.

1.1 Related Work

ConFL has been introduced by Karger and Minkoff [6] who gave the first approximation algorithm of a constant factor. For the *metric* ConFL in which c_e is a metric and $a_{ij} = 1/Mc_{ij}$, for some constant $M > 1$, Swamy and Kumar [17] used an integer linear programming (ILP) formulation to develop a primal-dual approximation algorithm of factor 8.55.

The *rent-or-buy* problem is a special case of ConFL in which there are no opening costs of the facilities and $\mathcal{F} = V$. A randomized 3.55-approximation algorithm for the metric rent-or-buy problem has been proposed by Gupta et al. [3]. The rent-or-buy problem has also been studied by Nuggehalli et al. [16] who gave a distributed greedy algorithm with approximation ratio 6, in the context of the design of ad hoc wireless networks. Their algorithm solves the rent-or-buy problem to optimality if the underlying graph has a tree topology.

The *Steiner tree-star* problem is another problem related to ConFL and to the node-weighted Steiner tree problem in graphs. The main difference to ConFL lies in the cost structure. To each non-customer node, we assign costs $f_i, i \in V \setminus \mathcal{D}$, assuming therefore that $\mathcal{F} = V \setminus \mathcal{D}$. If node i belongs to the Steiner tree T, we pay for it no matter if any customer is assigned to it or not. Thus, the objective of the Steiner tree-star problem looks as follows:

$$\min \sum_{i \in T} f_i + \sum_{j \in \mathcal{D}} c_{i(j)j} + \sum_{e \in T} c_e,$$

where c is the cost-function used both for assignments and for edge-costs. Khuller and Zhu [7] gave a 5-approximation algorithm for solving the metric version of the problem. Lee et al. [11] proposed a branch-and-cut algorithm based on a separation of anti-cycle constraints. Their algorithm solved instances with up to 200 nodes to provable optimality. Xu et al. [18] developed a tabu search heuristic that incorporates long-term memory and probabilistic move selections. The authors considered insert-, delete-, and swap-moves, whereas swap-moves are used for diversification purposes. Computational results are given for instances of Lee et al. [11] and for additional sets of instances with up to 900 nodes. For the largest instances, the running time of the algorithm of more than 10 hours was reported.

Note that without the connectivity requirement (connecting the facilities by a Steiner tree), the ConFL reduces to the *uncapacitated facility location problem* (UFLP). On the other hand, if the set of facilities to be opened is known in advance, the problem is reduced to the Steiner tree problem in graphs (STP). Therefore, clearly, the problem is NP-hard.

2 A VNS with a Tabu Search

2.1 Basic VNS Model

According to Hoefer's computational study [5] on the uncapacitated facility location problem (UFLP), one of the successful metaheuristic approaches for solving the UFLP is a tabu-search approach given by Michel and van Hentenryck [15]. Recently, the authors obtained a very efficient strategy by simply extending a tabu-search approach with a variable neighborhood search. Our VNS framework follows these basic ideas given in [4]. Note however that, due to the nature of the problem, the way we calculate the objective value significantly differs from the one used to evaluate UFLP (see Section 2.2). Algorithm 1 shows our generic approach.

Representation: Assuming that for a fixed set of facilities, we can deterministically find a (sub)-optimal solution, we conduct our local search in the space of facility locations, thus changing configurations of vectors $y = (y_1, \dots, y_{|\mathcal{F}|})$, where $y_i = 1$ if facility i is open.

When it is clear from context, we will use y to denote the subset of open facilities (i.e. those with $y_i = 1$).

k-Neighborhood: We define a k-neighborhood N_k of a solution $\hat{y}$ by all solutions y such that the Hamming distance $d(\hat{y}, y)$ between these two binary vectors is equal to k:

$$\mathcal{N}_k(\hat{y}) = \{y \in \{0,1\}^{|\mathcal{F}|} \mid d(\hat{y}, y) = k\}.$$

```
Data   : Instance of the ConFL.
Result : A feasible suboptimal solution to ConFL.

$best = \hat{y} = Initialize()$;
$nIter = 0$;
while $nSame < LimitSame$ and $Time < TimeLimit$ do
    $y' = TabuSearch(\mathcal{N}_1(\hat{y}))$;
    $nIter++$;
    if $Obj(y') > Obj(\hat{y})$ then
        $nSame++$;
        $\hat{y} = Shake(y')$;
    else
        $\hat{y} = y'$;
        if $Obj(y') < Obj(best)$ then
            $nSame = 0$;
            Decrease the $tabuLength$;
            $best = y'$;
        else
            $nSame++$;
            Increase the $tabuLength$;
        end
    end
end
return $best$;
```

Algorithm 1: VNS algorithm.

Reactive Tabu Search: The status of a facility $i \in \mathcal{F}$ is given by y_i. A basic *move* is the change of the status of a facility, i.e. $y_i \leftarrow 1 - y_i$. The tabu list consists of the set of facilities that cannot be flipped. A solution $\hat{y}$ is locally improved using the *best improvement* strategy with respect to its 1-neighborhood. Thus, all possible flips of single positions that are not in the tabu list are considered, and the best one is taken. If there is more than one best flip, we randomly select one.

In order to forbid the reversal of recent search steps, we use a self-learning mechanism that adapts the length of the tabu list during the search. We simplify the ideas of the *reactive tabu search*, which was originally proposed by Battiti and Tecchiolli [1]: the list size is increased whenever no improvement upon the best found solution is made. Whenever a new best solution is detected, the list size is decreased.

We implement a dynamic tabu list in the following way: to each facility $i \in \mathcal{F}$, we associate a counter $tabuList(i)$. When a facility is inserted into the tabu list, we set $tabuList(i) \leftarrow nIter + tabuLength$, which forbids flipping the facility i for the next $tabuLength$ iterations, whereby $nIter$ denotes the current iteration. The value of $tabuLength$ is adjusted automatically: if tabu search improves the value of $\hat{y}$, but it is still worse than the best obtained value, we increase the length of the tabu list by one. Otherwise, the length of the tabu list will be decreased by one. We use standard settings for minimal and maximal values of *tabuLength*, and set them to 2 and 10, respectively.

Shaking the Neighborhood: This diversification mechanism enables escaping from local optima found within the tabu search procedure. If the last tabu search iteration did not improve upon the last selected value $\hat{y}$, we randomly select k ($k \geq 2$) positions and flip them. The value k increases until it reaches a pre-specified maximum neighborhood size (50 in the default implementation), after which it starts from 2 again.

Using this technique, the diversification degree will be automatically adjusted. Increasing the size of neighborhoods systematically also assures that significant diversifications are avoided during early phases of the search.

Hash-Tables: Since the evaluation of solutions is computationally expensive (see next Subsection), we maintain hash-tables for all vectors y for which the objective value for STP, assignment or total ConFL value is already known (see also [10], where a similar idea has been used). This strategy ensures that the objective value of the same vector will not be calculated more than once within the whole procedure, even if we return back to the same solution.

Termination Criteria: The algorithm terminates if the best found solution was not improved within the last *LimitSame* iterations, or a pre-specified *TimeLimit* is exceeded. In our default implementation, we set *LimitSame* to 50, and *TimeLimit* to 1000 seconds.

2.2 Evaluation of the Objective Function

```
Data   : Vector $\hat{y}$: a facility $i$ is selected if $\hat{y}_i = 1$.
Result : Locally improved vector $y^P$ and its objective function value.

if $Hash(\hat{y})$ defined then
    $(x^P, T^P, y^P) = Hash(\hat{y})$;
else
    $(T^{MST}, y^{MST}) = MSTHeuristic(\hat{y})$;
    $(x^A, y^A) = Assign(y^{MST})$;
    $(x^P, T^P, y^P) = Peeling(T^{MST}, x^A, y^A)$;
    Insert $(x^P, T^P, y^P)$ into $Hash$;
end
return $\sum_{e \in T_P} c_e + \sum_{i \in \mathcal{F}} f_i y_i^P + \sum_{i \in \mathcal{F}} \sum_{j \in \mathcal{D}} a_{ij} x_{ij}^P$;
```

Algorithm 2: Calculating the objective function.

Algorithm 2 shows the main steps of calculating the objective function for a specified vector $\hat{y}$. We use the following notation:

– vectors x ($= x^P$ or x^A) refer to the assignment values, i.e. $x_{ij} = 1$ if customer j is assigned to facility i and $x_{ij} = 0$, otherwise;
– T^P and T^{MST} denote the sets of nodes and edges building a Steiner tree that connects the chosen set of facilities (y^P and y^{MST}, resp.).

Given $\hat{y}$, we first check if this configuration has been already calculated before. If so, we get the corresponding tree-, assignment-, and facility values from the hash-table *Hash*. Otherwise, we run a three-step procedure:

Step 1: $(T^{MST}, y^{MST}) = \mathit{MSTHeuristic}(\hat{y})$**:** We consider the graph $G' = (V', E')$ – a subgraph of G induced by the set of facilities and Steiner nodes $V' = \mathcal{F} \cup \mathcal{S}$ with the edge costs c. For G', we generate the so-called *distance network*[1] - a complete graph whose nodes correspond to facilities $i \in \mathcal{F}$, and whose edge-lengths $l(i, j)$ are defined as shortest paths in G', for all $i, j \in \mathcal{F}$.
We use the minimum spanning tree (MST) heuristic [14] to find a spanning tree T^{MST} that connects all selected facilities ($\hat{y}_i = 1$).
1. Let G'' be the subgraph of G' induced by $\hat{y}$.
2. Calculate the minimum spanning tree MST''_G of the distance sub-network G''.
3. Include in T^{MST} all intermediate edges and nodes of G contained in selected shortest-path edges from MST''_G.
4. Update the set of selected facilities: set $y_i^{MST} = 1$ for all nodes $i \in T^{MST} \cap \mathcal{F}$.

Step 2: $(x^A, y^A) = \mathit{Assign}(y^{MST})$**:** For each customer $j \in \mathcal{D}$, we find the cheapest possible assignment to a facility from y^{MST}. The values are stored in vector x^A. Since not every facility from y^{MST} necessarily serves a customer, we denote with y^A the subset of those that really need to be opened.
This operation is calculated from scratch – although the differences between two neighboring vectors $\hat{y}_1$ and $\hat{y}_2$ are in general very small, the corresponding y_1^{MST} and y_2^{MST} solutions may be significantly different. Thus, the total computational complexity for finding the cheapest assignment in the worst case is $O(|\mathcal{F}||\mathcal{D}|)$.

Step 3: $(x^P, T^P, y^P) = \mathit{Peeling}(T^{MST}, x^A, y^A)$**:** We finally want to get rid of some of those facilities that are still part of the Steiner tree, but that are not used at all. We do this by applying the so-called *peeling procedure*. Our peeling heuristic tries to recursively remove all redundant leaf nodes (including corresponding tree-paths) from the tree-solution T^{MST}. Let k denote a leaf node, and let P_k be a path that connects k to the next open facility from y^A, or to the next branch, towards the root r.

[1] Calculation of the distance network is done only once in the beginning of the VNS algorithm.

1. If the leaf node is not an (open) facility, i.e. if $k \notin y^A$, we simply delete P_k.
2. Otherwise, we try to to re-assign customers (originally assigned to k) to already open facilities (if possible). If such obtained solution is better, we delete P_k and continue processing other leaves.

The main steps of this procedure are given in Algorithm 3.
If the set of facilities is sorted for each customer in increasing order with respect to its assignment costs[2], this procedure can be implemented very efficiently. Indeed, in order to find an open facility (from y^P) nearest to j and different from k (denoted by $i^k(j)$), we only need to proceed this ordered list starting from k until we encounter a facility from y^P.
The algorithm stops when only one node is left, or when all the leaves from T^P have been proceeded. Thus, the worst-case running time of the whole peeling method is $O(|\mathcal{F}||\mathcal{D}|)$.

```
Data   : Set $y^A$, assignment $x^A$ and a Steiner tree $T^{MST}$.
Result : Locally improved vector $y^P$ and its objective function value.

$T^P = T^{MST}$, $y^P = y^{MST}$, $x^P = x^A$;
for all leaves $k$ in $T^P$ do
    Determine path $P_k$ and its costs $c(P_k) = \sum_{e \in P_k} c_e$;
    if $k \notin y^P$ then
        $T^P = T^P - P_k$;
    else
        $\mathcal{D}_k = \{j \mid j \in \mathcal{D}, x^P_{kj} = 1\}$;
        $i^k(j) = \arg\min\{a_{ij} \mid i \in y^P, i \neq k\}, \forall j \in \mathcal{D}_k$;
        if $\sum_{j \in \mathcal{D}_k} a_{i^k(j)j} < f_k + c(P_k) + \sum_{j \in \mathcal{D}_k} a_{kj}$ then
            $y^P_k = 0$;
            $T^P = T^P - P_k$;
            $x^P_{kj} = 0$, $x^P_{i(k)j} = 1, \forall j \in \mathcal{D}_k$;
        end
    end
end
```

Algorithm 3: Peeling procedure.

[2] Sorting of these lists is done once, in the initialization phase of VNS algorithm.

3 Branch-and-Cut for ConFL

We propose to calculate lower bounds and provably optimal solutions to ConFL using the integer linear programming (ILP) model given below. For solving the linear programming relaxations and for a generic implementation of the branch-and-cut approach, we used the commercial packages ILOG CPLEX (version 10.0) and ILOG Concert Technology (version 2.2).

We solve the ConFL to optimality on a directed graph $G_A = (V, A)$ obtained from the original one $G = (V, E)$ by simply replacing each edge $e \in E$ by two directed arcs of the same cost:

$$A = \{(k, l) | \{k, l\} \in E \wedge l \neq r\}$$

$$c_{kl} = c(\{k, l\}), \quad \forall (k, l) \in A$$

The assignment costs (a_{ij}) remain unchanged.

The problem of finding a rooted Steiner tree on a directed graph is known as the *Steiner arborescence problem*: given G_A, a root r and the set of terminals $F \subset V$, find a subset of arcs $R \subset A$ such that there is a directed path from r to each $i \in F$, and that $\sum_{(k,l)\in R} c_{kl}$ is minimized.

To model the problem, we use the following binary vectors: y_i indicates whether a facility i is open, x_{ij} indicates whether customer j is assigned to facility i and z_{kl} indicates whether the arc (k, l) is a part of the directed Steiner tree rooted at r.

$$(ConFL) \qquad \min \sum_{i\in\mathcal{F}} f_i y_i + \sum_{i\in\mathcal{F}} \sum_{j\in\mathcal{D}} a_{ij} x_{ij} + \sum_{(k,l)\in A} c_{kl} z_{kl} \tag{1}$$

$$\textstyle\sum_{i\in\mathcal{F}} x_{ij} \geq 1, \qquad \forall j \in \mathcal{D} \tag{2}$$

$$x_{ij} \leq y_i, \qquad \forall i \in \mathcal{F}\ \forall j \in \mathcal{D} \tag{3}$$

$$\textstyle\sum_{(k,l)\in\delta^-(S)} z_{kl} \geq y_i, \forall S \subseteq V \setminus \{r\}, i \in S \cap \mathcal{F} \neq \emptyset \tag{4}$$

$$y_r = 1 \tag{5}$$

$$0 \leq x_{ij}, z_{kl}, y_i \leq 1 \quad \forall i,\ \forall j,\ \forall (k, l) \in A \tag{6}$$

$$x_{ij}, z_{kl}, y_i \in \{0, 1\} \quad \forall i,\ \forall j,\ \forall (k, l) \in A \tag{7}$$

Here, with $\delta^-(S)$ we denote the set of ingoing edges of S, i.e., $\delta^-(S) = \{(k, l) \in A \mid k \notin S, l \in S\}$.

The *assignment constraints* (2) ensure that each customer is assigned to exactly one facility. The *capacity constraints* (3) ensure that customers can only be assigned to open facilities. The *connectivity constraints* (4) guarantee that there is a directed path between every open facility and the root r, i.e. they ensure that open facilities are connected to the root and to each other. With constraint (5) we fix the root node r. Constraints (4) and (5) ensure existence of the Steiner arborescence, whereas constraints (2) and (3) ensure a feasible assignment.

Initialization: We initialize the LP with relaxed integer requirements (6), with assignment- and capacity-inequalities (2)-(3), with indegree inequalities:

$$\sum_{(k,l)\in A} z_{kl} = y_l, \forall l \in \mathcal{F}$$

and with the *subtour elimination constraints* of size two:

$$z_{kl} + z_{lk} \leq y_l, \quad \forall l \in \mathcal{F}.$$

Additionally, we add flow-balance constraints ([8]) that ensure that the in-degree of each Steiner node is less or equal than its out-degree:

$$\sum_{(k,l)\in A} z_{kl} \leq \sum_{(l,k)\in A} z_{lk}, \quad \forall l \notin \mathcal{F}.$$

Separation of Cut Inequalities: In each node of the branch-and-bound tree we separate the cut-inequalities given by (4). For a given LP-solution $\hat{z}$, we construct a support graph $G_{\hat{z}} = (V, A, z)$ with arc-weights $\hat{z} : A \mapsto [0, 1]$. Then we calculate the minimum cost flow from the root r to each potential facility node $i \in \mathcal{F}$ such that $y_i > 0$. If this min-cost flow value is less than y_i, we have a violated inequality, induced by the corresponding min-cut in the graph $G_{\hat{z}}$, and we insert it into the LP.

To improve computational efficiency, we search for *nested, back* and *minimum-cardinality cuts* and insert at most 100 violated inequalities in each separation phase. For more details, see our implementation of the B&C algorithm for the prize-collecting Steiner tree problem, where the same separation procedure has been used [12, 13].

Branching: Branching on single arc variables produces a huge disbalance in the branch-and-bound tree. Whereas discarding an edge from the solution (setting z_{kl} to zero) doesn't bring much, setting the node variable to one, significantly reduces the size of the search subspace. Therefore we set the highest branching priority to potential facility nodes $i \in \mathcal{F}$.

4 Computational Results

We consider three classes of benchmark instances, obtained by merging data from three public sources. In general, we combine an UFLP instance with an STP instance, to generate ConFL input graphs in the following way: first $|\mathcal{F}|$ nodes of the STP instance are selected as potential facility locations, and the node with index 1 is selected as the root. The number of facilities, the number of customers, opening costs and assignment costs are provided in UFLP files. STP files provide edge-costs and additional Steiner nodes.

- We consider two sets of non-trivial UFLP instances from UflLib[3]:

[3] http://www.mpi-inf.mpg.de/departments/d1/projects/benchmarks/UflLib/

  - MP-{1,2} and MQ-{1,2} instances have been proposed by Kratica et al. [10]. They are designed to be similar to UFLP real-world problems and have a large number of near-optimal solutions. There are 6 classes of problems, and for each problem $|\mathcal{F}| = |\mathcal{D}|$. We took 2 representatives of the 2 classes MP and MQ of sizes 200×200 and 300×300, respectively.
  - The GS-{250,500}-{1,2} benchmark instances were initially proposed by Koerkel [9] (see also Ghosh [2]). Here we chose two representatives of the 250×250 and 500×500 classes, respectively. Connection costs are drawn uniformly at random from [1000, 2000], while opening costs are drawn uniformly at random from [100, 200].
- STP instances:
  - Instances {C,D}5, {C,D}10, {C,D}15, {C,D}20 were chosen randomly from the OR-library[4] as representatives of medium size instances for the STP.

All experiments were performed on a Pentium D, 3.0 GHz machine with 2GB RAM. The first table shows the number of facility nodes ($|\mathcal{F}|$), the number of customers ($|\mathcal{D}|$), and the number of Steiner nodes ($|\mathcal{S}|$); because sets are disjoint, $\mathcal{S} = V \setminus (\mathcal{D} \cup \mathcal{F})$. Furthermore, lower bounds (LB) and upper bounds (UB) obtained after running the B&C algorithm for one hour are provided.

The number of nodes in the branch-and-bound tree and the running time of the exact method indicate that the instances with no more than 300 customer- and facility nodes are not trivial, but also not too difficult for the selected method.

For 15 out of 48 benchmark instances – 6 from the first and 9 from the second group – our B&C algorithm finds an optimal solution in less than one hour. Note that for the rest of the instances, we provide upper bounds found by local improvement methods already incorporated in the CPLEX solver 10.0, without using any additional primal heuristics.

The second table shows average and best values (out of 10 runs) obtained from running the VNS strategy with time limit of 1000 seconds. Initial solutions are obtained by randomly selecting 5% of potential facilities.

We provide the best found value of the VNS approach, as well as the best- and average-gaps out of 10 runs (gap_{best} and gap_{avg}, resp.). Standard deviation of the gap is given in column gap_{stddev}. The average number of iterations, and the average running time (in seconds) needed to detect the best solution of each run are given in the last two columns. Note that the gap values are always calculated with respect to the lower bound given in the first column.

The obtained results clearly indicate that the B&C algorithm is not able to handle instances with a large number of customer- or facility nodes within a reasonable amount of time. Already for instances with $\mathcal{F} = \mathcal{D} = 500$, the algorithm is not able to close the optimality gap. The main difficulty for B&C (and exact methods in general) comes from the assignment and capacity constraints. On the other side, for the same set of instances, our VNS approach finds solutions which are within 1% of the lower bound.

[4] http://people.brunel.ac.uk/ mastjjb/jeb/orlib/steininfo.html

Instances		Properties					B&C Results			
UFLP	STP	$\|\mathcal{D}\|$	$\|\mathcal{F}\|$	$\|\mathcal{S} \cup \mathcal{F}\|$	$\|E\|$	$\|\mathcal{S}\|$	LB	UB	Time	B&Bnodes
mp1	c5	200	200	500	625	300	2868.6	2889.6	3602.1	3458
mp2	c5	200	200	500	625	300	2869.5	2869.5	818.4	76
mp1	c10	200	200	500	1000	300	2672.1	2693.9	3602.2	2496
mp2	c10	200	200	500	1000	300	2663.5	2663.5	185.8	62
mp1	c15	200	200	500	2500	300	2636.7	2636.7	332.4	82
mp2	c15	200	200	500	2500	300	2646.5	2646.5	226.3	12
mp1	c20	200	200	500	12500	300	2606.7	2619.7	3603.4	1059
mp2	c20	200	200	500	12500	300	2627.5	2627.5	172.6	102
mq1	c5	300	300	500	625	200	4357.8	4357.8	2740.8	50
mq2	c5	300	300	500	625	200	3770.8	4185.6	3602.8	61
mq1	c10	300	300	500	1000	200	3878.4	3971.2	3602.9	746
mq2	c10	300	300	500	1000	200	3697.6	3778.8	3602.6	1025
mq1	c15	300	300	500	2500	200	3778.9	3868.8	3603.1	500
mq2	c15	300	300	500	2500	200	3612.7	3692.6	3603.0	918
mq1	c20	300	300	500	12500	200	3738.4	3830.5	3603.9	277
mq2	c20	300	300	500	12500	200	3641.7	3689.5	3603.9	268
mp1	d5	200	200	1000	1250	800	2704.9	2704.9	445.6	208
mp2	d5	200	200	1000	1250	800	2759.6	2759.6	3394.3	1869
mp1	d10	200	200	1000	2000	800	2667.9	2678.9	3607.7	2315
mp2	d10	200	200	1000	2000	800	2688.5	2688.5	1777.7	1379
mp1	d15	200	200	1000	5000	800	2639.7	2639.7	1089.7	269
mp2	d15	200	200	1000	5000	800	2648.5	2648.5	174.4	14
mp1	d20	200	200	1000	25000	800	2620.7	2620.7	971.8	410
mp2	d20	200	200	1000	25000	800	2628.5	2628.5	2919.2	1011
mq1	d5	300	300	1000	1250	700	3919.1	3919.1	3230.4	417
mq2	d5	300	300	1000	1250	700	3721.5	3835.7	3609.1	478
mq1	d10	300	300	1000	2000	700	3765.7	3945.1	3609.2	220
mq2	d10	300	300	1000	2000	700	3627.3	3749.3	3608.7	293
mq1	d15	300	300	1000	5000	700	3844.5	3844.5	3574.1	768
mq2	d15	300	300	1000	5000	700	3609.0	3710.7	3609.3	387
mq1	d20	300	300	1000	25000	700	3751.6	3846.8	3614.4	119
mq2	d20	300	300	1000	25000	700	3597.0	3752.0	3614.4	63
gs250a-1	c5	250	250	500	625	250	258300.0	259067.0	3602.8	370
gs250a-2	c5	250	250	500	625	250	257920.0	258714.0	3602.8	141
gs250a-1	c10	250	250	500	1000	250	258062.0	258855.0	3604.1	80
gs250a-2	c10	250	250	500	1000	250	257685.0	258450.0	3606.8	99
gs250a-1	c15	250	250	500	2500	250	257843.0	258307.0	3603.4	205
gs250a-2	c15	250	250	500	2500	250	257524.0	257922.0	3604.3	106
gs250a-1	c20	250	250	500	12500	250	257815.0	258494.0	3603.8	51
gs250a-2	c20	250	250	500	12500	250	257452.0	258253.0	3604.1	22
gs500a-1	c5	500	500	500	625	0	511499.0	756415.0	3607.0	0
gs500a-2	c5	500	500	500	625	0	511485.0	756031.0	3614.2	0
gs500a-1	c10	500	500	500	1000	0	511056.0	666907.0	3607.8	0
gs500a-2	c10	500	500	500	1000	0	511018.0	756031.0	3614.1	0
gs500a-1	c15	500	500	500	2500	0	510733.0	665029.0	3606.6	0
gs500a-2	c15	500	500	500	2500	0	510718.0	671085.0	3606.8	0
gs500a-1	c20	500	500	500	12500	0	510584.0	668346.0	3611.5	0
gs500a-2	c20	500	500	500	12500	0	510559.0	600688.0	3608.7	0

Table 1. Lower and upper bounds of selected benchmark instances obtained by running B&C algorithm with time limit of one hour.

Instances		B&C		VNS					
UFLP	STP	LB	UB-gap	best	gap_{best}	gap_{avg}	gap_{stddev}	Iter	Time
mp1	c5	2868.6	0.7	2923.7	1.9	2.1	0.1	22.1	98.6
mp2	c5	2869.5	**0.0**	2880.4	0.4	0.4	0.0	66.8	363.1
mp1	c10	2672.1	0.8	2799.6	4.8	5.2	0.4	76.0	389.0
mp2	c10	2663.5	**0.0**	2743.3	3.0	3.0	0.0	40.0	187.2
mp1	c15	2636.7	**0.0**	2731.2	3.6	4.0	1.2	78.3	406.4
mp2	c15	2646.5	**0.0**	2801.2	5.8	6.7	1.4	35.7	186.1
mp1	c20	2606.7	0.5	2665.6	2.3	2.5	0.6	50.5	300.3
mp2	c20	2627.5	**0.0**	2700.6	2.8	3.3	0.6	52.6	305.2
mq1	c5	4357.8	**0.0**	4474.6	2.7	2.7	0.0	21.2	177.7
mq2	c5	3770.8	11.0	4185.6	11.0	11.9	1.4	40.0	353.0
mq1	c10	3878.4	2.4	4037.8	4.1	7.4	4.9	40.2	365.0
mq2	c10	3697.6	2.2	3934.9	6.4	7.2	1.7	36.7	340.1
mq1	c15	3778.9	2.4	4058.4	7.4	8.8	1.7	48.5	528.9
mq2	c15	3612.7	2.2	3743.0	3.6	3.6	0.1	25.9	250.5
mq1	c20	3738.4	2.5	3996.7	6.9	8.2	2.0	38.1	401.2
mq2	c20	3641.7	1.3	3782.8	3.9	5.1	0.6	35.8	375.5
mp1	d5	2704.9	**0.0**	2715.5	0.4	2.2	1.0	23.5	402.9
mp2	d5	2759.6	**0.0**	2766.2	0.2	1.4	1.2	31.7	482.3
mp1	d10	2667.9	0.4	2728.8	2.3	3.4	1.3	23.1	366.8
mp2	d10	2688.5	**0.0**	2691.5	0.1	0.6	0.8	13.6	365.0
mp1	d15	2639.7	**0.0**	2694.6	2.1	2.3	0.7	23.0	328.5
mp2	d15	2648.5	**0.0**	2705.6	2.2	2.8	1.3	20.4	379.0
mp1	d20	2620.7	**0.0**	2634.9	0.5	0.9	0.8	24.0	453.4
mp2	d20	2628.5	**0.0**	2629.5	0.0	0.0	0.0	15.4	321.9
mq1	d5	3919.1	**0.0**	4012.3	2.4	6.6	4.3	22.1	508.1
mq2	d5	3721.5	3.1	4007.3	7.7	11.8	2.5	21.4	460.7
mq1	d10	3765.7	4.8	3975.0	5.6	7.0	1.5	15.3	511.1
mq2	d10	3627.3	3.4	3796.4	4.7	7.0	1.4	18.0	593.8
mq1	d15	3844.5	**0.0**	3857.6	0.3	7.0	8.6	24.5	652.8
mq2	d15	3609.0	2.8	3798.9	5.3	8.9	3.8	27.9	627.0
mq1	d20	3751.6	2.5	3984.6	6.2	9.1	2.1	17.9	490.4
mq2	d20	3597.0	4.3	3749.0	4.2	5.4	1.5	17.6	495.5
gs250a-1	c5	258300.0	0.3	258592.0	0.1	0.3	0.1	65.2	523.5
gs250a-2	c5	257920.0	0.3	258145.0	0.1	0.3	0.1	63.3	458.3
gs250a-1	c10	258062.0	0.3	258555.0	0.2	0.4	0.1	74.9	668.5
gs250a-2	c10	257685.0	0.3	258223.0	0.2	0.5	0.1	60.6	341.7
gs250a-1	c15	257843.0	0.2	258268.0	0.2	0.3	0.1	58.9	548.6
gs250a-2	c15	257524.0	0.2	257819.0	0.1	0.3	0.1	88.0	598.3
gs250a-1	c20	257815.0	0.3	258239.0	0.2	0.2	0.0	87.2	598.0
gs250a-2	c20	257452.0	0.3	257776.0	0.1	0.3	0.1	106.0	697.4
gs500a-1	c5	511499.0	47.9	513871.0	0.5	0.6	0.1	44.8	838.1
gs500a-2	c5	511485.0	47.8	514124.0	0.5	0.6	0.0	55.2	845.9
gs500a-1	c10	511056.0	30.5	513429.0	0.5	0.5	0.0	50.4	881.1
gs500a-2	c10	511018.0	47.9	513543.0	0.5	0.5	0.0	58.5	939.5
gs500a-1	c15	510733.0	30.2	513165.0	0.5	0.6	0.0	48.9	928.0
gs500a-2	c15	510718.0	31.4	513108.0	0.5	0.6	0.1	41.7	871.2
gs500a-1	c20	510584.0	30.9	512764.0	0.4	0.5	0.0	48.1	943.8
gs500a-2	c20	510559.0	17.7	512560.0	0.4	0.5	0.1	51.4	906.0

Table 2. Comparison of the VNS with lower and upper bounds obtained by B&C.

For the instances of the first two groups, the algorithm does not always reach the optimal solution, but the average gaps and their standard deviation indicate a stable performance and the robustness of the approach.

The incorporation of the VNS method as a primal heuristic within the B&C framework seems a promising direction for further research. The synergy effect of this combination may bring advantages to both approaches: good starting solutions obtained by rounding fractional solutions for the VNS, on one side, and fast high-quality upper bounds for B&C, on the other side.

References

1. R. Battiti and G. Tecchiolli. The reactive tabu search. *ORSA Journal on Computing*, 6(2):126–140, 1994.
2. D. Ghosh. Neighborhood search heuristics for the uncapacitated facility location problem. *European Journal of Operations Research*, 150:150–162, 2003.
3. A. Gupta, A. Kumar, and T. Roughgarden. Simpler and better approximation algorithms for network design. In *STOC*, pages 365–372. ACM, 2003.
4. G. Harm and P. V. Hentenryck. A multistart variable neighborhood search for uncapacitated facility location. In *Proceedings of MIC2005: The Sixth Metaheuristics International Conference*. 2005.
5. M. Hoefer. Experimental comparison of heuristic and approximation algorithms for uncapacitated facility location. In *Proceedings of the Second International Workshop on Experimental and Efficient Algorithms (WEA 2003)*, pages 165–178. 2003.
6. D. R. Karger and M. Minkoff. Building Steiner trees with incomplete global knowledge. In *FOCS*, pages 613–623, 2000.
7. S. Khuller and A. Zhu. The general steiner tree-star problem. *Information Processing Letters*, 84(4):215–220, 2002.
8. T. Koch and A. Martin. Solving Steiner tree problems in graphs to optimality. *Networks*, 32:207–232, 1998.
9. M. Koerkel. On the exact solution of large-scale simple plant location problems. *European Journal of Operations Research*, 39:157–173, 1989.
10. J. Kratica, D. Tošić, V. Filipović, and I. Ljubić. Solving the simple plant location problem by genetic algorithms. *RAIRO - Operations Research*, 35(1):127–142, 2001.
11. Y. Lee, 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.
12. I. Ljubić. *Exact and Memetic Algorithms for Two Network Design Problems.* PhD thesis, Faculty of Computer Science, Vienna University of Technology, November 2004.
13. I. Ljubić, R. Weiskircher, U. Pferschy, G. Klau, P. Mutzel, and M. Fischetti. An algorithmic framework for the exact solution of the prize-collecting Steiner tree problem. *Mathematical Progamming, Series B*, 105(2-3):427–449, 2006.
14. K. Mehlhorn. A faster approximation for the Steiner problem in graphs. *Information Processing Letters*, 27:125–128, 1988.
15. L. Michel and P. V. Hentenryck. A simple tabu search for warehouse location. *European Journal of Operational Research*, 157(3):576–591, 2004.
16. P. Nuggehalli, V. Srinivasan, and C.-F. Chiasserini. Energy-efficient caching strategies in ad hoc wireless networks. In *MobiHoc*, pages 25–34, 2003.

17. C. Swamy and A. Kumar. Primal-dual algorithms for connected facility location problems. *Algorithmica*, 40:245–269, 2004.
18. 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.