Location Routing Problems on Simple Graphs

Julián Aráoz *, Elena Fernández †, and Salvador Rueda ‡

1 Department of Statistics and Operation Research, Universitat Politècnica de Catalunya-BcnTech, Spain

Abstract

This paper addresses combined location/routing problems defined on trees. Several problems are studied, which consider service demand both at the vertices and the edges of the input tree. Greedy type optimal heuristics are presented for the cases when all vertices have to be visited and facilities have no set-up costs. Facilities set-up costs can also be handled with interchange heuristics of low order. More general problems where not all vertices have to be necessarily visited require more sophisticated solution methods. Low order optimal algorithms based on recursions are proposed for such problems.

1 Introduction

Location and routing problems are among core problems in combinatorial optimization. Both types of problems have received the attention of a large number of researchers and have multiple applications fields, for instance in logistics and telecommunications, where difficult optimization problems arise. It is well-known that location and routing decisions are most often closely interrelated. Indeed there are a number of location applications in which the selected locations will become the depots for the routes that will serve the demand of a given set of customers. However, location problems frequently ignore the tactical or operational routing decisions and focus on the strategic location/allocation decisions. On the other hand, in routing problems it is typically assumed that the depots for the routes are set in advance, despite the enormous influence that the location of such depots may have in the design of efficient service routes. Therefore, because of the impact that joint location/routing decisions may have on the overall costs, a joint location/routing perspective is fully justified regardless of the increase in the difficulty of the resulting problem. Different types of formulations and solution techniques have been proposed for various location/routing problems (LRPs) under different modeling assumptions [5, 11, 2, 4]. The interested reader is addressed to [8, 10, 9, 12, 1] for overviews and surveys on the topic.

The state of the art on LRPs deals with generic graphs where both location and routing decisions typically produce \(NP\)-hard optimization problems. To the best of our knowledge, the topology of the graph where problem is defined has not been exploited so far. In contrast, the topology of the network has been exploited quite extensively to derive polynomial time solution algorithms for a number of location/allocation problems, which are \(NP\)-hard in the general case. This is particularly true for the case of tree networks [6, 7, 15, 16, 14, 13], and is the focus of this work, in which we give polynomial time optimal algorithms for several LRPs formulated on a tree. These results can be extended to cacti (connected graphs in which any two simple cycles have at most one vertex in common), and can be used as the basis for heuristics for other LRPs on more general graphs.

We work on an undirected connected graph that we assume is a tree, with a cost function on the edge set and a given set of users with service demand, which can be located at both the vertices and the edges of the graph. We assume there are no capacity constraints, neither for the demand that can be allocated to each open facility, nor for the demand that can be served by a route.
study several types of LPRs, which mainly differ from each other on the characteristics of the set of demand customers. Another possible difference among problems is whether or not open facilities incur set-up costs. In all cases, the LRP that we study consists of (i) selecting the best location for a set of facilities at vertices of the graph, (ii) allocating each demand customer to one selected facility, and (iii) designing a set of closed routes through the given set of customers, each of them using as depot one of the selected facilities, of minimum total cost.

Once the location of the facilities and the allocation of customers to the selected facilities are determined, the optimal routing problem can be decomposed into a number of smaller subproblems, one associated with each facility, which becomes the depot for its allocated customers. Indeed, the routes must visit vertices or traverse edges, depending on the type of demand. Since we work on a tree and there are no capacity constraints, in all cases all optimal routes are bipaths where the edges that are used are traversed exactly twice.

Broadly speaking, LRP s in which all the vertices of the original graph have to be visited reduce to finding a suitable partition of the original vertex set. The edges connecting vertices in the same set of the partition are the ones that will be traversed twice in optimal solutions. Alternatively, one can find the edges connecting vertices that do not belong to the partition and thus will not be traversed. In the absence of facilities set-up costs, such a problem can be solved with a simple greedy algorithm, both for demand at the vertices or the edges of the tree. Solutions satisfying optimality conditions can also be obtained in polynomial time for the case with set-up costs.

We also study LRP s where not all vertices have demand or when the graph induced by demand edges does not span the complete vertex set. Now the interpretation of a solution as a partition of the vertex set does not hold. Furthermore, such LRP s can no longer be solved with a greedy heuristic. While in the case of general graphs, one can get rid of non-demand vertices by defining edges associated with shortest paths, this cannot be done in the case of a tree without breaking the tree-structure of the original graph, unless all non-demand vertices have degree two. Hence, non-demand vertices cannot be eliminated from the original graph. The fact that it is not known a priori whether or not a non-demand vertex will be covered in an optimal solution, makes these LRP s substantially more difficult than when all vertices must be necessarily visited. As we will see, a dynamic programming algorithm can optimally solve LRP s of this type in polynomial time.

The paper is structured as follows. Section 2 introduces the notation and recalls some well-known definitions and results. Section 3 deals with LRP s without set-up costs, in which all vertices have to bee visited, both for the case of demand at the vertices or the edges of the tree. The optimality conditions for the cases with set-up cost are given in Section 4. Section 5 studies LRP s in which all the vertices of the original graph do not necessarily have to be visited, and gives a polynomial time dynamic programming algorithm for these problems. The paper ends in Section 7 with some conclusions and avenues for research.

2 Preliminaries

Next we introduce the notation that we will use in this work. We also recall some definitions and well known results.

Let \( T = (V, E) \) denote an undirected tree with vertex set \( V, |V| = n \), and edge set \( E, |E| = n - 1 \).

- For any subgraph \( H \) of \( T \) we denote by \( V(H) \) the vertex set of \( H \) and by \( E(H) \) the edge set of \( H \), in particular, \( V = V(T), E = E(T) \).

- For a given vertex set \( S \subset V \), we denote by \( E(S) = \{ e \in E | e = (u, v), u, v \in S \} \), the set of edges with both vertices in \( S \). For any vertex set \( S \subseteq V \) we denote by \( G[S] = (S, E(S)) \) the induced graph.

- For any edge set \( F \subseteq E \), the set of vertices incident to edges in \( F \) is denoted by \( V(F) \). The subgraph induced by \( F \) is \( G(F) = (V(F), F) \).

- For any nonempty subset of vertices \( S \subset V \), \( \delta(S) = \{ e \in E | e = (u, v), u \in S, v \in V \setminus S \} = \delta(V \setminus S) \), denotes the set of edges in the cut between \( S \) and \( V \setminus S \).
For a singleton \( \{v\} \subset V \) we do not use the brackets and simply write \( \delta(v) \equiv \delta(\{v\}) \) to denote the set of edges \( (v, u), v \neq u, \) incident to \( v. \)

We use the standard compact notation \( f(A) \equiv \sum_{e \in A} f_e \) when \( A \subseteq E, \) and \( f \) is a vector or a function defined on \( E. \)

**Definition 1 Location-Routing Problem on a tree \( T. \)**

Consider a tree \( T = (V, E), \) with a setup cost function on the vertices \( f : V \to \mathbb{R} \) and a cost function on the edges \( c : E \to \mathbb{R}_+. \) Let also \( p > 0 \) be a given integer number, and \( D \subseteq V \) and \( R \subseteq E \) two given demand sets of vertices and edges, respectively. The \( p \)-Location-Routing Problem on \( T \) with vertex demand \( D \) and edge demand \( R \) is to:

1. find a set of locations \( I \subset V \) with \( |I| = p \) to place a facility at each of them,
2. allocate each vertex of \( D \) and each edge of \( R \) to one facility of \( I, \) and
3. define one route from each selected facility \( i \in I, \) visiting all its allocated vertices and traversing all its allocated edges,

of minimum total cost. The cost of a solution is the sum of the setup costs of the facilities of \( I, \) plus the sum of the costs of the edges of the routes. The above LRP is denoted by \( (T, f, c, p, D, R). \)

The version of the problem in which at most \( p \) facilities must be opened is denoted by \( (T, f, c, p^\leq, D, R). \)

The more general version in which the maximum number of facilities is not set is, thus, \( (T, f, c, n^\leq, D, R). \)

Since an optimal solution to \( (T, f, c, p^\leq, D, R) \) can be obtained by identifying the best solution among the optimal solutions to problems \( (T, f, c, r, D, R), \) with \( 1 \leq r \leq p, \) an exact algorithm for \( (T, f, c, p, D, R) \) also gives an exact algorithm for \( (T, f, c, p^\leq, D, R), \) whose complexity increases by a factor of \( p \) with respect to the complexity of the algorithm for \( (T, f, c, p, D, R). \) In the following we focus on the \( p \)-Location-Routing Problem on \( T, \) \( (T, f, c, p, D, R). \)

A solution to \( (T, f, c, p, D, R) \) is associated with a family of vertex disjoint connected subtrees \( S = \{T^k\}_{k \in K}, \) with \( |K| \leq p, \) each of them hosting at least an open facility. That is, a solution to \( (T, f, c, p, D, R) \) is associated with a forest \( T^k = (V^k, E^k), k \in K \) with \( V^k \subseteq V \) and \( E^k \subseteq E, |V^k| \geq 1. \)

The subtrees \( T^k, k \in K \) will also be called \( S \)-components. For \( k \in K, I(k) \subseteq V^k \) denotes the set of locations for the facilities in \( S \)-component \( T^k, \) and \( i(k) \) an arbitrarily selected lowest setup cost vertex in \( T^k, \) i.e. \( i(k) \in \arg\min\{f_i | i \in V^k\}. \) Abusing notation we will refer to solution \( \{T^k\}_{k \in K} \) instead of to solution associated with \( \{T^k\}_{k \in K}. \)

A solution \( \{T^k\}_{k \in K} \) contains, for each \( i \in I(k), k \in K, \) a minimum cost route with depot at facility \( i \) visiting all its allocated demand vertices and traversing all its allocated demand edges. Since \( T \) is a tree, each such route is a bipath replicating all the edges of \( E^k, k \in K. \) We will indistinctively use \( S = \{T^k\}_{k \in K} \) and \( S = (I^S, \overline{E}^S), \) where \( I^S = \bigcup_{k \in K} I(k) \) is the set of open facilities, and \( \overline{E}^S = \bigcup_{k \in K} E^k \) the set of edges traversed in the routes (note that \( |\overline{E}^S| = n - |K|). \)

The cost of a solution \( S \) is thus \( C(S) = f(I^S) + 2c(\overline{E}^S). \) When the solution \( S \) is clear from the context will drop the superindex.

**Remark 1**

- As it will become apparent later on, it is possible that \( |K| < p \) in some optimal solution. In that case, since exactly \( p \) facilities have to be located, \( |I(k)| > 1 \) for some \( k \in K. \) When \( |K| = p \) exactly one facility is located at each \( S \)-component, i.e. \( |I(k)| = 1 \) for all \( k \in K. \) Then, \( I(k) = \{i(k)\}. \)
- Taking into account the minimization objective function, once the \( S \)-components are known, the sets \( I(k) \) indicating the optimal locations for the facilities can be found in a simple way: one facility is first located at each \( i(k), k \in K, \) the lowest setup cost facility of each \( S \)-component. The remaining \( p - |K| \) facilities are then located in a greedy fashion by increasing values of their setup cost.
Since the optimal locations for the facilities can be found in a greedy way once the $S$-components are known, the main difficulty in LRPs on trees is thus identifying the set of edges that define the $S$-components.

3 Location/Routing on Trees without Set-up Costs

In this section we address $(T,0,c,p,D,R)$. That is, we assume there are no setup costs, i.e. $f_i = 0$ for all $i \in V$. Because exactly $p$ facilities are located, $(T,0,c,p,D,R)$ also solves the case when all facilities have the same set-up cost.

We assume that all vertices have to be visited, that is, $D \cup V(R) = V$. This means that for $i \in V$ either $i \in D$ or $i$ is the end-vertex of some edge in $R$. Under this assumption, we first address the Vertex Location/Routing case in which demand is located only at the vertices of the graph (i.e. $D = V$, $R = \emptyset$). For this problem we give a simple greedy algorithm and prove its optimality. Then we see that the Edge Location/Routing case when demand is located only at the edges (i.e. $D = \emptyset$, $V(R) = V$) can be reduced to Vertex Location/Routing and finally we indicate how to handle the general case with both $D \neq \emptyset$ and $R \neq \emptyset$. The more general problem in which all vertices do not have to be visited necessarily, i.e. $D \cup V(R) \subset V$, will be studied in Section 5.

3.1 Vertex Location/Routing: $p$-VLRP

When $D = V$ and $R = \emptyset$ an optimal solution to $(T,0,c,p,V,\emptyset)$ exists with exactly $p$ $S$-components. Indeed, if the forest associated with a feasible solution has less than $p$ components, removing any edge from the solution produces a feasible solution with a smaller cost. Hence, in this section we look for solutions with $|K| = p$ or, equivalently, with $n - p$ edges, i.e. $|E| = n - p$. As mentioned, $|K| = p$ implies that $|I(k)| = 1$ for all $k \in K$. Thus, $V^k$ contains exactly the customers allocated to facility $i(k)$. Note that it is possible that $E^k = \emptyset$ for some $k \in K$; in this case, the associated $V^k$ is a singleton. By analogy with location/allocation problems, this particular case will be referred to as $p$-VLRP.

In contrast to location/allocation problems on trees, in an optimal solution to a $p$-VLRP, a vertex is not necessarily allocated to its closest open facility. Figure 1 illustrates this observation. Figure 1(a) depicts the original tree; values next to the edges indicate their costs. In Figures 1(b)-(c) we assume that $p = 2$ and two facilities are located at vertices 1 and 6, respectively. As depicted in Figure 1(b), the optimal location/allocation solution is to allocate $\{1, 2, 3, 4\}$ to facility 1, and $\{5, 6\}$ to facility 6. The location/allocation objective function value of this solution is $1+1+2+3=7$. The routing value of this allocation for the 2-VLRP is $2 \times (1 + 1 + 2) + 2 \times 3 = 14$. However, the optimal 2-VLRP decision, which is depicted in Figure 1(c), is to allocate $\{1, 2, 3, 4, 5\}$ to facility 1, and $\{6\}$ to facility 6. The location/allocation objective function value of this solution is $1+1+2+4=8$. The routing value of this allocation for the 2-VLRP is $2(1 + 1 + 2 + 2) = 12$. Such a solution can be obtained by simply removing the largest edge.

Figure 1: Optimal location/allocation and location/routing on a tree
Since any subset $E \subseteq E$ with $|E| = n - p$ induces a solution, solutions to $p$-VLRP can be obtained by simply removing $p - 1$ edges from $E$. Moreover, since there are no setup costs, if the removed edges are the $p - 1$ largest ones, the resulting solution will be optimal. This is the idea of Algorithm 1, which obtains a solution by removing edges in a greedy fashion by decreasing values of their costs.

**Algorithm 1**

0. $E^0 \leftarrow E; t \leftarrow 1$

1. $e^t \leftarrow \arg \max \{c_e \mid e \in E^{t-1}\}$ (with ties arbitrarily broken).

2. $E^t \leftarrow E^{t-1} \setminus \{e^t\}$.

3. $t \leftarrow t + 1$ if $(t < p)$ goto 1.

4. end

It is clear that the solution produced by the above algorithm is optimal to $p$-VLRP. The complexity of Algorithm 1 is $O(pn)$. An alternative optimal algorithm for the $p$-VLRP is to start with an empty solution and to iteratively incorporate $n - p$ edges greedily selected by increasing values of their costs. Our preference for Algorithm 1 is justified, as the complexity of such a constructive algorithm is $O((n-p)n)$, which, in practice, is expected to be larger than $O(pn)$. Indeed, if edges were sorted by non-increasing or decreasing costs initially, the complexity of the above algorithm 1 would be $O(n\log n)$, which, in practice, is expected to be larger than $O(pn)$.

### 3.2 Edge Location Routing on trees

Below we study the $(T, 0, c, p, \emptyset, R)$ under the assumption that $V(R) = V$. This problem is referred to as $p$-ELRP. Observe that now any feasible solution must traverse all edges of $R$. Thus $2c(R)$ is a fixed minimum traveling cost that will be incurred by all solutions. Let $R^i$, $i = 1, \ldots, r$, denote the connected components induced by $R$. Since there are no setup costs, when $r \leq p$ the solution where each $R^i$, $i \in \{1, \ldots, r\}$ defines an $S$-component, i.e., $T^k = G(R^k)$, $k \in \{1, \ldots, r\}$, is optimal. When $r < p$ the number of $S$-components defining this solution is smaller than $p$, so some $T^k$ will host more than one facility. The location of the facilities can be arbitrarily selected, provided that $|I(k)| \geq 1$ for all $k \in K$, as all setup costs are zero.

In order to obtain a feasible solution when $r > p$, some $R^i$'s have to be connected with each other until the number of $S$-components is reduced to $p$. For this, $r - p$ edges from $E \setminus R$ must be added to $R$. Consider the graph $T^R = (V^R, E^R)$ where $V^R$ contains a vertex associated with each $R^i$, $i \in \{1, \ldots, r\}$, and $E^R = E \setminus R$ are the non-demand edges of the tree $T$. Note that $T^R$ is a tree with $|V^R| = r$ and $|E^R| = n - 1 - |R|$. Consider also the cost vector $c^R$, where each non-demand edge $e \in E^R$ inherits its cost from $T$ ($c^R_e = c_e$). It is clear that any optimal solution to the $p$-VLRP $(T^R, 0, c^R, p, V^R, \emptyset)$ is also an optimal solution to $(T, 0, c, p, \emptyset, R)$. Therefore, Algorithm 2, which is the adaptation of Algorithm 1 to $(T^R, 0, c^R, p, V^R, \emptyset)$ is an optimal algorithm for the Edge Location/Routing problem.

**Algorithm 2**

0. $E^0 \leftarrow E \setminus R; t \leftarrow 1$

1. $e^t \leftarrow \arg \max \{c_e \mid e \in E^{t-1}\}$ (with ties arbitrarily broken).

2. $E^t \leftarrow E^{t-1} \setminus \{e^t\}$.

3. $t \leftarrow t + 1$ if $(t < p)$ goto 1.

4. end
The complexity of Algorithm 2 is $\mathcal{O}(pn)$. Depending on the number of components induced by the demand edges ($r$), $r - p + 1$ can be smaller than $p - 1$. In this case, the constructive version of the greedy algorithm would be preferred to Algorithm 2 for $(T, R', 0, c, p, V, \emptyset)$, since it would request a smaller number of iterations. In this case the initial set of edges will consist of all demand edges, i.e. $E^0 \leftarrow R$.

The general case $(T, 0, c, p, D, R)$ with $D \neq \emptyset$ and $R \neq \emptyset$ can be solved as follows. If $D \subseteq V(R)$, then $(T, 0, c, p, D, R)$ reduces to $(T, 0, c, p, \emptyset, R)$. Thus, let us assume that $V \setminus V(R) \subseteq D$ and let $s = |V \setminus V(R)|$. Consider the tree $T^R = (V^R, E^R)$ where now $V^R$ contains a vertex associated with each vertex $i \in V \setminus V(R)$ and a vertex associated with each $R'$, $i \in \{1, \ldots, r\}$, and $E^R = E \setminus R$ are the non-demand edges of the tree $T$. Now $|V^R| = s + r$ and $|E^R| = (n - 1) - |R|$. Consider as before the cost vector $c^R$, where each non-demand edge $e \in E^R$ inherits its cost from $T$ $(c^R_e = c_e)$. Since any optimal solution to the $p$-VLR ($T^R, 0, c^R, p, V^R, \emptyset$) is also optimal to $(T, 0, c, p, D, R)$, the general $(T, 0, c, p, D, R)$ can also be solved with Algorithm 2. The following result thus follows:

**Corollary 1** $(T, 0, c, p, D, R)$ can be solved in polynomial time.

4 Location/Route on Trees with Set-up Costs

4.1 $p$-VLRP with setup costs

Next we address the case $(T, f, c, p, V, \emptyset)$ with setup costs $f$, which are not necessarily all the same. Indeed Algorithm 1 produces a feasible solution also for this case. Now the outcome of Algorithm 1 may fail to produce an optimal solution, even if a facility is placed at a lowest setup cost vertex in each $S$-component, i.e. $i(k) \in \arg \min\{f_i | i \in V^k\}$ for all $k \in K$. This is illustrated in Figure 2, which uses the same input tree as Figure 1, but now we assume that $f_1 = 1$, $f_4 = 3$ and $f_6 = 6$, and $f_i = 100$ for all $i \neq 1, 4, 6$. Figure 2(a) depicts the outcome of Algorithm 1, whereas Figure 2(b) depicts an optimal solution. As can be seen, the outcome of Algorithm 1 is $E^1 = \{(1, 2), (1, 3), (1, 4), (4, 5)\}, E^2 = \emptyset$, which induces the vertex set partition $V^1 = \{1, 2, 3, 4, 5\}$, $V^2 = \{6\}$, with set of open facilities $I = \{i(1), i(2)\}$ with $i(1) = 1$ and $i(2) = 2$. The value of this solution is $(1 + 3) + 2 \times (1 + 1 + 2 + 2) = 19$. The solution depicted in Figure 2(b) is associated with the edge sets $E^1 = \{(1, 2), (1, 3)\}, E^2 = \{(4, 5), (5, 6)\}$, with vertex set partition $V^1 = \{1, 2, 3\}$, $V^2 = \{4, 5, 6\}$, and set of open facilities $I = \{1, 4\}$ $(i(1) = 1$ and $i(2) = 4)$. The total cost of this solution is $(1 + 3) + 2 \times (1 + 1 + 2 \times (2 + 3) = 18$. Note that the solution of Figure 2(b) can be obtained from the solution of Figure 2(a) by interchanging facility 6 by facility 1. Closing facility 6 requires to add edge $(5, 6)$. In its turn, opening facility 1 requires to remove edge $(1, 4)$.

The above example suggests that improvements to a given solution $S = (I, \overline{E})$ may be obtained by closing one open facility in $I$ and opening one non-open facility in $V \setminus I$. The effect of such an interchange is twofold. On the one hand the $S$-component $k$ of the newly open facility $i \in V_k$ splits in two subtrees. For this one edge in the path connecting $i$ and $i(k)$ must be removed. On the
other hand, the component $k'$ of the facility $j = i(k')$ that is closed must be merged with some other component. For this one edge in $\delta(V^{k'})$ must be added. If the opening and closing facilities are given, the best possible move, in terms of the objective function value, is clear: select as leaving edge the one with maximum cost in the path connecting $i$ and $i(k)$, $P_{i,i(k)}$, and select as entering edge the one of minimum cost in $\delta(V^{k'})$. That is $e^+_S(i) = \arg \max \{c_e : e \in P_{i,i(k)}\}$ and $e^-_S(j) = \arg \min \delta(V_{k'})$.

The increase in the objective function implied by opening facility $C$ is thus, $\Delta^+_S(i) = f_i - 2c^+_S(i)$; whereas the decrease due to closing facility $j$ is $\Delta^-_S(j) = f_j - 2c^-_S(j)$. Hence, interchanging facilities $i \in V \setminus I$ and $j \in I$ in solution $S$ yields an improved solution if and only if $\Delta^+_S(i) - \Delta^-_S(j) < 0$.

**Definition 2** A solution $S = (I, E)$ is of minimum cost for $I$, if $c(E') \geq c(E)$ for any solution $S' = (I, E')$.

**Remark 2**

1. Let $S = (I, E)$ be the solution to $(T, f, c, p, V, \emptyset)$ obtained with Algorithm 1. Then $S$ is of minimum cost for $I$.

2. Let $S = (I, E)$ be a minimum cost solution for $I$, and $S'$ the solution obtained after opening facility $i \in V \setminus I$, closing facility $j \in I$ and interchanging edges $e^+_S(i)$ and $e^-_S(j)$ as explained above. Then

   (a) $S'$ is of minimum cost for $I' = I \cup \{i\} \setminus \{j\}$.

   (b) In the reverse interchange of $j$ and $i$ in $S'$, $e^-_S(i) = e^+_S(i)$ and $e^-_S(j) = e^+_S(j)$. Thus, $\Delta^+_S(i) - \Delta^-_S(j) = [-\Delta^+_S(i) - \Delta^-_S(j)]$.

   (c) $\Delta^+_S(i) \geq \Delta^+_S(i)$ for all $i \in V \setminus \{I' \setminus \{i\}\}$ and $\Delta^-_S(j) \leq \Delta^-_S(j)$ for all $j \in I' \setminus \{i\}$.

**Proposition 1** [Facility interchange optimality] A solution to $(T, f, c, p, V, \emptyset)$, $S = (I, E)$ of minimum cost for $I$, is optimal if and only if $\Delta^+_S(i) - \Delta^-_S(j) \geq 0$ for all $i \in V \setminus I$ and $j \in I$.

**Proof:**

$\Rightarrow$: From the above analysis it is clear that if there exist $i \in V \setminus I$, $j \in I$, such that $\Delta^+_S(i) - \Delta^-_S(j) < 0$, then the current solution is not optimal.

$\Leftarrow$: To see the reverse implication, let $S = (I, E)$ be a solution of minimum cost for $I$ such that $\Delta^+_S(i) - \Delta^-_S(j) \geq 0$ for all $i \in V \setminus I$, $j \in I$, and suppose $S$ is not optimal. Let $\delta^+ = \min_{i \in V \setminus I} \Delta^+_S(i)$ and $\delta^- = \max_{j \in I} \Delta^-_S(j)$. Let also $S^* = (I^*, E^*)$ be an optimal solution, and consider the subset of facilities $I^* \setminus I$.

Indeed, $S^*$ can be obtained from $S$ by performing $|I^* \setminus I|$ interchanges. At each interchange one facility in $I^* \setminus I$ will open and one facility in $I \setminus I^*$ will close. By Remark 2(c) the sequence of the values $\Delta^+_S(i)$ obtained at the subsequent iterations will be non decreasing whereas the sequence of the values $\Delta^-_S(j)$ will be non increasing. Hence, $C(S^*) \geq C(S) + [\delta^- - \delta^+] \geq C(S)$ for any solution $S'$ obtained at the different iterations. In particular, $C(S^*) \geq C(S)$ contradicting the hypothesis that $S$ is not optimal.

Taking into account Proposition 1 the algorithm below produces an optimal solution starting from the solution produced by Algorithm 1.

**Algorithm 3**

0. Let $S = (I, E)$ be the solution produced by Algorithm 1;

   terminate $\leftarrow$ false

   while (not terminate) do

   1. Compute $i^+ \leftarrow \arg \max \{\Delta^+_S[i] : i \in V \setminus IS\}$

      (this requires to identify $e^+_S(i)$ for all $i \in V \setminus I$).

7
Figure 3: \( p \)-ELRP example with \( r = p \) and optimal solution with edge in \( E \setminus R \)

2. Compute \( j^- \leftarrow \arg \min \{ \Delta_S^- (j) : j \in I \} \)  
   (this requires to identify \( e_S^- (j) \) for all \( j \in I \)).

3. if\( (\Delta_S^+ (i^+) - \Delta_S^- (j^-) \geq 0) \) then
   terminate \( \leftarrow \text{true} \)
   else
   \( I \leftarrow I \cup \{ i^+ \} \setminus \{ j^- \} \)
   \( E \leftarrow E \cup \{ e_S^+ (i^+) \} \setminus \{ e_S^- (j^-) \} \)
   endif

4. end

In Step 1 the selection of the candidate facility to enter is \( O(n - p)^2 \), as there are \( n - p \) candidate entering facilities and \( n - p \) potential leaving edges. In Step 2 the selection of the candidate facility to leave is \( O(p^2) \), as there are \( p \) candidate leaving facilities and \( p - 1 \) potential leaving edges. Thus each iteration in the while loop is dominated by \( O(n^2) \). Selecting the entering and leaving facilities with the greedy criterion, guarantees that no facility that leaves the solution at a given iteration will enter the solution at a later iteration. Hence, the overall number of iterations is bounded by \( p \). Therefore,

**Proposition 2** The complexity of the facility interchange starting with the solution to \((T, f, c, p, V, \emptyset)\) obtained with Algorithm 1 is \( O(pm^2) \)

### 4.2 \( p \)-ELRP with setup costs

Next we address the case \((T, f, c, p, \emptyset, R)\) with setup costs \( f \), which are not necessarily all the same. For similar reasons to the \( p \)-VLRP, the outcome of the greedy heuristic of Algorithm 2 may fail to produce an optimal solution, and interchanging closed and open facilities may yield improved \( p \)-ELRP solutions. Of course, the edges involved in such interchanges must now be restricted to non-demand edges. This may cause additional difficulties as no feasible facility interchange may exist even if the current solution is not optimal. The solution of Figure 3.(b) is an example: since the solution contains only demand edges, none of the edges in the solution may leave it, so no feasible facility interchange exists. Still the solution is not optimal. This situation is related to the fact that, in contrast to the cases studied so far, the number of trees in optimal \( p \)-ELRP solutions is not known in advance. This is again illustrated in the example of Figure 3, with two demand components \((r = 2)\) and \( p = 2 \), whose optimal solution, depicted in Figure 3.(c), consists of a unique \( S \)-component \( T^1 = (V^1, E^1) \) with \( V^1 = V \), \( E^1 = E \) and \( I(1) = \{ 5, 6 \} \).

We can thus conclude that, when the number of subtrees produced by Algorithm 2 (which is always \( p \)) does not coincide with the number of subtrees in an optimal solution, Algorithm 3 will terminate with a non-optimal solution, as facilities interchanges as presented in Section 4.1 produce solutions with a fixed number of \( S \)-components.

A natural way overcome this situation is to allow modifications that involve a change in the number of subtrees induced by the solutions. Next we consider the change in a solution due to adding one
edge. Consider a feasible solution $S = (I, E)$, where again we assume that the facilities of $I$ have been chosen as indicated in Remark 1 and $S$ is of minimum cost for $I$. Let $e \in E \setminus E$ be a non-demand edge connecting $S$-components $T^k$ and $T^{k'}$ where, without loss of generality, we assume that $f_{i(k')} \geq f_{i(k)}$. The effect of adding edge $e$ to the solution $S$ without removing any edge from the solution is the following. The set of edges in the solution is extended to $E \cup \{e\}$ and $S$-components $T^k$ and $T^{k'}$ merge, i.e. $K \leftarrow K \setminus \{k'\}$. Thus, if $|I(k')| > 1$ the set of open facilities will remain unchanged. (This is a consequence of Remark 1, since $|I(k')| > 1$ means that all open facilities in this $S$-component obey to the greedy criterion.) Thus, $I(k) \leftarrow I(k) \cup I(k')$ and the variation in objective function value due to adding edge $e$ to solution $S$ is $\Theta_S[e] = 2c_e \geq 0$. However, when $|I(k')| = 1$, facility $i(k')$ does not necessarily satisfy the greedy criterion, so it will close if $f_{i(k')} > f_{i^+}$ where $i^+$ is the index of the cheapest vertex among the ones where no facility sits in solution $S$, i.e. $i^+ \in \arg\min\{f_i \mid i \in V \setminus T\}$. In this case, the set of open facilities of the $S$-component containing vertex $i^+$, say $k \in K$, is updated to $I(\hat{k}) \cup \{i^+\}$. Now, the variation in objective function value is $\Theta_S[e] = 2c_e + f_{i^+} - f_{i(k')}$, which can be negative. Hence, adding edge $e \in E \setminus E$ to solution $S$ yields an improved solution if and only if $\Theta_S[e] < 0$. We thus have the following result in which the proof omitted, as it is largely based on that of Proposition 1 and the above analysis.

**Proposition 3** A solution $S = (I^S, E^S)$ to $(T, f, c, p, \emptyset, R)$ is optimal if and only it holds that:

- $\Delta^+_S(i) - \Delta^-_S(j) \geq 0$ for all $i \in V \setminus I$, $j \in I$
- $\Theta_S[e] \geq 0$, $e \in E \setminus E$

5 The case with non-demand vertices

Now we study $(T, f, c, p, D, R)$ when not all vertices must be necessarily visited, i.e. $D \cup V(R) \subset V$. We address the case without setup costs and with only vertex demand $(T, 0, c, p, D, \emptyset)$. As in previous sections the case with demand at edges can be reduced to this case by compacting each demand component into one single vertex.

In the case of general graphs, non-demand vertices can be eliminated by defining edges associated with shortest paths if the intermediate nodes of such paths use some non-demand vertex. This cannot be done however in the case of a tree without breaking the tree-structure of the original graph, except for non-demand vertices with degree two.

In the case of trees, allowing for vertices that do not necessarily have to be covered in feasible solutions increases notably the difficulty of the associated location routing problem. The main reason is that a priori one does not know the vertices in $V \setminus D$ that will be covered in an optimal solution. In other words, one does not know in advance, what edges incident with non-demand vertices should be considered as potential candidates for an optimal solution. This is illustrated by the example of Figure 4 where for $p = 4$ Algorithm 1 produces the non-optimal solution of value 19 Figure 4(b). The optimal solution to this instance is depicted in Figure 4(c); it has a value of 18 and does not cover vertex 5. Observe, however, that if in this example we modify slightly the edge costs to $c_{45} = c_{56} = 5$ and $c_{57} = 6$, all other costs remaining unchanged, then the outcome of Algorithm 1, which is the solution depicted in Figure 4 (b), is optimal. The value of this solution for the instance with modified costs is 16, and it covers the non-demand vertex 5. Note also that all distances between pairs of demand vertices in the $S$-component $\{4, 5, 6, 7\}$, are greater than 9, which is the length of two tree edges connecting pairs of demand vertices, that do not make part of this optimal solution. This may seem counterintuitive at first sight.

Below we present the recursion of an algorithm that optimally solves $(T, 0, c, p, D, \emptyset)$. We arbitrarily root the tree $T$ at some demand vertex $i_0 \in D$. For each $i \in V$, we say that $j$ is a successor of $i$ if $i$ is on the unique path connecting $j$ to $i_0$. If $j$ is a successor of $i$ then we say that $i$ is a predecessor of $j$. A direct descendant (children) of $i$ is a descendant connected to $i$ via an edge of $T$. Let $C(j) = \{i_1(j), i_2(j), \ldots, i_{n_j}(j)\}$ denote the index set of the children of $j$. When clear from the context we just write $C(j) = \{i_1, i_2, \ldots, i_{n_j}\}$. Without loss of generality we assume that the elements
of \(C(j)\) are ordered by nondecreasing distance to \(j\), i.e. \(c_{j,i_1} \leq c_{j,i_2} \cdots \leq c_{j,i_{n_j}}\). If \(C(j)\) is empty, \(j\) is called a leaf of \(T\). (Note that all leaves have degree one in the original tree \(T\), and the only possible non-leaf with degree one could be the root.)

Throughout the section we make the following assumptions:

(i) All the leaves of \(T\) are demand vertices. Otherwise, they can be eliminated from the tree, as they will not be covered in any optimal solution.

(ii) All non-demand vertices have degree at least three. (All non-demand vertices with degree two have been eliminated).

(iii) No optimal solution contains an \(S\)-component with no demand vertex. (Such an \(S\)-component could be eliminated)

Hence, since there are no set-up costs, similarly to Section 3.1, we focus on solutions where each \(S\)-component contains exactly one open facility, thus associating the number of open facilities with the number of \(S\)-components.

To solve \((T, 0, c, p, D, \emptyset)\), we recursively solve a sequence of problems defined on certain subtrees of \(T\), starting from the leaves. To define these subtrees, consider a vertex \(j \in V\). For any \(k = 1, \ldots, n_j\), let \(T_{j,k}\) denote the subtree induced by the vertices in \(\{j\} \cup C(j_1) \cup \cdots \cup C(j_k)\). We also denote by \(D_{j,k}\) the set of demand vertices in \(T_{j,k}\), i.e. \(D_{j,k} = D \cap V(T_{j,k})\), and by \(q_{j,k} = \min\{|D_{j,k}|, p\}\). Note that \(q_{j,k}\) is an upper bound on the number of facilities that can be located in the subtree \(T_{j,k}\) in any feasible solution to \((T, 0, c, p, D, \emptyset)\). Figure 5 depicts an example of these definitions.

In the recursion, for all \(j \in V\), \(z(j, k, q)\) denotes the optimal value to \((T_{j,k}, 0, c, q, D_{j,k}, \emptyset)\), where \(k \leq n_j\) and \(q \leq q_{j,k}\). We initialize \(z(j, k, 0) = \infty\) for all \(j \in V\), \(k \leq n_j\), indicating that in any feasible solution any subtree containing no facility must be connected to some vertex outside the subtree. When \(j\) is a leaf, \(z(j, 0, 1)\) is initialized to 0, indicating the cost of an \(S\)-component containing one facility located at the singleton \(\{j\}\). For all other vertices the values of \(k\) are restricted to be greater than or equal to 1. When \(j\) is not a leaf, the values \(z(j, k, q)\) are only computed after the values \(z(i, r, s)\) have been computed for all its children \(i \in C(j)\) and possible values of \(r\) and \(s\). For ease of presentation and without loss of generality we assume that vertices indices are ordered in such a way that \(i_k(j) > j\) for all \(k = 1 \ldots n_j\). Then, selecting the indices \(j\) by decreasing values, ensures that the recursions \(z(j, k, q)\) are computed in a correct order. The optimal value to \((T, 0, c, p, D, \emptyset)\) will be given by \(z(i_{0}, n_{i0}, p)\).

Only for clarity purposes, we first present the case where all vertices have demand, \(D = V\). In this case, for any subtree \(T_{i,k}\), \(D_{i,k} = V(T_{j,k})\). The general case where \(D \subseteq V\) will follow.

**Proposition 4** Suppose \(D = V\). Let \(j \in V\) be a non-leaf vertex and suppose the values \(z(i, r, s)\) have been computed for all \(i \in V\), \(i > j\) and possible values of \(r\) and \(s\). Then,
Initialization:

\( z(j, 0, 1) = 0 \) for all \( j \in V \) leaf
\( z(j, k, 0) = \infty \) for all \( j \in V \), \( 1 \leq k \leq n_j \)

Theorem 4 gives rise to the following algorithm, which solves \((T, 0, c, q, V, \emptyset)\) for all \( q \in \{1, \ldots, |V|\} \).

Algorithm 4

(a) \( z(j, 1, q) = \min\{z(i_1, n_{i_1}, q - 1), z(i_1, n_{i_1}, q) + c_{j,i_1}\} \), for all \( 1 \leq q \leq q_j \)

(b) For all \( k \in \{2, \ldots, n_j\} \), \( 1 \leq q \leq q_j \)

\[
\begin{align*}
\min_{q_1 + q_2 = q} & \quad \left[ z(j, k - 1, q_1) + z(i_k, n_{i_k}, q_2) \right] \\
\text{s.t.} & \quad 1 \leq q_1 \leq q_{j,k-1} \leq q_j, \quad 1 \leq q_2 \leq q_{j,k} = q_j \end{align*}
\]

(1)

\[
\begin{align*}
\min_{q_1 + q_2 = q + 1} & \quad \left[ z(j, k - 1, q_1) + z(i_k, n_{i_k}, q_2) \right] \\
\text{s.t.} & \quad 1 \leq q_1 \leq q_{j,k-1} \leq q_j, \quad 1 \leq q_2 \leq q_{j,k} = q_j \end{align*}
\]

(2)

Proof:

(a) In any optimal solution to \((T_{j,1}, 0, c, q, V(T_{j,1}), \emptyset)\), either vertex \( j \) defines one \( S \)-component on its own (and \( T_{i_1,n_{i_1}} \) contains \( q - 1 \) facilities), or \( T_{i_1,n_{i_1}} \) contains \( q \) facilities and vertex \( j \) is connected to some \( S \)-component of \((T_{i_1,n_{i_1}}, 0, c, q, V(T_{i_1,n_{i_1}}), \emptyset)\) via edge \((j, i_1)\). In the former case, the optimal value is given by \( z(i_1, n_{i_1}, q - 1) \). In the second case, the optimal value is given by \( z(i_1, n_{i_1}, q) + c_{j,i_1} \).

(b) When \( k > 1 \), in any optimal solution to \((T_{j,k}, 0, c, q, V(T_{j,k}), \emptyset)\), either the solution contains edge \((j, i_k)\) or it does not. In the first case, since \( c_{j,i_1} \leq c_{j,i_2} \leq \cdots \leq c_{j,i_{k-1}} \leq c_{j,i_k} \), vertex \( j \) must also be connected to some other vertex in \( T_{j,k-1} \). Otherwise the solution could be improved (or would not change, if ties exist) by interchanging edge \((j, i_k)\) with edge \((j, i_1)\). Thus, if the first case holds, the optimal value of \((T_{j,k}, 0, c, q, V(T_{j,k}, k), \emptyset)\) can be obtained by computing the best value \( z(j, k - 1, q_1) + z(i_k, n_{i_k}, q_2) \) among all possible combinations of \( q_1 \) facilities in \( T_{j,k-1} \), \( 1 \leq q_1 \leq q_{j,k-1} \) and \( q_2 \) facilities in \( T_{j_k,n_{i_k}} \), \( 1 \leq q_2 \leq q_{j,k} \) with \( q_1 + q_2 = q \). In the second case, edge \((j, i_k)\) would connect a partial solution with \( q_1 \) facilities (components) in \( T_{j,k-1} \) and a partial solution with \( q_2 \) facilities (components) in \( T_{i_k,n_{i_k}} \). That is, the solution containing edge \((j, i_k)\) and the two partial solutions will have \( q_1 + q_2 - 1 \) components. Hence, when the second case holds, the optimal value of \((T_{j,k}, 0, c, q, V(T_{j,k}), \emptyset)\) can be obtained by computing the best value \( c_{j,i_k} + z(j, k - 1, q_1) + z(i_k, n_{i_k}, q_2) \) among all possible combinations of \( q_1 \) facilities in \( T_{j,k-1} \), \( 1 \leq q_1 \leq q_{j,k-1} \) and \( q_2 \) facilities in \( T_{i_k,n_{i_k}} \), \( 1 \leq q_2 \leq q_{i_k} \) with \( q_1 + q_2 - 1 = q \).

Proposition 4 gives rise to the following algorithm, which solves \((T, 0, c, q, V, \emptyset)\) for all \( q \in \{1, \ldots, |V|\} \).
for \((j \in V \text{ not leaf})\) do

for \((k \in \{1, \ldots, n_j\}, 1 \leq q \leq q_{jk})\) do

if\((k = 1)\)

\[
z(j, 1, q) = \min\{z(i_1, n_{i_1}, q - 1), z(i_1, n_{i_1}, q) + c_{j,i_1}\}
\]

else

\[
z(j, k, q) = \min \left\{ \begin{array}{l}
\min_{q_1 + q_2 = q} \left[ z(j, k - 1, q_1) + z(i_k, n_{i_k}, q_2) \right] \\
\min_{q_1 \leq q_{j,k-1}} c_{j,i_k} + \min_{q_2 = q_{j,k-1}} \left[ z(j, k - 1, q_1) + z(i_k, n_{i_k}, q_2) \right] \\
\min_{q_2 \leq q_{i_k,n_{i_k}}} \left[ z(j, k - 1, q_1) + z(i_k, n_{i_k}, q_2) \right]
\end{array} \right. 
\]

Since \(\sum_{j \in V} n_j = n - 1\), the initialization step of Algorithm 4 as well as the number of possible pairs \((j, k)\) in the main loop are \(O(n)\). For a given pair \((j, k)\) at most \(p\) possible values of \(q\) will be considered. Computing \(z(j, k, q)\) with \(j, k, q\) fixed is \(O(q) \leq O(p)\), as there are at most \(q - 1\) possible combinations \(q_1 + q_2 = q\), with \(q_1, q_2 \geq 1\). Therefore:

**Proposition 5** The complexity of Algorithm 4 is \(O(np^2)\).

Because in the rooted tree used in Algorithm 4 the children of each node \(j\) are ordered by non-decreasing costs, \(c_{j,i_1} \leq c_{j,i_2} \cdots \leq c_{j,i_{n_j}}\), we can assume that if vertex \(j\) is connected to some \(S\)-component in one of its subtrees, this would be a component in its first subtree \(T_{j,1}\). Introducing facilities set-up costs would make this assumption false. Finding the best subtree for connecting any explored node \(j\) in this case would increase the complexity of the algorithm in one order of magnitude.

**Example 1** We illustrate Algorithm 4 on the graph of Figure 4 for \(p=4\), assuming that all vertices (including vertex 5) have demand, i.e., \(D = \{1, 2, 3, 4, 5\}\). Figure 6a depicts the rooted tree. Values next to each edge are their costs.

In the initialization we set \(z(2,0,1) = z(3,0,1) = z(6,0,1) = z(7,0,1) = 0\) and \(z(1,1,0) = z(1,2,0) = z(1,3,0) = z(4,1,0) = z(5,1,0) = z(5,2,0) = \infty\). Figure 6b gives the \(z(j,k,q)\) values computed via the recursion. The optimal value is \(z(1,3,4) = 19\).

Algorithm 4 produces solutions in which all vertices belong to some \(S\)-component. Thus it is not valid for the case when \(D \subset V\) and there is an optimal solution where some non-demand vertex does

\[
\begin{array}{|c|c|c|}
\hline
j & k & z(j, k, q) \\
\hline
1 & 2 & c_{12} + z(2,0,1) = 9 \\
& 3 & c_{13} + z(1,1,1) + z(3,0,1) = 12 \\
& 4 & z(1,2,2) = 6 \\
1 & 4 & z(5,2,3) = 0 \\
1 & 1 & c_{12} + z(2,0,1) = 9 \\
& 2 & c_{13} + z(1,1,1) + z(3,0,1) = 18 \\
& 3 & z(1,2,2) + z(4,1,1) = 37 \\
1 & 3 & z(1,2,2) + z(4,1,1) = 37 \\
& 4 & z(1,2,2) + z(4,1,1) = 19 \\
\hline
\end{array}
\]

(a) Rooted tree

(b) Values of \(z(j,k,q)\)

Figure 6: Illustration of Algorithm 4 for Example 1
not belong to any \( S \)-component. Below we present the extension to the general case \((T,0,c,p,D,\emptyset)\) with \(D \subseteq V\). Using the same notation as before, \(D_{i,k} = D \cap V(T_{i,k})\) denotes again the set of demand vertices in \(T_{i,k}\), although it is now possible that \(V(T_{i,k}) \setminus D_{i,k} \neq \emptyset\). Again, for all \(j \in V\), \(z(j,k,q)\) denotes the optimal value of \((T_{j,k},0,c,q,D_{j,k},\emptyset)\), with \(k \leq n_j\), and \(1 \leq q \leq q_{jk}\). As before, the optimal value to \((T,0,c,p,D,\emptyset)\) will be given by \(z(i_0,n_{i_0},p)\).

Now in the solution producing \(z(j,k,q)\), vertex \(j\) will not be covered unless \(j \in D\), or \(j \notin D\) but it is used to connect \(S\)-components in two of its subtrees, \(T_{i(r),n_{i(r)}}\) and \(T_{i(s),n_{i(s)}}\), with \(r, s \leq k\). The same may happen to some of its non-demand descendants. Nevertheless, even if vertex \(j\) is not covered in the solution producing \(z(j,k,q)\), it is possible that vertex \(j\) is covered in the solution to some subtree of some predecessor of \(j\). Hence, at a later step the recursion may consider solutions that connect \(j\) (or some of its uncovered descendants) with some predecessor of \(j\). For computing correctly the value of such solutions we need additional information. In particular, we use an auxiliary function \(g(j,k,q)\), which records the distance between \(j\) and the vertex covered in the solution producing \(z(j,k,q)\), which is closest to \(j\). Indeed, if \(j \in D\), then \(g(j,k,q) = 0\). Moreover, \(g(j,k,q) = 0\) even if \(j \notin D\) but it is covered by the partial solution of value \(z(j,k,q)\). Therefore, \(g(j,k,q) > 0\) only when \(j\) is not covered by the solution associated with \(z(j,k,q)\). In this case, the value of \(g(j,k,q)\) is given by the shortest distance between \(j\) and any of the vertices covered by the partial solution associated with \(z(j,k,q)\). Because, such distance may correspond to a path of \(G\), rather than to an edge, it has to be updated recursively.

To illustrate the above comments, consider again the rooted tree of Figure 6a, but assume now that \(D = \{1,2,3,4,5,6,7\}\) and vertex 5 does not have demand. Now \(z(5,1,1) = 0\) corresponds to the \(S\)-component \(\{6\}\), in which vertex \(j = 5\) is not covered, and \(g(5,1,1) = 6\) is the distance from vertex 5 to the \(S\)-component, which is given by \(c_{5,6} = 6\). The solution which gives \(z(4,1,2)\) consists of the \(S\)-components \(\{4, 5, 6\}\) and \(\{7\}\). The value \(z(4,1,2)\) is now given by \((c_{45} + z(5,1,1) + g(5,1,1)) + z(7,0,1) = 6 + 0 + 0 = 12\).

Taking into account the definition of the function \(g(j,k,q)\), the analog to Proposition 4 is now:

**Proposition 6** Suppose \(D \subseteq V\). Let \(j \in D\) be a non-leaf vertex and suppose the values \(z(i,r,s)\) and \(g(i,r,s)\) have been computed for all \(i \in V\), \(i > j\) and possible values of \(r\) and \(s\). Then, \(g(j,k,q) = 0\), for all \(k \in \{1,\ldots,n_j\}\), \(1 \leq q \leq q_{jk}\). Furthermore,

(i) \(z(j,1,q) = \min\{z(i_1,n_{i_1},q-1), z(i_1,n_{i_1},q) + c_{j,i_1} + g(i_1,n_{i_1},q)\}\) for all \(1 \leq q \leq |D_{jk}|\)

(ii) For all \(k \in \{2,\ldots,n_j\}\), \(1 \leq q \leq q_{jk}\)

\[
z(j,k,q) = \min \left\{ \begin{array}{ll}
\min_{1 \leq q_1 \leq q_{i_k-1}, 1 \leq q_2 \leq q_{i_k}} \{z(j,k-1,q_1) + z(i_k,n_{i_k},q_2)\} & (3) \\
\min_{1 \leq q_1 \leq q_{i_k}, 1 \leq q_2 \leq q_{i_k}} \{c_{j,i_k} + g(i_k,n_{i_k},q_2) + z(j,k-1,q_1) + z(i_k,n_{i_k},q_2)\} & (4)
\end{array} \right.
\]

The recursion for non-demand vertices is slightly more complicated, but follows the rationale explained above. It is summarized below:

**Proposition 7** Suppose \(D \subseteq V\). Let \(j \notin D\) be a non-leaf non-demand vertex and suppose the values \(z(i,r,s)\) and \(g(i,r,s)\) have been computed for all \(i \in V\), \(i > j\) and possible values of \(r\) and \(s\). Then,

(i) \(z(j,1,q) = z(i_1,n_{i_1},q)\), for all \(1 \leq q \leq q_{jk}\)

(ii) \(g(j,1,q) = g(i_1,n_{i_1},q) + c_{j,i_1}\), for all \(1 \leq q \leq q_{jk}\)
(ii) For all \( k \in \{2, \ldots, n_j\} \), \( 1 \leq q \leq q_{jk} \)

\[
\begin{align*}
z(j, k, q) & = \min \left\{ \begin{array}{ll}
\min & \{z(j, k - 1, q_1) + z(i_k, n_{ik}, q_2)\} \\
\text{s.t.} & 2 \leq q_1 \leq q_{ik}, k - 1 \\
& 2 \leq q_2 \leq n_{ik}, q_{jk} \end{array} \right. \\
& \begin{array}{ll}
\min & \{g(j, k - 1, q_1) + g(i_k, n_{ik}, q_2) + z(j, k - 1, q_1) + z(i_k, n_{ik}, q_2)\} \\
\text{s.t.} & 2 \leq q_1 \leq q_{ik}, k - 1 \\
& 2 \leq q_2 \leq n_{ik}, q_{jk} \end{array}
\end{align*}
\] (5)

(iii) For \( k \in \{1, \ldots, n_j\} \), \( 1 \leq q \leq q_{jk} \)

\[
\begin{align*}
g(j, k, q) & = \begin{cases}
\min\{g(j, k - 1, q_1), c_{j, ik} + g(i_k, n_{ik}, q_2)\} & \text{if value of } z(j, k, q) \text{ given by (5)} \\
0 & \text{if value of } z(j, k, q) \text{ given by (6)}.
\end{cases}
\end{align*}
\] (6)

Propositions 6 and 7 give rise to an algorithm with the same structure as Algorithm 4 which uses the expressions above for computing all possible values of \( z(j, k, q) \) and \( g(j, k, q) \), depending on whether or not \( j \in D \). Broadly speaking it performs in the same number of iterations, so its complexity is again \( O(np^2) \).

6 A mathematical programming formulation with integrality property for \((T, f, c, p, D, R)\)

Below we present a mathematical programming formulation for \((T, f, c, p, D, R)\) that, as we will see, has the integrality property. The formulation builds a directed forest where \( p \) facilities are located and each connected component hosts at least one open facility. Let \( A \) denote the set containing two arcs associated with each edge of \( E \), one in each direction. That is, \( A = \{(u, v), (v, u) \mid (u, v) \in E\} \). The dicuts associated with a vertex subset \( S \subset V \), are denoted as \( \delta^+(S) = \{(u, v) \in A \mid u \in S, v \in V \setminus S\} \) and \( \delta^-(S) = \{(u, v) \in A \mid u \in V \setminus S, v \in S\} \).

The formulation uses two sets of decision variables, one to represent the vertices where facilities are located and another one to represent the arcs of the forest. For \( i \in D \) let \( y_i \) a binary variables, which takes the value one if and only if a facility is located at vertex \( i \). In addition, each arc \((u, v) \in A\) is associated with a binary decision variable \( x_{uv} \), which takes the value one if and only if arc \((u, v)\) belong to the directed forest. The formulation is as follows:

\[
LR^T: \quad \min \sum_{i \in D} f_i y_i + 2 \sum_{(u, v) \in A} c_{uv} x_{uv} \quad \text{subject to} \quad \sum_{i \in D} y_i = p \quad (7a)
\]

\[
\sum_{i \in S} y_i + \sum_{(u, v) \in \delta^+(S)} x_{uv} \geq 1 \quad (u, v) \in A \quad (7c)
\]

Constraints (7b) impose that exactly \( p \) facilities are selected, whereas constraints (7c) guarantee that the connected components induced by demand vertices contain at least one open facility. The objective function computes the setup costs of the open facilities plus the routing costs of the used arcs.
7 Conclusions

In this paper we have studied combined location/routing problems (LRPs) defined on simple graphs, namely trees. Several problems have been studied, which consider demand both at the vertices and the edges of the input tree. Greedy type optimal heuristics have been presented for the cases when all vertices have to be visited and facilities have no set-up costs. Facilities set-up costs can be handled with interchange heuristics of low order. Nevertheless, these methods solve no longer problems where some of the vertices do not have to be necessarily visited. For these more general LRPs we have presented low order optimal algorithms based on recursions.

All presented algorithms can be extended with slight modification to cacti, graphs in which each edge may belong to at most one cycle. It is possible to deal with each such cycle by considering two possible cases for an optimal solution: either all the edges of the cycle are traversed once, or at least one edge of the cycle is not traversed. The former case can be handled by compacting the cycle into one single vertex. In the later case the cost of largest cost in the cycle can be removed. (See [3] for further details in the case of prize collecting problems on cacti.) Indeed, such extensions imply an increase of one order of magnitude in the complexity of the presented algorithms.

All considered problems consider no capacity constraints. A promising avenue for future research is the study of LRPs with the simplest type of capacity constraints, i.e. cardinality constraints on the number of vertices in each route. From a different perspective, it can be interesting to study the usefulness of the proposed algorithms in heuristic methods for LRPs in general graphs.

8 Acknowledgements

This research has been partially supported by the Spanish Ministry of Economy and Competitiveness through project MTM2012-36163-C06-05, and ERDF funds. This support is gratefully acknowledged.

References


