Facility Location with Hierarchical Facility Costs

Zoya Svitkina[*] Éva Tardos[†]

Abstract

We consider the facility location problem with hierarchical facility costs, and give a $(4.236 + \epsilon)$-approximation algorithm using local search. The hierarchical facility location problem models multilevel service installation costs. Shmoys, Swamy and Levi [13] gave an approximation algorithm for a two-level version of the problem. Here we consider a multilevel problem, and give a constant factor approximation algorithm, independent of the number of levels, for the case of identical costs on all facilities.

1 Introduction

In the basic facility location problem, we are given a set of clients and a set of possible facilities. A solution consists of opening some facilities and assigning each client to an open facility, with the objective of minimizing the sum of the facility opening cost and the client connection cost. In the most basic and well-studied version of the problem, the metric uncapacitated facility location, there is a metric of distances between the clients and the facilities, and the connection cost is the sum of distances between clients and the facilities to which they are assigned. Each facility has a cost given as part of the input, and the total facility opening cost is the sum of costs for the facilities that are opened.

Facility location problems have been used to model a wide range of practical settings, including location problems, supply chain management, Web server locations, etc. While even the basic uncapacitated metric facility location problem is NP-complete, there are many good approximation algorithms known for it, using the whole range of techniques including local search, linear programming and the primal-dual method.

The most studied extensions of the basic facility location problem involve facility costs that are more complex than the constant opening costs of the uncapacitated problem. Versions of the capacitated facility location problems aim to model the limited capacity of facilities to serve clients. In the most general of these models, the universal facility location problem [11], the facility cost is an arbitrary (non-decreasing) function of the number of clients assigned to the facility.

In this paper, we study a variant of the facility location problem where facility cost depends on the *set* of clients assigned to the facility, and not just their number. We propose a general model in which the facility costs are submodular functions of the set of clients assigned to the facility, where submodularity models a natural economy of scale. We focus on a subclass of submodular cost functions which we call hierarchical costs. Shmoys, Swamy and Levi [13] introduced the problem of facility location with service installation costs. In their model, each client requests a certain service, which has to be installed at the facility where this client is assigned. There are costs for opening facilities and for installing services. The hierarchical facility location problem is an extension of this model to many levels of service costs. Instead of allowing only two levels (facility opening and service installation), we allow an arbitrarily deep hierarchy that describes the costs. Such a hierarchy can be used to model a richer set of cost structures. Our main result is a local search algorithm that gives a $(4.236 + \epsilon)$-approximation (independent of the number of levels in the hierarchy) in the case that the costs are identical at all facilities.

At the end of the paper we consider the general model in which the facility costs are submodular functions of the set of clients assigned to the facility, not necessarily defined based on a tree-structure. We conjecture that the problem admits a constant factor local search approximation algorithm if the facility costs are monotone submodular functions. This generalization would allow us to model an even wider range of applications.

Our model and methods. More formally, we assume that we are given a set of facilities $\mathcal{F}$, a set of demands or clients $\mathcal{D}$, and a distance metric *dist* on the set $\mathcal{F} \cup \mathcal{D}$, as is also the case for the standard facility location problem. In addition, we are given a rooted

[*]Cornell University, Department of Computer Science, Upson Hall, Ithaca, NY 14853. Email: zoya@cs.cornell.edu. Supported in part by ONR grant N00014-98-1-0589.

[†]Cornell University, Department of Computer Science, Upson Hall, Ithaca, NY 14853. Email: eva@cs.cornell.edu. Supported in part by NSF grant CCR-0311333, ITR grant CCR-0325453, and ONR grant N00014-98-1-0589.

cost tree T (used to define facility costs) whose leaves correspond to the different possible services. Each client requests a service, i.e., is associated with a leaf in the tree. Each node k of T has a non-negative cost, $cost(k)$. The goal is to assign each client to a facility, minimizing the sum of the *connection cost* C and the *facility cost* F. The connection cost of a client j that is assigned to facility i is $C(j) = dist(j, i)$. The facility cost for facility i, $F(i)$, is a function of the set of clients assigned to i. We assume that the function is the same for all facilities, and is defined using the tree T as follows. For a set S of clients assigned to facility i, we use the notation T_S to denote the subgraph of T induced by the nodes that lie on a path from the root to some leaf corresponding to a service requested by a client in S (see Figure 1). Then

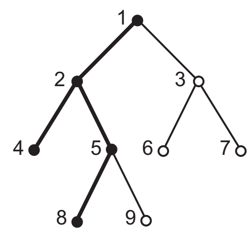

Figure 1: Example of a cost tree. If clients corresponding to leaves 4 and 8 form the set S which is assigned to a facility i, then the cost of i would be $F(i) = cost(1) + cost(2) + cost(4) + cost(5) + cost(8)$. Shaded nodes and thick edges form the subgraph T_S.

the cost of i is $F(i) = cost(S) = \sum_{k \in T_S} cost(k)$. The cost of an empty facility is zero. The overall objective then is to minimize $C + F = \sum_{j \in \mathcal{D}} C(j) + \sum_{i \in \mathcal{F}} F(i)$.

Our main result is a local search method, where the local optima are 5-approximate solutions to the problem, in the case that the costs are identical at all facilities. Then we improve this bound to 4.236 by scaling. Note that the problem with independent cost functions on different facilities is SET COVER-hard, as was shown by [13].

We develop the local search method in the next section. The algorithm starts with an arbitrary assignment of clients to facilities, and uses two types of local improvement moves: *aggregate* and *disperse*. The aggregate move is analogous to the move *open*, which opens a new facility, used in local search algorithms for the traditional facility location problem (e.g. [2, 1]). In that case, however, once a facility is open, it is easy to decide which clients should be assigned to it. In the case of our hierarchical service cost model, this is less easy to decide. In fact, there is no clear meaning of opening a facility, as the cost may depend dramatically on which clients are assigned to it. We describe the move in the next section, where we also prove that if there are no improving aggregate moves, then the connection cost of the current solution can be bounded.

To bound the facility costs, we use a different operation, *disperse*, which is the analog of the traditional *close* operation. The high-level idea is to disperse the clients of one currently open facility to other facilities, and hence to save on facility cost. Implementing such a disperse move optimally can be used to solve the original problem in a single step. In Section 3 we describe an approximate disperse move that can be implemented in polynomial time. We then use the aggregate and disperse moves to show that a locally optimal solution is a 5-approximation for the problem.

Using scaling we improve the bound to 4.236, and accepting only sufficiently large improvements, we can turn this into a polynomial time $(4.236 + \epsilon)$-approximation algorithm for the hierarchical facility location problem.

Motivation. Our hierarchical facility location problem can be used to model practical scenarios, such as the costs of multiple levels of service installation [5]. For example, one may consider opening some stores and wonder about which items to carry in each of them. The cost of opening a store would be represented by the root of the tree, the cost of carrying a particular category of items (e.g., those supplied by a particular vendor) would be at a node one level down, that for a subcategory still lower, and so on. A client may then be identified with the item he wants to buy.

Another possible application is for data storage. Then facilities are the file servers, and clients are the users (or contributors) of data. The connection cost is the price of transferring data over the network, and facility cost is the price for storing the data. The hierarchy may represent how similar the data items are, with the assumption that similar files can be compressed together to save storage cost.

The generalization of our problem in which the facility costs are allowed to differ from each other by constant factors can model data gathering in sensor networks (see, for example, [16]). In that case, instead of representing storage expenses, the facility costs would model the cost of delivering compressed data to the destination.

Related work. There has been much work on approximation algorithms for the uncapacitated facility location problem. Small constant factor approximation algorithms exist that use essentially every known approximation algorithmic technique, including local search [10, 2, 1], primal-dual method [9], and linear programming and rounding [14, 3, 4, 15]. The current best approximation guarantee is 1.52 [12], and no better than

1.463 approximation is possible unless NP $\subseteq \tilde{\mathrm{P}}$ [7].

The most well-studied extension of the facility location problem considers facilities with capacities, where the capacity of a facility bounds the number of clients it can serve if opened. Many of the uncapacitated approximation algorithms extend to the version of the problem with soft capacities [9, 2, 1, 4], where soft capacities allow us to open a facility multiple times, in order to support more clients. Similarly many of the approximation algorithms can be used to derive bicriteria approximations for the capacitated case, by allowing the algorithm to violate the capacities to some extent. Local search is the only known algorithmic technique that extends to handle hard capacities, i.e., it gives an approximation algorithm that neither violates the capacities, nor opens the facilities multiple times. The most general such problem formulation is the universal facility location problem of Mahdian and Pál [11], where the cost of a facility is an arbitrary monotone function of the number of clients assigned to it.

In contrast, in the hierarchical facility location problem we consider, the facility cost depends on the *set of clients* assigned to the facility and not just their number. In the facility location with service installation costs, introduced by Shmoys, Swamy and Levi [13], the costs can be represented by a tree of height two. Note that the Shmoys et al. model is somewhat more general, as the costs need not be the same at all facilities. However, they also require that the facility costs are related in that they assume that the facilities have an ordering in which each earlier facility costs less than a later one for all services. For example, this rule allows proportional facility costs. The algorithm of [13] can be extended to handle hierarchical costs, but with approximation ratio that depends linearly on the height of the tree. In contrast, our algorithm yields a bound of 5, independent of the number of levels in the tree. The group facility location problem of Hayrapetyan, Swamy and Tardos [8] contains a similar two-level cost structure, but for connection costs. For facility costs they use the uncapacitated model.

Notation. Finally, we define some further notation that will be useful in the rest of the paper. The analysis of the algorithm compares a locally optimal solution to a fixed globally optimal one. We call the former LOC and the latter OPT. Let C and F denote the connection and facility costs of LOC, respectively, and C^* and F^* denote the same quantities for OPT. For a client j, let $f(j)$ be the facility where j is assigned in LOC and $f^*(j)$ be the facility where j is assigned in OPT. For a facility i, $\mathcal{D}(i)$ denotes the set of clients assigned to i in LOC, and $\mathcal{D}^*(i)$ is the set assigned in OPT. If a set of clients S is moved to a facility i that already contains some clients $\mathcal{D}(i)$, then $cost_i(S)$ is the additional facility cost that has to be paid, and is equal to $cost(S \cup \mathcal{D}(i)) - cost(\mathcal{D}(i))$.

2 Aggregate move and the connection cost

In this section we only consider one of the two local moves, *aggregate*. When performed on a facility i, it transfers some clients from other facilities to i. This change in assignment of clients affects the cost in three ways: the clients that change assignments have a changed connection cost; facility i has an increased facility cost; and other facilities may have decreased facility costs as some clients move away from them. While it is possible to find the optimal way to move clients to facility i, we instead use a simplified cost that is easier to compute, and is good enough for our purposes. Rather than exactly computing the cost of the new assignment, we ignore the savings in cost at the other facilities, and we find a set of clients $S \subseteq \mathcal{D}$ minimizing the change in connection cost plus the added facility cost incurred at facility i. This estimated change in cost can be expressed as

$$\sum_{j \in S} dist(j,i) - \sum_{j \in S} dist(j, f(j)) + cost_i(S). \tag{2.1}$$

We call (2.1) the *value* of the aggregate move (despite the fact that expression (2.1) did not take into account the possible cost-savings at other facilities), and we call the move *improving* if it has negative value. Note that if a move is performed, then the real cost of the solution decreases by at least as much as estimated using the move's value, which means that the solution is indeed improving when we make improving moves. In the next subsection we show how to find the optimal set S with respect to expression (2.1) for a given facility i. Then we show that if a solution has no improving aggregate moves, its connection cost can be bounded.

2.1 Finding the aggregate move with optimal value.

In this and later sections we assume without loss of generality that the cost tree T is binary: if it is not, then any node with high degree can be expanded into a binary subtree without increasing the overall size of the tree by much. Similarly, we assume without loss of generality that each client is associated with a separate leaf of the tree.

Consider a facility i on which the aggregate move is performed. Moving a particular client j from his current facility $f(j)$ to i has two effects: the connection cost decreases by $dist(j, f(j)) - dist(j,i)$, and the facility cost at i increases. The goal is to find a subset of clients for which the savings in connection cost outweigh the increased facility cost. For a client j we will think of the difference

$$b(j) = dist(j, f(j)) - dist(j, i),$$

as a budget that he has for paying the increased facility cost.

Now consider the added facility cost incurred at facility i, which is

$$cost_i(S) = cost(S \cup \mathcal{D}(i)) - cost(\mathcal{D}(i))$$

for a set S. Notice that this is the same as the facility cost obtained from a modified cost tree T' in which we set $cost'(k) = 0$ for all nodes $k \in T_{\mathcal{D}(i)}$ that are already paid for at i. Given this new notation, we seek a set S minimizing $cost_i(S) - b(S)$. If this expression is negative, then transferring clients in S to i is an improving move. The set optimizing this expression, and hence defining the best aggregate move, can be found by a simple dynamic programming on the tree T' analogous to the algorithm used by Feigenbaum, Papadimitriou and Shenker [6] for cost-sharing on a multicast tree.

LEMMA 2.1. *The subset S that minimizes $cost_i(S) - b(S)$, and hence defines the aggregate move of minimum value for a given facility i, can be found by dynamic programming on the tree T' in linear time $O(|T|)$.*

Proof. The optimal set S can be found using a bottom-up traversal of T'. For each node $k \in T'$, we calculate the maximum value $v(k)$ that can be obtained by including a subset of clients in the subtree rooted at k, and the corresponding subset $S(k)$. This value is the difference of the budgets of included clients and the facility costs that have to be paid for including them. If k is a leaf node and j is the client corresponding to it, then

$$v(k) = max(0,\ b(j) - cost(k)).$$

We set $S(k) = \{j\}$ if $v(k) > 0$ and $\emptyset$ otherwise. If k is an internal node with children k_1 and k_2, then

$$v(k) = max(0,\ v(k_1) + v(k_2) - cost(k)),$$

and $S(k) = S(k_1) \cup S(k_2)$ if $v(k) > 0$ and $\emptyset$ otherwise. It can be shown by induction on the depth of the tree that the resulting set $S = S(r)$, where r is the root of T', is optimal. The move of transferring $S(r)$ to facility i has value $-v(r)$. $\square$

2.2 Bounding the connection cost. Now consider a solution LOC with no improving aggregate moves. We bound the connection cost of this solution in a similar way as used in local search algorithms for other facility location problems.

LEMMA 2.2. *The connection cost C of the locally optimal solution LOC can be bounded by the optimal cost as $C \leq C^* + F^*$.*

Proof. If there are no improving aggregate moves, then expression (2.1) is nonnegative for moving *any* set of clients to *any* facility i. We consider expression (2.1) for the set of clients $\mathcal{D}^*(i)$ that are assigned to i in OPT. We have that

$$\sum_{j \in \mathcal{D}^*(i)} dist(j, i) - \sum_{j \in \mathcal{D}^*(i)} dist(j, f(j)) + cost_i(\mathcal{D}^*(i)) \geq 0.$$

Using the fact that $cost_i(\mathcal{D}^*(i)) \leq cost(\mathcal{D}^*(i))$ and adding the inequalities for all facilities i, we get

$$\sum_j dist(j, f^*(j)) - \sum_j dist(j, f(j)) + \sum_i cost(\mathcal{D}^*(i)) \geq 0,$$

or $C^* - C + F^* \geq 0$, which implies that $C \leq C^* + F^*$. $\square$

3 Disperse move and the facility cost

Next we consider the *disperse* move, which reassigns clients from one facility i to other facilities, decreasing the facility cost at i. We use this move to bound facility costs. The outline of this section is analogous to that of the previous one. First we define a certain class of disperse moves, then we show that the optimal move in this class can be found in polynomial time, and then we exhibit a particular set of moves that allows us to bound the facility cost of a locally optimal solution.

3.1 Definition of the disperse move. Consider a facility i. The idea of the disperse move is to move some of the clients assigned to facility i to other facilities. If we could find the optimal such move, then one disperse move would solve the whole problem: just start with all clients assigned to one facility, and do the optimal disperse move on that facility. As a result, we do not consider the most general version of a disperse move with the exact evaluation function, but instead restrict our attention to a subclass of moves and an approximate evaluation.

The key notion used to define the restricted set of moves we consider is that of a tree-partition. A *tree-partition* of a set of clients $S \subseteq \mathcal{D}$ is any partition that can be obtained by cutting a subset of edges of T_S and grouping the clients by the resulting connected components of their associated tree leaves (see Figure 2). For example, if no edges are cut, then all clients are in the same set; if all leaf edges are cut, then each

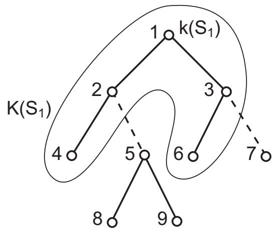

Figure 2: Example of a tree-partition. Cutting the two dashed edges produces three subsets: $S_1 = \{4, 6\}$, $S_2 = \{8, 9\}$, and $S_3 = \{7\}$. The marked component is $K(S_1)$, node 1 is $k(S_1)$, and node 5 is $k(S_2)$.

client is in a separate set. Let us refer to the connected component associated with a set S_h as $K(S_h)$. Let $k(S_h)$ be the unique highest node of the connected component $K(S_h)$, i.e. the one closest to the root of T. A *disperse move* for facility i consists of a tree-partition $\{S_h\}$ of the set of clients $\mathcal{D}(i)$ currently at i, and a facility $f'(S_h)$ for each set S_h in the partition, which is where this set should be reassigned. It is possible that $f'(S_h) = i$ for some sets.

Next we define the way we evaluate a disperse move. Consider a facility i, a tree-partition $\{S_h\}$ of its clients, and assume that we are assigning S_h to the facility $f'(S_h)$. To estimate the facility cost of moving a set S_h to $i' = f'(S_h)$, we use the incremental cost incurred by adding clients in S_h to the clients already at i', $cost_{i'}(S_h)$. We treat the facility i itself as if it were empty, i.e. for the purposes of expression (3.2), we define $cost_i(S_h) = cost(S_h)$. For different sets S_h and $S_{h'}$ we do not take into account the savings incurred if we are adding both sets to the same facility, so we simply sum the costs $\sum_h cost_{f'(S_h)}(S_h)$ for the partition. This upper bounds the resulting facility costs. For the change in connection costs, we also use an upper bound instead of the exact value. Here the triangle inequality on the distance metric is used: the increase in connection cost for client j that is moved from i to i' is upper-bounded by $dist(i, i')$.

We wish to find a tree-partition $\{S_h\}$ of the clients $\mathcal{D}(i)$ and an assignment of each set S_h to a facility $f'(S_h)$ so as to minimize the expression

$$(3.2)\quad \sum_{S_h} cost_{f'(S_h)}(S_h) + \sum_{S_h} |S_h| dist(i, f'(S_h)) - F(i),$$

where $F(i)$ is the cost of facility i which is saved when clients are removed from it. This expression, which we call the *value* of the disperse move, is an upper bound on the change in the solution cost when the disperse move is performed. Any move with negative value is an *improving* move.

3.2 Finding the disperse move with optimal value. In this section we show how the optimal disperse move with respect to expression (3.2) can be found in polynomial time. First we prove a lemma that is useful for deriving the algorithm.

LEMMA 3.1. *There exists a disperse move of minimum value in which if S_h is assigned to facility $f'(S_h)$, then none of the nodes of $K(S_h)$ are paid for at $f'(S_h)$.*

Proof. Given an optimal disperse move that does not satisfy the required condition, we transform it into one that does, without increasing the cost. If any node of $K(S_h)$ is paid for at facility $f'(S_h)$, then so is $k(S_h)$, since it lies on the path from that node to the root. By our assumption that each client is associated with a different leaf of T, the node $k(S_h)$ has to be an internal node (there are at least two clients in its subtree: one belonging to S_h, and one at facility $f'(S_h)$). In this case we transform the tree-partition by cutting additional edges of the tree, namely the ones connecting $k(S_h)$ to its children (see Figure 3). This

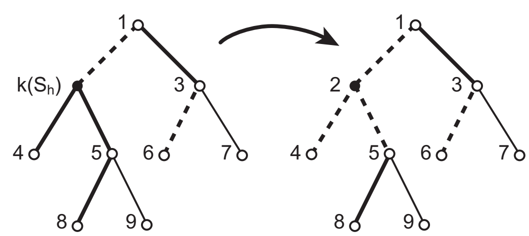

Figure 3: Operation performed in the proof of Lemma 3.1. Suppose that S_h consists of clients 4 and 8, and that node 2 is paid for at the facility $f'(S_h)$.

may split S_h into two sets, both of which we assign to the same facility $f'(S_h)$ as S_h was assigned. The new solution costs no more than the old one: the connection cost is clearly the same, and since the node $k(S_h)$ and ones above it are already paid for at $f'(S_h)$, the facility cost does not increase. This procedure can be repeated eliminating each set that does not satisfy the condition, in a top-down pass through the tree. $\square$

The algorithm for finding the optimal disperse move satisfying the conditions of Lemma 3.1 is based on dynamic programming.

LEMMA 3.2. *For a given facility i, the disperse move with minimum value can be found by dynamic programming on the tree $T_{\mathcal{D}(i)}$ in polynomial time.*

Proof. The algorithm makes a bottom-up pass through the tree. At each node k, it considers whether or not the tree edge above k should be cut, thus defining the tree-partition. If the edge above k is cut, producing a new component, then the corresponding set of clients S_h is assigned to a facility $f'(S_h)$ minimizing $cost_{f'(S_h)}(S_h) + |S_h| \cdot dist(i, f'(S_h))$ and satisfying the condition of Lemma 3.1. For a given tree-partition and a client j belonging to the set S_h in it, let $k(j)$ be the highest node in $K(S_h)$. Three parameters describing a solution for the subtree below node k will be sufficient for proceeding to higher levels and choosing the best solutions there. We call these parameters $N(k)$, $P(k)$, and $Y(k)$.

- We define $N(k)$ to be the number of clients j from k's subtree for which $k(j)$ is strictly higher than k. If the edge above k is cut, then $N(k) = 0$. This is the number of clients whose connection cost is yet to be determined at a node higher in the tree. For example, for node $k = 2$ in Figure 2, $N(2) = 1$, as only client 4 satisfies this property. Note that our way of estimating the increase in connection cost for each client reassigned from i to i' by $dist(i, i')$ makes sure that the value of the move only depends on the number $N(k)$ and not the actual set of $N(k)$ clients that are reassigned.

- $P(k)$ is the sum over connected components contained entirely in k's subtree of the connection and facility costs of moving the corresponding groups of clients. More formally, for each such S_h assigned to $f'(S_h)$, the term $|S_h| \cdot dist(i, f'(S_h)) + cost_{f'(S_h)}(S_h)$ is included in the sum. This is the cost that the solution has already paid for this subtree. In Figure 2, $P(2)$ would include the connection cost and the facility cost of sending clients 8 and 9 to their new facility.

- Now consider the $N(k)$ clients from k's subtree that will be assigned to a facility i' later on. By Lemma 3.1, all costs in these clients' component will have to be paid at i'. Let $Y(k)$ be the amount of this cost that comes from k's subtree. In other words, if U is a set of nodes which lie on a path between k and some client included in $N(k)$, then $Y(k)$ is the total cost of nodes in U. This is the facility cost that the solution has committed to paying. In Figure 2, $Y(2) = cost(4) + cost(2)$.

The idea of the dynamic programming algorithm is that the two values $N(k)$ and $P(k)+Y(k)$ fully describe the quality of the solution in the part of the tree rooted at k. For each node k and integer $n \in \{0, ..., |\mathcal{D}|\}$, the algorithm computes the least possible value $A_k(n)$ of the cost $P(k) + Y(k)$ that can be achieved by a solution with $N(k) = n$. The table is constructed in a bottom-up pass through $T_{\mathcal{D}(i)}$. For each node k, the algorithm considers the options of cutting or not cutting the edge above it, filling in the entries of A_k corresponding to the outcomes. If k is an internal node with children k_1 and k_2, then the algorithm also loops over all entries of A_{k_1} and A_{k_2}, combining them to construct A_k. The cost of the final solution is $A_r(0)$ for the root r of T. □

3.3 Bounding the facility cost: a specific set of disperse moves.

The way we bound facility cost of LOC is by focusing on one specific disperse move for each facility and noticing that these moves (like all moves in a locally optimal solution) have non-negative values. In this section we describe these disperse moves, which consist of a tree-partition of clients at each facility and a destination facility for each set of clients in the partition.

Our technique involves finding a mapping on the set of clients, as was also done by Arya et al. [1], but is different in other respects. The idea is to find for each client j a "partner" client $\pi(j)$, close to j in the cost tree, such that the two are at the same facility in OPT but at different facilities in LOC. Then j can be sent to facility $f(\pi(j))$, the current facility of $\pi(j)$. This is good in two ways: first, because $\pi(j)$ is close to j in T, the additional facility cost that we have to pay for j at $f(\pi(j))$ is not too big; second, the connection cost for reassigning j can be bounded using the connection costs of j and $\pi(j)$ in OPT and in LOC (as shown in Figure 4). However, because of our inexact estimate of the facility cost in expression (3.2), if we reassign each client separately, then we may be paying certain facility costs multiple times. To avoid this, the clients are grouped into sets (in particular, ones forming a tree-partition), and each set is moved as a unit to the current facility $f(\pi(j))$ of one of the members of that set.

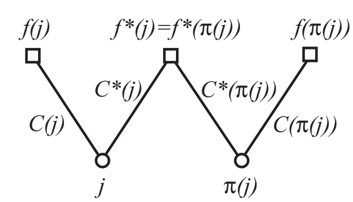

Figure 4: Clients j and $\pi(j)$ are assigned to the same facility in OPT, but to different facilities in LOC (unless $\pi(j) = j$). The marked distances are used in the proof of Lemma 3.4.

Defining the mapping. We start by defining the mapping π. We do this separately for sets $\mathcal{D}^*(l)$ of clients assigned to each facility l in OPT. The mapping $\pi : \mathcal{D}^*(l) \to \mathcal{D}^*(l)$ is 1-1. Note that the following hypothetical procedure is only used for the analysis, and need not (and cannot) be performed by the algorithm.

Consider a facility l used in the optimal solution. We will perform a bottom-up pass through the cost tree of its clients, $T_{\mathcal{D}^*(l)}$, mapping each client $j \in \mathcal{D}^*(l)$ to another client $\pi(j) \in \mathcal{D}^*(l)$. This procedure works even without our earlier simplifying assumptions of the tree being binary and of each client being at a separate leaf.

First consider the special case that $T_{\mathcal{D}^*(l)}$ consists of only one node, which contains the set of clients $S = \mathcal{D}^*(l)$. We partition the clients in S according to their facility in LOC, forming subsets $\mathcal{D}(i) \cap S$ for all facilities i (see Figure 5). We aim to assign $\pi(j) \in S$ for all clients $j \in S$ satisfying the condition that $f(j) \neq f(\pi(j))$, i.e. that j and $\pi(j)$ are assigned to different facilities in LOC. If there is a facility i that has the majority of clients, i.e., $|\mathcal{D}(i) \cap S| > |S|/2$, then we cannot assign all the clients at this level, so we assign as many of them as possible, and leave the others to be assigned later.

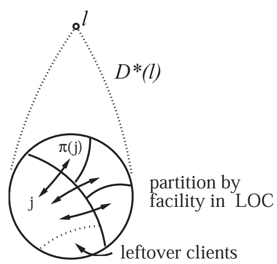

Figure 5: Example of how π is defined if the set of clients $\mathcal{D}^*(l)$ has only one level. If there are more levels, then the leftover clients of the majority facility propagate up the tree.

More generally, we consider such a partition at each level of the tree. Clients that are not assigned at one level propagate up the tree to be assigned at a higher node. All clients start out at their corresponding leaves. At each node k in $T_{\mathcal{D}^*(l)}$, bottom-up, perform the following actions. Consider the set S of clients at this node. These clients may in general belong to different facilities in LOC. Let i be a "majority" facility, one for which $|\mathcal{D}(i) \cap S|$ is the largest. In the simpler case, $|\mathcal{D}(i) \cap S| \leq |S|/2$. Then we can assign $\pi(j)$ for all clients $j \in S$ satisfying the condition that $f(j) \neq f(\pi(j))$, and none of the clients are propagated up the tree.

In the other case, if $|\mathcal{D}(i) \cap S| > |S|/2$, it is impossible to find a mapping π satisfying our condition among the clients at k. So we take only $|S - \mathcal{D}(i)|$ clients from $\mathcal{D}(i)$ and assign the mapping π between them and clients from $S - \mathcal{D}(i)$. The leftover clients from $\mathcal{D}(i)$ are propagated to the parent of k (we say that these clients *pass through* the edge between k and its parent). See Figure 6 for an example of this process.

If k is the root, then the leftover clients are assigned to themselves, that is we set $\pi(j) = j$. Note that we have $f^*(j) = f^*(\pi(j))$ for all j, and $f(j) \neq f(\pi(j))$ unless $j = \pi(j)$.

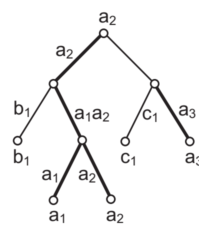

Figure 6: Example of the mapping. The leaves $\{b_1, a_1, a_2, c_1, a_3\}$ represent clients assigned to facility i in OPT. The different letters represent different facilities to which they are assigned in LOC. The labels on edges indicate which clients pass through them. The result is the mapping $a_1 \leftrightarrow b_1$ and $c_1 \leftrightarrow a_3$, with a_2 assigned to itself. The thick edges are included in $E(a)$.

Defining the tree-partition and the facility assignment. Consider a facility i used in LOC, and the set of clients $\mathcal{D}(i)$ assigned to i. We need to find a tree-partition for this set of clients. The above process assigning $\pi(j)$ for each client j was performed separately for the sets of clients $\mathcal{D}^*(l)$ assigned to each facility l in OPT. To form a tree-partition we combine the information obtained from performing the above procedure on all the optimal facilities.

For a particular facility i in LOC, let $E(i)$ be the set of all edges of T that clients from $\mathcal{D}(i)$ pass through. For example, the thick edges in Figure 6 belong to the set $E(a)$, where a is the facility of a_1, a_2 and a_3 in LOC. The set is the union of all such edges, from all the optimal facilities. $E(i)$ induces some connected components in $T_{\mathcal{D}(i)}$, which therefore define a tree-partition. We call this tree-partition of clients belonging to i in LOC $\{S^i_h\}$. For example, if LOC=OPT, then the facility $i = l$ is always a majority facility at all nodes of $T_{\mathcal{D}^*(l)}$ and so $E(i)$ contains all edges of the cost tree $T_{\mathcal{D}(i)} = T_{\mathcal{D}^*(i)}$. As a result, the tree-partition contains a single set with all of $T_{\mathcal{D}(i)}$ for each i.

Recall that the idea of the mapping π was to find for each client j assigned to a facility $i = f(j)$ in LOC a "partner" client $\pi(j)$, so that we can reassign j to the facility $f(\pi(j))$ when dispersing facility i. If we assign each client separately, then we may be paying certain facility costs multiple times. To avoid this, we want to assign all clients in a set S^i_h to one facility

$f(\pi(j))$ for some $j \in S_h^i$. A set of clients S_h^i in the tree-partition is assigned to the facility $f'(S_h^i) = i'$ from the set $\{i' : i' = f(\pi(j)) \textit{ for some } j \in S_h^i\}$ that minimizes $dist(i, i')$. For example, if $\pi(j) = j$ for any $j \in S_h^i$, then we set $i' = i$.

The tree-partition $\{S_h^i\}$ and the facility assignment $f'(S_h^i)$ define the desired specific disperse move for the facility i.

3.4 Bounding the facility cost: analysis.

We now give two lemmas that bound the facility and connection costs incurred when the sets of clients S_h^i in the partition defined above are transferred to their assigned facilities, $f'(S_h^i)$.

LEMMA 3.3. *The sum of incremental facility costs incurred when each S_h^i is transferred to $f'(S_h^i)$ is at most the optimal facility cost,*

$$\sum_{i \in \mathcal{F}} \sum_{S \in \{S_h^i\}} cost_{f'(S)}(S) \ \leq \ F^*.$$

Proof. Recall that $F^* = \sum_{l \in \mathcal{F}} \sum_{k \in T_{\mathcal{D}^*(l)}} cost(k)$. To prove the inequality, we divide the optimal cost F^* among the facilities i in LOC, and among the sets S_h^i in them, in such a way that each set's part is sufficient to pay the incremental cost of moving it to its new facility. This is done by adding each term in the definition of F^* to the budget of one of the sets.

A key observation is that when the mapping π is defined on a facility l used in OPT, all clients that pass through any given edge $(k,\ parent(k))$ belong to the same facility i in LOC. Moreover, they will be in the same component and set S_h^i (since all edges from such a client's leaf to k will be in $E(i)$). For any such edge, let us add $cost(k)$ to the *budget* of the set S_h^i, denoted B_h^i. Also, for any facility l in OPT that has clients assigned to themselves by π (for which no partner has been found), let us add $cost(root(T))$ to the budget of the set in the partition containing these clients.

The total amount given out in this way does not exceed F^*: for each facility l used in the optimum, the cost of each node k in its cost-tree $T_{\mathcal{D}^*(l)}$ is added to the budget of at most one set.

Next we show that the accumulated budget of a set S_h^i is sufficient for paying the incremental facility cost of transferring it to $f'(S_h^i)$, which proves the lemma. Consider the client $j \in S_h^i$ whose facility was chosen, i.e. such that $f(\pi(j)) = f'(S_h^i)$. First suppose that $\pi(j) = j$, i.e., j has no partner. We claim that in this case, B_h^i is sufficient to pay for the full $cost(S_h^i)$. Indeed, since the edges of any path from a client $j' \in S_h^i$ to the root are included in the connected component, the cost of their lower endpoints was added to B_h^i, and since $\pi(j) = j$, so was the cost of the root.

Now consider the case that $\pi(j) \neq j$. Then the least common ancestor of j and $\pi(j)$, say k, has to be in the connected component defining S_h^i. Then k and nodes above it in the tree are paid for at the facility $f'(S_h^i) = f(\pi(j))$. The path from a client $j' \in S_h^i$ to one of these nodes is paid for by B_h^i. □

LEMMA 3.4. *For the tree-partition $\{S_h^i\}$ and facilities $f'(S_h^i)$ defined above,*

$$\sum_{i \in \mathcal{F}} \sum_{S \in \{S_h^i\}} |S| \cdot dist(i, f'(S)) \ \leq \ 2C + 2C^*.$$

Proof. When defining $f'(S_h^i)$ we choose $f(\pi(j))$ with minimum distance to i, where $i = f(j)$ for all $j \in S_h^i$. So the left-hand side of the inequality is at most

$$\sum_{j \in \mathcal{D}} dist(f(j), f(\pi(j))).$$

To bound this expression recall that $f^*(\pi(j)) = f^*(j)$. Then by triangle inequality (see Figure 4) we get that

$$dist(f(j), f(\pi(j))) \leq C(j) + C^*(j) + C^*(\pi(j)) + C(\pi(j)).$$

Note that this bound is also valid when $\pi(j) = j$ as then $dist(f(j), f(\pi(j))) = 0$. Since the mapping π is 1-1 and onto, when this expression is summed over all j, we obtain

$$\sum_{j \in \mathcal{D}} dist(f(j), f(\pi(j))) \leq 2C + 2C^*,$$

proving the lemma. □

We conclude the analysis of the algorithm by combining the results obtained so far.

THEOREM 3.1. *A locally optimal solution LOC with no* aggregate *or* disperse *moves with negative value has cost at most 5 times the cost of the optimal solution.*

Proof. If we consider the above disperse move for a facility i in LOC, then the cost of the solution will change by an amount estimated using expression (3.2). Because LOC is a locally optimal solution, the net change in the cost is non-negative:

$$\sum_{S_h^i} cost_{f'(S_h^i)}(S_h^i) + \sum_{S_h^i} |S_h^i| dist(i, f'(S_h^i)) - F(i) \geq 0.$$

Summing these inequalities over all i and applying Lemmas 3.3 and 3.4, we get that $F^* + 2C + 2C^* - F \ \geq\ 0$, or $F \leq F^* + 2C^* + 2C$. Combining this with the bound on C (Lemma 2.2), we get $F \ \leq \ 3F^* + 4C^*$, and $F + C \leq 4F^* + 5C^*$. □

Using standard scaling techniques, we improve the bound to $2+\sqrt{5} \approx 4.236$. To do that, scale the original facility costs by λ and run the algorithm, obtaining a solution with the guarantees $C \leq \lambda F^* + C^*$ (by Lemma 2.2), and $\lambda F \leq 3\lambda F^* + 4C^*$ (see proof of Theorem 3.1). Combining these gives

$$C + F \leq (3+\lambda)F^* + (1+4/\lambda)C^*,$$

which yields the claimed result when λ is set to $\sqrt{5}-1$.

Further, by taking only *aggregate* and *disperse* moves with large negative values (as in, for example, [1]), we obtain a polynomial time $(4.236+\epsilon)$-approximation algorithm.

THEOREM 3.2. *There is a polynomial time $(4.236+\epsilon)$-approximation algorithm for the hierarchical facility location problem that is based on local search and uses* aggregate *and* disperse *moves.*

4 Conclusion

In this paper we consider the facility location problem with hierarchical facility costs, and give the first constant factor approximation algorithm for the problem independent of the number of levels in the hierarchy tree. The hierarchical facility costs are a special case of the setting in which the facility cost is a more complex function of the *set* of clients assigned to a single facility, and does not simply depend on their number. A more general class of such problems would be defined by using a facility cost that is an arbitrary submodular function $cost(S)$ of the set of clients S assigned to the facility. Submodularity represents a natural economy of scale in handling extra clients at an existing facility.

In the case of such general submodular facility costs, the natural *aggregate* move can still be implemented in polynomial time, as it is a special case of minimizing a submodular function $cost'(S) - b(S)$, as defined in Lemma 2.1. In fact, even the stronger *aggregate* move that evaluates the true change in costs when a set S of clients is moved to facility i (taking into account the savings incurred at other facilities) is a submodular function, hence can be minimized in polynomial time. Unfortunately, we don't know a natural extension of the *disperse* move for this general setting. Hence we leave it as an open problem if this generalization of the hierarchical facility location problem also has a polynomial time constant factor approximation algorithm.

Acknowledgments

We thank David Shmoys for insightful discussions and the suggestion to try the local search technique on this problem.

References

[1] V. Arya, N. Garg, R. Khandekar, K. Munagala, and V. Pandit. Local search heuristic for k-median and facility location problems. In *Proc. 33rd ACM Symp. on Theory of Computing*, pages 21–29, 2001.

[2] M. Charikar and S. Guha. Improved combinatorial algorithms for the facility location and k-median problems. In *Proc. 40th IEEE Symp. on Foundations of Computer Science*, pages 378–388, 1999.

[3] F. A. Chudak. Improved approximation algorithms for uncapitated facility location. In *IPCO*, pages 180–194, 1998.

[4] F. A. Chudak and D. B. Shmoys. Improved approximation algorithms for the uncapacitated facility location problem. *SIAM J. Comput.*, 33(1):1–25, 2003.

[5] Z. Drezner. *Facility Location: A Survey of Applications and Methods*. Springer, 1995.

[6] J. Feigenbaum, C. H. Papadimitriou, and S. Shenker. Sharing the cost of multicast transmissions. *J. Comput. Syst. Sci.*, 63(1):21–41, 2001.

[7] S. Guha and S. Khuller. Greedy strikes back: Improved facility location algorithms. *J. Algorithms*, 31(1):228–248, 1999.

[8] A. Hayrapetyan, C. Swamy, and E. Tardos. Network design for information networks. In *Proc. 16th ACM Symp. on Discrete Algorithms*, pages 933–942, 2005.

[9] K. Jain and V. V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. *J. ACM*, 48(2):274–296, 2001.

[10] M. R. Korupolu, C. G. Plaxton, and R. Rajaraman. Analysis of a local search heuristic for facility location problems. *J. Algorithms*, 37(1):146–188, 2000.

[11] M. Mahdian and M. Pál. Universal facility location. In *European Symposium on Algorithms*, pages 409–421, 2003.

[12] M. Mahdian, Y. Ye, and J. Zhang. Improved approximation algorithms for metric facility location problems. In *APPROX*, pages 229–242, 2002.

[13] D. Shmoys, C. Swamy, and R. Levi. Facility location with service installation costs. In *Proc. 15th ACM Symp. on Discrete Algorithms*, pages 1088–1097, 2004.

[14] D. B. Shmoys, E. Tardos, and K. Aardal. Approximation algorithms for facility location problems. In *Proc. 29th ACM Symp. on Theory of Computing*, pages 265–274, 1997.

[15] M. Sviridenko. An improved approximation algorithm for the metric uncapacitated facility location problem. In *IPCO*, pages 240–257, 2002.

[16] P. von Rickenbach and R. Wattenhofer. Gathering correlated data in sensor networks. In *DIALM-POMC '04: Proceedings of the 2004 Joint Workshop on Foundations of Mobile Computing*, pages 60–66, 2004.