

Facility Location with Hierarchical Facility Costs*

Zoya Svitkina[†]

Éva Tardos[‡]

Abstract

We introduce a facility location problem with submodular facility cost functions, and give an $O(\log n)$ approximation algorithm for it. Then we focus on a special case of submodular costs, called hierarchical facility costs, and give a $(4.237 + \epsilon)$ -approximation algorithm using local search. The hierarchical facility costs model multilevel service installation. Shmoys, Swamy and Levi [18] gave a constant factor 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.

In this paper, we study a variant of the facility location problem in which the cost of a facility depends on the specific set of clients assigned to that facility. We propose a general model in which the facility costs are submodular functions of the set of clients assigned to the facility, and give an $O(\log n)$ -approximation algorithm for it, where n is the number of clients. Submodularity of the cost functions models a natural economy of scale. We then focus on a subclass of submodular cost functions which we call hierarchical costs. Shmoys, Swamy and Levi [18] and Ravi and Sinha [16] 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 $(4.237 + \epsilon)$ -approximation (independent of the number of levels in the hierarchy) in the case that the costs are identical at all facilities.

Our models and methods. We introduce the general problem of facility location with submodular facility costs, and then focus on a special case, called facility location with hierarchical facility costs. A *submodular*

*A preliminary version of this paper has appeared in the Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, 2006

[†]Cornell University, Department of Computer Science, Upson Hall, Ithaca, NY 14853. Supported in part by ONR grant N00014-98-1-0589.

[‡]Cornell University, Department of Computer Science, Upson Hall, Ithaca, NY 14853. Supported in part by NSF grant CCR-0311333, ITR grant CCR-0325453, and ONR grant N00014-98-1-0589.

function $g : 2^{\mathcal{D}} \rightarrow \mathbb{R}$ on a set \mathcal{D} is one that satisfies the inequality $g(A) + g(B) \geq g(A \cup B) + g(A \cap B)$ for any $A, B \subseteq \mathcal{D}$. Equivalently, for any $A \subseteq B \subset \mathcal{D}$ and an element $x \notin B$, it satisfies $g(A \cup \{x\}) - g(A) \geq g(B \cup \{x\}) - g(B)$. Submodularity models decreasing marginal costs and is a natural property of functions in many settings.

Definition 1.1 Facility location with submodular facility costs problem has as input a set of facilities \mathcal{F} , a set of demands or clients \mathcal{D} , a distance metric dist on the set $\mathcal{F} \cup \mathcal{D}$, and a monotone non-decreasing submodular function $g_i : 2^{\mathcal{D}} \rightarrow \mathbb{R}^+$ for each facility $i \in \mathcal{F}$. The goal is to find a facility $f(j) \in \mathcal{F}$ for each client $j \in \mathcal{D}$ so as to minimize the sum of the connection cost, $\sum_{j \in \mathcal{D}} \text{dist}(j, f(j))$, and the facility cost, $\sum_{i \in \mathcal{F}} g_i(\mathcal{D}(i))$, where $\mathcal{D}(i) = \{j \in \mathcal{D} : f(j) = i\}$ denotes the set of clients assigned to facility i .

We give an $O(\log |\mathcal{D}|)$ -approximation algorithm for facility location with submodular facility costs in Section 2, using a greedy algorithm with a submodular function minimization subroutine.

In the rest of the paper we focus on a special case of this problem, in which the form of the functions g_i is restricted to what we call hierarchical cost functions. It is not hard to see that these hierarchical functions are in fact submodular. Moreover, we assume that the cost functions of all facilities are the same, i.e. $g_i = g$ for all $i \in \mathcal{F}$. Note that the problem with independent cost functions on different facilities is *set cover*-hard, as was shown in [18]. The hierarchical facility cost function g is specified by a rooted cost tree T , whose set of leaves is \mathcal{D} (the set of clients). The leaves of the tree can also be thought of as the services that the corresponding clients request. Each node k of T has a non-negative cost, $\text{cost}(k)$. The facility cost function g is defined using the tree T as follows. For a set $\mathcal{D}(i) \subseteq \mathcal{D}$ of clients assigned to facility i , we use the notation $T_{\mathcal{D}(i)}$ to denote the subgraph of T induced by the nodes that lie on a path from the root to some leaf (client) in $\mathcal{D}(i)$ (see Figure 1). Then the facility cost of i is $\sum_{k \in T_{\mathcal{D}(i)}} \text{cost}(k)$. We also use $\text{cost}(S)$ as a shorthand for

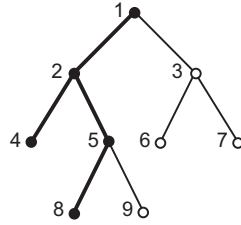


Figure 1: Example of a cost tree. If clients corresponding to leaves 4 and 8 form the set $\mathcal{D}(i)$ which is assigned to a facility i , then the cost of i is $F(i) = \text{cost}(1) + \text{cost}(2) + \text{cost}(4) + \text{cost}(5) + \text{cost}(8)$. Shaded nodes and thick edges form the subgraph $T_{\mathcal{D}(i)}$.

the facility cost of a set of clients $S \subseteq \mathcal{D}$, i.e. $\text{cost}(S) = \sum_{k \in T_S} \text{cost}(k)$. The cost of an empty facility is zero.

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 4.237 by scaling. The algorithm starts with an arbitrary assignment of clients to facilities, and performs two types of local improvement moves, *aggregate* and *disperse*, while the objective function of the solution improves. 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 Section 3, where we also prove that if there are no improving aggregate moves, then the connection cost of the 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 4 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 4.237, and accepting only sufficiently large improvements, we turn this into a polynomial-time $(4.237 + \epsilon)$ -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 cost functions are allowed to differ from each other by constant factors can model data gathering in sensor networks (see, for example, [22]). 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 [13, 2, 1], primal-dual method [12], and linear programming and rounding [19, 3, 4, 20]. The current best approximation guarantee is 1.52 [15], and no better than 1.463 approximation is possible unless $NP \in DTIME[n^{O(\log \log n)}]$ [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 [12, 2, 1, 4], where soft capacities allow us to open a facility multiple times, paying the opening cost each time, 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 [8, 14, 6], 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. Shmoys, Swamy and Levi [18] and Ravi and Sinha [16] introduced the facility location problem with service installation costs (called multicommodity facility location in [16]), where the costs can be represented by a tree of height two. Ravi and Sinha consider the version of the problem where the facility costs can be arbitrarily different for different facilities, and they develop a logarithmic-factor approximation, which is the best possible in that case. Shmoys et al. restrict the model by assuming that facilities have an ordering where each earlier facility costs less than a later one for all services, which allows them to design a constant-factor approximation. We make an even more restrictive assumption of facility costs being identical on all facilities. However, in contrast to the two-level model, our algorithm works for arbitrary number of levels and yields a bound of 5, independent of the number of levels in the tree.

The group facility location problem of Hayrapetyan, Swamy and Tardos [9] 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. At each step of the local search, the algorithm has some assignment of clients to the facilities which constitutes its current solution. We call this solution SOL. For the analysis of the algorithm, we assume that SOL is a locally optimal solution, and we compare it to a fixed globally optimal one, which we call OPT. Let C and F denote the connection and facility costs of SOL, 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 SOL 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 SOL, 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 Approximation for submodular costs

In this section we present a logarithmic approximation for facility location with submodular facility costs. The procedure that we present works for the case that the submodular facility cost functions are different for different facilities, and does not require the distances to form a metric. The guarantee is the best possible for this case, because either one of these two generalizations makes the problem *set cover*-hard. However, for the case of identical facility costs and metric distances, we do not know of any lower bounds that would rule out the possibility of a constant-factor approximation.

Our algorithm is obtained by a reduction to the set cover problem, with a polynomial-time subroutine for implementing the greedy algorithm on this instance with exponentially many sets. Such an approach has been used for the plain uncapacitated facility location problem by Hochbaum [10], and is mentioned in [16] as a possibility for obtaining a logarithmic approximation for the facility location with service installation. Thus, our main contribution is to show how to find the best facility and a set of clients for one step of the greedy set cover algorithm.

Facility location can be viewed as an instance of set cover in the following way. The set of clients \mathcal{D} is the set of elements to be covered. For each facility $i \in \mathcal{F}$ and each possible subset of clients $S \subseteq \mathcal{D}$, there is a set $A(i, S)$ in the set cover instance, covering the elements in S , whose cost is $g_i(S) + \sum_{j \in S} dist(i, j)$, the facility and connection cost of assigning clients S to facility i . Because of monotonicity of the functions g_i , any set cover solution can be transformed into one in which the selected sets are disjoint, thus forming a feasible facility location solution, without increase in cost.

A greedy algorithm that repeatedly selects a set minimizing the ratio of the set's cost to the number of newly-covered elements is well-known to be a $O(\log n)$ -approximation for the set cover problem [21], where n is the number of elements, which, in our case, is the number of clients. However, in our case such a set cannot be found by simple enumeration, because the instance of set cover that we describe has a number of sets exponential in the number of clients of the corresponding facility location instance. So we describe a procedure that finds a set $A(i, S)$ minimizing the ratio of its cost to the number of newly-covered clients in polynomial time, using submodular function minimization [11].

At each step of the greedy algorithm, in order to keep track of which clients have already been covered and which have not, let us assign a weight $w_j = 0$ for all covered clients j , and a weight $w_j = 1$ for all clients that have not been covered yet. To find the best facility i for the set $A(i, S)$, we simply try all of them, finding the best set S for each one, and then select the one with the best ratio. So for a given facility i , the task is to find a set $S \subseteq \mathcal{D}$ minimizing $\frac{g_i(S) + \sum_{j \in S} dist(i, j)}{\sum_{j \in S} w_j}$, where the denominator is just the number of clients in S that have not been covered yet. To minimize this ratio, we can do a binary search for the minimum value α for which there exists a set S such that $\frac{g_i(S) + \sum_{j \in S} dist(i, j)}{\sum_{j \in S} w_j} \leq \alpha$, or, equivalently, $g_i(S) + \sum_{j \in S} dist(i, j) - \alpha \cdot \sum_{j \in S} w_j \leq 0$. The left-hand side of this last expression is a submodular function, as g_i is submodular by assumption, and the last two terms are modular functions (i.e. ones for which the submodular inequality holds with equality). Thus it can be minimized in polynomial time.

3 Aggregate move and the connection cost

In this section we consider one of the two local improvement moves that are performed by the local search algorithm for facility location with hierarchical costs. When this move, *aggregate*, is performed on a facility i , it transfers some clients from other facilities to i . This change in assignment of clients affects the cost of the solution in three ways: the clients that change assignments have a changed connection cost; facility i has an increased facility cost; and other facilities have decreased facility costs as some clients move away from

them. As we show at the end of this section, it is possible to find, in polynomial time, the optimal set of clients to be transferred to facility i , even for the case of general submodular facility cost functions.

First, we present a simpler approximate way of evaluating an aggregate move, and a much more efficient algorithm for finding the best set of clients according to this approximate evaluation. 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

$$\sum_{j \in S} \text{dist}(j, i) - \sum_{j \in S} \text{dist}(j, f(j)) + \text{cost}_i(S). \quad (1)$$

We call (1) the *value* of the aggregate move (despite the fact that expression (1) does 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 (1) for a given facility i . Then we show that if a solution has no improving aggregate moves, its connection cost can be bounded.

3.1 Finding the aggregate move with optimal value

Consider the problem of finding a set of clients S that minimizes expression (1) for a particular facility i . The change in distance is equal to $\text{dist}(j, i) - \text{dist}(j, f(j))$ for each client j in the set S that we choose, and the added facility cost at i depends on S as well as the set of clients $\mathcal{D}(i)$ already at i : $\text{cost}_i(S) = \sum_{k \in T_S \setminus T_{\mathcal{D}(i)}} \text{cost}(k)$. Let us now construct a modified tree T' from T which will be useful for designing the algorithm. The structure of T' is the same as that of T ; the only difference is in the costs associated with the nodes. We incorporate the change in connection cost associated with each client j into the cost tree, so that the algorithm that makes a pass over this tree would be able to take into account both the changes in the connection costs and in the facility costs simultaneously. In particular, for every client $j \in \mathcal{D}$, set the cost of its leaf to $\text{cost}_{T'}(j) = \text{cost}_T(j) + \text{dist}(j, i) - \text{dist}(j, f(j))$, where $\text{cost}_T(j)$ is the original cost of this leaf in the tree T . Note that this new cost may be negative. Also, we discount the tree nodes which are already paid for at i : for these nodes $k \in T_{\mathcal{D}(i)}$, set $\text{cost}_{T'}(k) = 0$. For all other nodes k we keep the original cost $\text{cost}_{T'}(k) = \text{cost}_T(k)$. The reason that the construction of T' is helpful is explained in the following lemma.

Lemma 3.1 *There exists an improving aggregate move for facility i if and only if there is a connected subgraph U of T' , containing the root, such that the total cost of nodes in U is negative, i.e. $\sum_{k \in U} \text{cost}_{T'}(k) < 0$.*

Proof. Suppose there is a set of clients S such that moving S to i constitutes an improving aggregate move. Then the subgraph of T' which consists of all leaves that correspond to clients in S and the union of their paths to the root has total node cost exactly equal to expression (1), and therefore negative.

Conversely, suppose that there is a connected subgraph U of T' , containing the root, with negative total node cost. Since the only nodes with negative cost are the leaves, we can assume without loss of generality that U consists of a union of paths from the root to a subset of leaves (any internal nodes of T' that are in U but do not have descendants in U can be removed without increasing the cost). Then we take S to be the clients corresponding to the leaves in U and observe that the value of moving S to i is equal to the total cost of nodes in U . \square

Given this lemma, the problem of finding the set of clients S minimizing expression (1) reduces to the problem of finding a minimum-cost connected component of T' containing the root. The latter problem can be solved by a simple dynamic programming procedure. In a bottom-up pass through the tree, for each node k we find the cheapest subgraph of the subtree rooted at k which contains k . For a leaf node k , this subgraph is the node itself, and its cost is $\text{cost}_{T'}(k)$. For an internal node k , the cheapest subgraph contains k itself, as well as those subgraphs of its children whose costs are negative.

The above algorithm, together with Lemma 3.1, yields the following results.

Lemma 3.2 *The subset $S \subseteq \mathcal{D}$ that minimizes expression (1), and hence defines the aggregate move of minimum value for a given facility i , can be found in time $O(|T|)$. The aggregate move of minimum value over all facilities can therefore be found in time $O(|\mathcal{F}| \cdot |T|)$.*

3.2 Bounding the connection cost

Now consider a solution 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 3.3 *The connection cost C of a locally optimal solution can be bounded by the optimal cost as $C \leq C^* + F^*$.*

Proof. If there are no improving aggregate moves, then expression (1) is nonnegative for moving *any* set of clients to *any* facility i . We consider expression (1) for the set of clients $\mathcal{D}^*(i)$ that are assigned to i in OPT. We have that

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

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

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

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

3.3 Aggregate move for general submodular functions

In this section we show that finding the optimal aggregate move for the facility location problem with submodular facility costs can be done in polynomial time. We do this by showing that for a particular facility i , finding a set of clients S which would minimize the cost of the resulting solution when transferred to i can be done using submodular function minimization. This optimization is done using exact measure of the change in solution cost, without making the simplification of ignoring costs at facilities other than i , as done in expression 1.

If $\mathcal{D}(i)$ is the set of clients currently at the facility i , then the goal is to find a set $S \subseteq \mathcal{D} \setminus \mathcal{D}(i)$ of clients which are currently at other facilities to be transferred to facility i so as to minimize the cost of the resulting solution, or, equivalently, to minimize the difference between the cost of the new solution and the cost of the current solution. This change in cost can be expressed as

$$\begin{aligned} \Delta(S) &= g_i(\mathcal{D}(i) \cup S) - g_i(\mathcal{D}(i)) + \sum_{j \in S} [\text{dist}(j, i) - \text{dist}(j, f(j))] \\ &\quad + \sum_{i' \neq i} [g_{i'}(\mathcal{D}(i') \setminus S) - g_{i'}(\mathcal{D}(i'))], \end{aligned}$$

where g_i is the facility cost of facility i , the sum over clients in S is the change in the connection cost, and the last sum is the change in facility costs of facilities other than i . We show that $\Delta(S)$ is a submodular function over the set $\mathcal{D} \setminus \mathcal{D}(i)$, which allows us to use a polynomial-time submodular function minimization algorithm [11, 17] for finding the optimal set S of clients to be transferred.

Lemma 3.4 *$\Delta(S)$ for $S \subseteq \mathcal{D} \setminus \mathcal{D}(i)$ is a submodular function.*

Proof. To show that $\Delta(S)$ is submodular, it suffices to show that it consists of a sum of several submodular functions. To verify the submodularity of the first term, $g_i(\mathcal{D}(i) \cup S)$, we observe that

$$\begin{aligned} g_i(\mathcal{D}(i) \cup (A \cup B)) + g_i(\mathcal{D}(i) \cup (A \cap B)) &= \\ &= g_i((\mathcal{D}(i) \cup A) \cup (\mathcal{D}(i) \cup B)) + g_i((\mathcal{D}(i) \cup A) \cap (\mathcal{D}(i) \cup B)) \\ &\leq g_i(\mathcal{D}(i) \cup A) + g_i(\mathcal{D}(i) \cup B), \end{aligned}$$

where the inequality follows by submodularity of g_i .

The second term, $-g_i(\mathcal{D}(i))$, is just a constant, as it does not depend on S . The sum $\sum_{j \in S} [dist(j, i) - dist(j, f(j))]$ representing the change in distances is a modular function. The terms $-g_{i'}(\mathcal{D}(i'))$ are constant as well.

For terms of the form $g_{i'}(\mathcal{D}(i') \setminus S)$, we get

$$\begin{aligned} g_{i'}(\mathcal{D}(i') \setminus (A \cup B)) + g_{i'}(\mathcal{D}(i') \setminus (A \cap B)) &= \\ &= g_{i'}((\mathcal{D}(i') \setminus A) \cap (\mathcal{D}(i') \setminus B)) + g_{i'}((\mathcal{D}(i') \setminus A) \cup (\mathcal{D}(i') \setminus B)) \\ &\leq g_{i'}(\mathcal{D}(i') \setminus A) + g_{i'}(\mathcal{D}(i') \setminus B), \end{aligned}$$

by submodularity of $g_{i'}$, showing that $\Delta(S)$ is a submodular function. \square

The ability to find the optimal aggregate move allows us to bound the connection cost of a solution which is locally optimal with respect to this move for the facility location problem with submodular facility costs. This is done in the same way as in Lemma 3.3 for hierarchical costs. Unfortunately, though, we do not know of an analogue to the disperse move that would work for this more general problem.

4 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.

4.1 Definition of a disperse move

The idea of the disperse move is to move some of the clients from a particular 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 a solution that assigns all clients 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. We use approximate evaluations both for the change in connection costs, which we bound in the usual way with a triangle inequality, and for the change in facility costs, for which we have a more complex scheme. For both the connection and the facility costs, the estimated change in solution cost that we use is an upper bound on the true change in cost.

We consider removing all clients from facility i and distributing them among all the facilities, possibly putting some clients back on the (now empty) facility i . This operation affects the cost of the solution in the following ways: there is a change in connection cost for clients that are moved; facility cost decreases at i and increases at the facilities where the clients are placed. Using the triangle inequality, we upper bound the change in connection cost for each client by the distance between i and that client's new facility. The decrease in facility cost at i is just its whole cost in the current solution, $F(i)$. For the estimation of increase in facility costs we define the notion of a tree-partition.

A *tree-partition* $\mathcal{S}(\mathcal{D}(i)) = \{S_h\}$ of a set of clients $\mathcal{D}(i)$ is any partition that can be obtained by cutting a subset of edges of the tree $T_{\mathcal{D}(i)}$ and grouping the clients by the resulting connected components of their tree leaves (see Figure 2). For example, if no edges are cut, then all clients are in the same set; if all edges incident on leaves are cut, then each client is in a separate set. Let us refer to the connected component

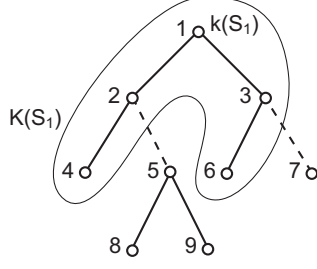


Figure 2: Example of a tree-partition. Cutting the two dashed edges produces three subsets: $S_1 = \{4, 6\}$, $S_2 = \{8, 9\}$, and $S_3 = \{7\}$. The marked component is $K(S_1)$.

associated with a set S_h as $K(S_h)$. A *disperse move* for facility i can be specified as a tuple (i, \mathcal{S}, f') , where $\mathcal{S}(\mathcal{D}(i))$ is a tree-partition of the set of clients from i , and $f' : \mathcal{S}(\mathcal{D}(i)) \rightarrow \mathcal{F}$ is an assignment specifying the facility $f'(S_h)$ to which each set S_h should be reassigned. It is possible that $f'(S_h) = i$ for some sets.

Returning to the evaluation of a disperse move (i, \mathcal{S}, f') , we estimate the facility cost of moving a set S_h to its new facility $f'(S_h)$ using the incremental cost incurred by adding clients in S_h to the clients already at this facility $f'(S_h) = i'$, which is $cost_{i'}(S_h) = cost(\mathcal{D}(i') \cup S_h) - cost(\mathcal{D}(i'))$. For different sets added to the same facility, we simply sum their incremental costs, which upper bounds the true increase in facility costs. More precisely, the true increase in cost at a facility i' is

$$cost \left(\mathcal{D}(i') \cup \bigcup_{h: f'(S_h)=i'} S_h \right) - cost(\mathcal{D}(i')),$$

whereas we estimate it as

$$\sum_{h: f'(S_h)=i'} [cost(\mathcal{D}(i') \cup S_h) - cost(\mathcal{D}(i'))].$$

This is an upper bound on the true increase by submodularity of the cost function. Facility i , from which the clients were just removed, is treated as empty, i.e. $cost_i(S_h) = cost(S_h)$. Thus, the overall upper bound on the change in solution cost resulting from the disperse move (i, \mathcal{S}, f') can be expressed as

$$value(i, \mathcal{S}, f') = \sum_{S_h \in \mathcal{S}(\mathcal{D}(i))} cost_{f'(S_h)}(S_h) + \sum_{S_h \in \mathcal{S}(\mathcal{D}(i))} |S_h| \cdot dist(i, f'(S_h)) - F(i), \quad (2)$$

where the first term is the estimate of the increase in facility costs, the second term is the estimate of the change in connection cost, and $F(i)$ is the cost of facility i which is saved when clients are removed from it. This expression defines the *value* of a disperse move. Any move with negative value is called an *improving* move. Next we show how to find a disperse move with minimum value in polynomial time.

4.2 Finding the disperse move with optimal value

We begin by proving a lemma that is useful for deriving the algorithm.

Lemma 4.1 *There exists a disperse move (i, \mathcal{S}, f') of minimum value such that for all sets S_h of the tree-partition \mathcal{S} , none of the nodes of the subgraph $K(S_h)$ are paid for at this set's new facility $f'(S_h)$.*

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 the subgraph $K(S_h)$ is paid for at facility $f'(S_h)$, then so is its top node, call it $k(S_h)$, since it lies on the path from a paid node to the root. Since each client corresponds to a different leaf of the tree 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 the 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 may split S_h into multiple sets, all of which we assign to the same facility $f'(S_h)$ as S_h was

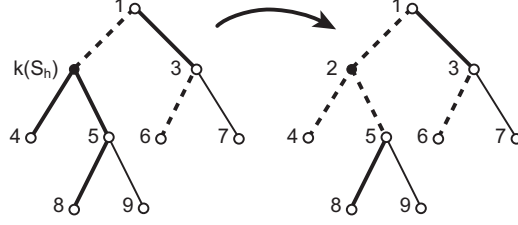


Figure 3: Operation performed in the proof of Lemma 4.1. Suppose that S_h consists of clients 4 and 8, and that node 2 is paid for at the facility $f'(S_h)$.

assigned to. The new disperse move has value no greater 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 in a top-down manner, eliminating each set that does not satisfy the condition. \square

The algorithm for finding the optimal disperse move satisfying the conditions of Lemma 4.1 is based on dynamic programming. Recall that $T_{\mathcal{D}(i)}$ is the subgraph of the cost tree T for which the tree-partition has to be found, because it represents the facility costs for clients which are currently at i and which have to be dispersed. Since the dynamic programming will proceed by considering various subtrees of this tree, let the notation $T_{\mathcal{D}(i)}(k)$ stand for the subtree of $T_{\mathcal{D}(i)}$ rooted at node k .

For any node $k \in T_{\mathcal{D}(i)}$, we define three parameters, $N(k)$, $Y(k)$, and $P(k)$, that describe the part of the solution related to the subtree $T_{\mathcal{D}(i)}(k)$. First we describe what the values of these parameters are for a given disperse move (i, \mathcal{S}, f') and a given node k , and then we use these parameters to construct a dynamic programming table for determining the optimal disperse move. For a given node k and a disperse move with its tree-partition \mathcal{S} , the components of this tree-partition that are relevant to determining the values of the parameters are the components which are entirely contained in the subtree rooted at k , and at most one special component which contains the edge between k and its parent. For example, if k is node 2 in Figure 2, then its special component is $K(S_1)$, as it contains the edge between node 2 and its parent, node 1. Let us call this special component with respect to the tree node k , if it exists, $K(S^k)$, and the set of clients contained in it S^k . Then the three parameters are

- $N(k) = |S^k \cap T_{\mathcal{D}(i)}(k)|$ is the number of clients from node k 's special component $K(S^k)$ that are in the subtree rooted at k . For example, for node $k = 2$ in Figure 2, $N(k) = 1$, as only client 4 satisfies this property. In the algorithm, this parameter is used to keep track of the connection cost that will have to be paid when S^k is transferred to its new facility.
- $Y(k) = \sum_{k' \in U} \text{cost}(k')$, where $U = T_{S^k} \cap T_{\mathcal{D}(i)}(k)$. $Y(k)$ is the part of the facility cost for clients in the special component S^k which comes from the subtree rooted at k . In other words, if U is the 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 . For example, in Figure 2, the special component for node $k = 2$ is $K(S_1)$, and the only client which is included both in this component and in the subtree rooted at node 2 is client 4. So the cost for client 4 which comes from the subtree is $Y(2) = \text{cost}(4) + \text{cost}(2)$. In the algorithm, $Y(k)$ is used to keep track of the facility cost that will have to be paid when k 's special component $K(S^k)$ is transferred to its new facility. Note an important property that $N(k) = 0$ implies that $Y(k) = 0$.
- $P(k)$ is the connection and facility costs of reassigning sets S_h of the tree-partition, that are contained entirely in the subtree rooted at k , to their new facilities. Formally,

$$P(k) = \sum_{S_h: K(S_h) \subseteq T_{\mathcal{D}(i)}(k)} [|S_h| \cdot \text{dist}(i, f'(S_h)) + \text{cost}_{f'(S_h)}(S_h)].$$

In Figure 2, $P(2)$ includes the connection cost and the facility cost of sending clients 8 and 9 to their new facility.

The dynamic programming algorithm for computing the optimal disperse move for facility i is shown in figure as Algorithm 1. It constructs a table A whose entries, $A_k(x)$, indexed by the nodes k of $T_{\mathcal{D}(i)}$ and integers x , contain the minimum values of $Y(k) + P(k)$ over all disperse moves on i that satisfy Lemma 4.1 and have $N(k) = x$. The reason that values $A_k(x)$ are interesting is that the overall minimum value of a disperse move on facility i is equal to $A_r(0) - F(i)$, where r is the root of T . 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.

The algorithm considers the nodes of the tree starting from the leaves, and for each node considers the possibilities of cutting or not cutting the edge between this node and its parent. The idea is that for constructing the disperse move at a node one level higher in the tree, it suffices to know the two parameters, N and $Y + P$, about the solutions at the subtrees of its children. The costs of assignments for components entirely contained lower in the tree are accounted for by P , the facility cost for the unassigned special components of the children nodes is accounted for by Y , and their connection costs are determined by the number of clients in them, N . After initializing these values for the leaves (see lines 3-5 of Algorithm 1), the algorithm considers an internal node k with $l = 1$ or 2 children. To determine $A_k(x)$ for a value $x > 0$, the corresponding solution cannot cut the edge connecting k to its parent. To get such a solution, we need to partition the number of clients x between the components corresponding to the children's subtrees. Let k_1, \dots, k_l denote the children of k . We consider $A_{k_c}(x_c)$ for every partition $x = \sum_c x_c$. The facility costs of the solutions in the subtrees are combined to get the solution for the tree rooted at k , so the cost is $\sum_c A_{k_c}(x_c) + \text{cost}(k)$, and we select the partition in line 12 that minimizes this sum.

For the case of $x = 0$ there are two possibilities for the solution that achieves the minimum value $A_k(0)$. Either $N(k_c) = 0$ for all children k_c of k , or, if not, then the edge above k must be cut to combine the special components of the children into a new component containing k . We get the minimum value of the first case by taking $\sum A_{k_c}(0)$ (as done in line 9). An alternate way is to combine the special components of the subtrees into a single (not special) component by cutting the edge connecting k to its parent. For this case we must consider all values x_c and let $x = \sum_c x_c$. The set S_h corresponding to the component of the tree-partition containing k must be sent to some facility, say i' . The connection cost for this is $|S_h| \cdot \text{dist}(i, i')$, but notice that $|S_h| = x$, and we now have to find the lowest-cost facility i' where the new component (containing k) should be sent. Because of Lemma 4.1, the facility cost is $\sum_{c=1}^l Y(k_c) + \text{cost}(\text{Path}(k, \text{root}) \setminus T_{\mathcal{D}(i')})$, where $\text{Path}(k, \text{root}) \setminus T_{\mathcal{D}(i')}$ is just the set of nodes on the path between k and the root (inclusive) which are not paid for at the facility i' . As a result, we have that $Y(k) = 0$ and $P(k)$ is the sum of all $P(k_c)$ plus the new connection and facility costs of moving the set S_h to i' . That is,

$$\begin{aligned} Y(k) + P(k) &= \\ &= 0 + \sum_{c=1}^l P(k_c) + x \cdot \text{dist}(i, i') + \sum_{c=1}^l Y(k_c) + \text{cost}(\text{Path}(k, \text{root}) \setminus T_{\mathcal{D}(i')}) \\ &\geq \sum_{c=1}^l A_{k_c}(x_c) + x \cdot \text{dist}(i, i') + \text{cost}(\text{Path}(k, \text{root}) \setminus T_{\mathcal{D}(i')}). \end{aligned}$$

The minimum of these values are computed in line 15 of the algorithm.

We now give two lemmas regarding the correctness and running time of Algorithm 1.

Lemma 4.2 *For all $k \in T_{\mathcal{D}(i)}$ and $x \in \{0 \dots |\mathcal{D}(i)|\}$, $A_k(x)$ found by Algorithm 1 is the minimum value of $Y(k) + P(k)$ over all disperse moves on i that satisfy Lemma 4.1 and have $N(k) = x$.*

Proof. We use induction on the height of the subtree rooted at k . The base case is when k is a leaf of $T_{\mathcal{D}(i)}$, which means that it corresponds to some client in $\mathcal{D}(i)$, say j . Since there is only one client in $T_{\mathcal{D}(i)}(k)$, $N(k)$ can only take values 0 or 1. All disperse moves for which $N(k) = 1$ do not cut the edge above k , and have $P(k) = 0$ and $Y(k) = \text{cost}(k)$, so $A_k(1) = \text{cost}(k)$ (as set in line 3 of the algorithm) is indeed the minimum value of $Y(k) + P(k)$ for moves with $N(k) = 1$. Only the move that cuts the edge between k and its parent has $N(k) = 0$. If the resulting set $S_h = \{j\}$ is sent to a facility i' , then $Y(k) = 0$ and $P(k) = \text{dist}(i, i') + \text{cost}_{i'}(\{j\})$, so the minimum of $Y(k) + P(k)$ is obtained by choosing i' that achieves the minimum in line 5 of the algorithm.

Algorithm 1 Finding optimal disperse move for facility i

```
1: Initialize  $A_k(x) \leftarrow \infty$  for all  $k \in T_{\mathcal{D}(i)}$  and all  $x \in \{0 \dots |\mathcal{D}(i)|\}$ 
2: for all leaves  $k$  of  $T_{\mathcal{D}(i)}$  do
3:    $A_k(1) \leftarrow \text{cost}(k)$  // case when edge  $(k, \text{parent}(k))$  is not cut
4:   Let  $j$  be the client associated with  $k$ 
5:    $A_k(0) \leftarrow \min_{i' \in \mathcal{F}} (\text{dist}(i, i') + \text{cost}_{i'}(\{j\}))$  // edge cut,  $j$  sent to  $i'$ 
6: end for
7: for all internal nodes  $k \in T_{\mathcal{D}(i)}$ , bottom-up do
8:   Let  $k_1, \dots, k_l$  be the children of  $k$  in  $T_{\mathcal{D}(i)}$ 
9:    $A_k(0) \leftarrow \sum_{c=1}^l A_{k_c}(0)$  // no clients in children's special components
10:  for all  $x_1, \dots, x_l$  such that  $1 \leq \sum_{c=1}^l x_c \leq |\mathcal{D}(i)|$  do
11:    Let  $x = \sum_{c=1}^l x_c$  // number of clients from children's components
12:     $A_k(x) \leftarrow \min(A_k(x), \sum_{c=1}^l A_{k_c}(x_c) + \text{cost}(k))$  // case when edge  $(k, \text{parent}(k))$  is not cut
13:  for all  $i' \in \mathcal{F}$  such that  $k \notin T_{\mathcal{D}(i')}$  do
14:    Let  $U_{i'} = \text{Path}(k, \text{root}) \setminus T_{\mathcal{D}(i')}$  // nodes on the path from  $k$  to the root which are not paid for at  $i'$ 
15:     $A_k(0) \leftarrow \min(A_k(0), \sum_{c=1}^l A_{k_c}(x_c) + x \cdot \text{dist}(i, i') + \text{cost}(U_{i'}))$  // edge  $(k, \text{parent}(k))$  is cut, clients sent to  $i'$ 
16:  end for
17: end for
18: end for
```

For an internal node k we have argued that $A_k(x) \leq \min\{Y(k) + P(k) : N(k) = x\}$ for all integers x while constructing the algorithm.

To prove the induction step of the other direction, $A_k(x) \geq \min\{Y(k) + P(k) : N(k) = x\}$, we show that for all k and x such that $A_k(x) < \infty$, there exists a disperse move on i , satisfying Lemma 4.1, for which $Y(k) + P(k) = A_k(x)$ and $N(k) = x$. For an internal node k , let us consider several cases depending on which line of the algorithm produced the final value of $A_k(x)$. If the value was produced by line 9, then combining the solutions (which exist by induction hypothesis) corresponding to $A_{k_c}(0)$ for all children k_c of k , and leaving the edge above k intact, gives the desired result. If the value of $A_k(x)$ was produced by line 12 in the iteration of the loop corresponding to x_1, \dots, x_l , then combine the solutions corresponding to $A_{k_c}(x_c)$ from the induction hypothesis, and leave the edge between k and its parent intact. Since in this case $N(k) = \sum_c N(k_c)$, $P(k) = \sum_c P(k_c)$ and $Y(k) = \sum_c Y(k) + \text{cost}(k)$, $Y(k) + P(k)$ is equal to $A_k(x)$ as produced by line 12. The last case is if the value of $A_k(x)$ resulted from line 15 when executed in the loop for facility i' . In this case we again combine solutions corresponding to $A_{k_c}(x_c)$ of k 's children, but this time cut the tree edge above k and send the resulting set of clients to the facility i' . It can be verified that the value of $Y(k) + P(k)$ will be equal to $A_k(x)$ as produced by line 15 of the algorithm. \square

Lemma 4.3 *Algorithm 1 runs in time $O(|\mathcal{D}|^3 \cdot |\mathcal{F}|)$.*

Proof. The assumption that T is binary implies that $l \leq 2$, which means that the **for** loop on line 10 executes at most $|\mathcal{D}|^2$ times each time it is entered. The loops on lines 7 and 13 execute at most $|T|$ and $|\mathcal{F}|$ times respectively, and $|T| = O(|\mathcal{D}|)$. \square

Combining the above results and observing that the minimum value of a disperse move for a facility i is equal to $A_r(0) - F(i)$, where r is the root of T , we get the following theorem.

Theorem 4.4 *The disperse move that minimizes expression 2 for a given facility can be found in polynomial time $O(|\mathcal{D}|^3 \cdot |\mathcal{F}|)$, and the one with minimum value over all facilities can be found in time $O(|\mathcal{D}|^3 \cdot |\mathcal{F}|^2)$.*

4.3 Bounding the facility cost: a specific set of disperse moves

The way we bound facility cost of a solution SOL which is locally optimal with respect to the disperse move 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], who use it to analyze local search with *swap* moves for the k -median problem and local search with *open*, *close* and *swap* moves for the facility location problem. However, in their case, no cost trees are involved, and all clients are the same from the point of view of the facility cost, so the definition and the use of the mapping is much more straight-forward. The idea in our case 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 SOL. Then, when the current facility $f(j)$ of client j is dispersed, 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 SOL (as shown in Figure 4). However, because of

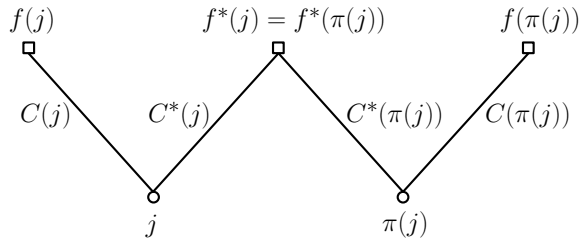


Figure 4: Clients j and $\pi(j)$ are assigned to the same facility in OPT, but to different facilities in SOL (unless $\pi(j) = j$). The marked distances are used in the proof of Lemma 4.7.

our inexact estimate of the facility cost in expression (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 the partner of one of its members. To compensate for the fact that a client j is not necessarily an immediate neighbor of $\pi(j)$ in the tree T , we have a scheme for allocating credit to groups of clients in such a way that on one hand, this credit can be used to pay the extra facility costs at their new facilities, and on the other hand, the total amount of credit given out is no more than the optimal facility cost, F^* . This idea is made precise in Lemma 4.6.

4.3.1 Defining the mapping.

We present a procedure (which is only used for analysis, and is never performed by the local search algorithm) that defines a mapping $\pi : \mathcal{D} \rightarrow \mathcal{D}$ on the clients. Also, for each facility $i \in \mathcal{F}$, it defines a set $H(i)$ of edges and nodes of T . These sets are used later for defining the tree-partition and for distributing credit among groups of clients.

The mapping π maps clients from \mathcal{D} to other clients of \mathcal{D} in a one-to-one and onto fashion. Usually a client j is mapped to a different client $\pi(j)$, but it could also be that $\pi(j) = j$. In either case, it is always true that j and $\pi(j)$ are at the same facility in the optimal solution OPT. Except for the case when $\pi(j) = j$, it is also true that j and $\pi(j)$ are at different facilities in the locally optimal solution SOL. The purpose of the sets $H(i)$ is to partition the facility cost paid by OPT among groups of clients, so as to enable them to pay the additional facility costs at their new facilities to which they are reassigned. The facility cost of OPT is partitioned by including tree nodes in the sets $H(i)$, ensuring that every time a tree node is paid for in OPT, it is placed in at most one set. Then the total cost of nodes placed in sets $H(i)$ does not exceed the facility cost paid by OPT. We summarize these properties below, with addition of two more, which will also be useful for the analysis.

1. $\pi : \mathcal{D} \rightarrow \mathcal{D}$ is 1-1 and onto.
 2. $f^*(j) = f^*(\pi(j))$ for all $j \in \mathcal{D}$. That is, j and $\pi(j)$ are at the same facility in the optimal solution.
 3. For all $j \in \mathcal{D}$, either $j = \pi(j)$ or $f(j) \neq f(\pi(j))$. That is, unless $\pi(j) = j$, j and $\pi(j)$ are assigned to different facilities in SOL.
 4. Each node of the tree T is included in the sets $H(i)$ at most as many times as it is paid for by OPT.
 5. If an edge $(k, \text{parent}(k))$ is included in set $H(i)$, then so is its lower endpoint k .
 6. For j such that $\pi(j) \neq j$, let $\text{lca}(j)$ be the least common ancestor of j and $\pi(j)$ in T . Then the path between j and $\text{lca}(j)$, except for the node $\text{lca}(j)$ itself, is included in the set $H(f(j))$ of j 's facility in SOL.
- For j such that $\pi(j) = j$, the path between j and the root of T , including the root itself, is included in $H(f(j))$.

To define π and $H(i)$, we consider in turn each facility $l \in \mathcal{F}$ used in the optimal solution, and the set $\mathcal{D}^*(l)$ of clients assigned to it by OPT. We assign partners $\pi(j)$ to clients j within this set, thus satisfying property 2. We perform a bottom-up pass through the cost tree of these clients. For each node $k \in T_{\mathcal{D}^*(l)}$ of this tree, we consider a set of clients $S(k) \subseteq \mathcal{D}^*(l)$ that we think of as being “at node k ”, starting with the leaves of the tree and their corresponding clients (the sets $S(k)$ for internal nodes k are defined as the procedure progresses).

For each node k and its set $S(k)$, we partition the clients in $S(k)$ according to their facility in SOL, forming subsets $S(k) \cap \mathcal{D}(i)$ for all facilities i used in SOL (see Figure 5). We aim to assign a partner $\pi(j) \in S(k)$ for all clients $j \in S(k)$ satisfying the condition that j and $\pi(j)$ are assigned to different facilities in SOL. It may not always be possible to find a mapping for all clients in $S(k)$, but we assign as many of them as possible, leaving others to be assigned at higher levels of the tree.

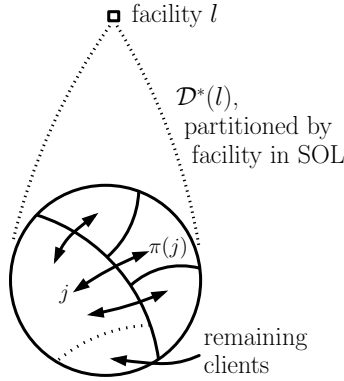


Figure 5: Example of how π is defined on the set $S(k)$ at one node of the tree. If there are extra clients from the majority facility, then they propagate up the tree.

More formally, let i be a *majority* facility, one for which $|S(k) \cap \mathcal{D}(i)|$ is the largest. In the simpler case, at most half of the clients belong to this majority facility, $|S(k) \cap \mathcal{D}(i)| \leq \frac{1}{2}|S(k)|$. Then we can assign $\pi(j) \in S(k)$ for all clients $j \in S(k)$, with $\pi(j) \neq j$, satisfying $f(j) \neq f(\pi(j))$ (condition 3), and none of the clients are propagated up the tree. For example, this assignment can be done by numbering the members of $S(k)$ from 0 to $|S(k)| - 1$ so that clients from the same facilities in SOL form continuous stretches, and then assigning $\pi(j) = j + \lfloor \frac{|S(k)|}{2} \rfloor \pmod{|S(k)|}$, as is done in [1]. Note that such an assignment satisfies condition 1.

In the other case, if the majority facility has more than half the clients, $|S(k) \cap \mathcal{D}(i)| > \frac{1}{2}|S(k)|$, it is impossible to find a mapping π satisfying our condition among the clients at k . So we take the maximum number of clients from the majority facility that can be assigned, $|S(k) \cap \mathcal{D}(i)|$ clients from $\mathcal{D}(i)$, and assign

the mapping π between them and the clients from the other facilities $S(k) \setminus \mathcal{D}(i)$. The remaining clients from the majority facility, which were not assigned, are added to the set $S(\text{parent}(k))$, thus propagating up the tree. Also, in this case we add the node k and the edge $(k, \text{parent}(k))$ to the set $H(i)$ of the majority facility i . This satisfies condition 5. Since node k is paid for at facility l in OPT, and we add it to at most one set $H(i)$ for each such facility l , condition 4 is satisfied as well. If k is the root, then the remaining clients from $\mathcal{D}(i)$ are assigned to themselves, i.e. for them we set $\pi(j) = j$. Also, in this case we add the root of the tree to $H(i)$. Thus, at the end of this process on a facility l , for each client $j \in \mathcal{D}^*(l)$ there is a client $\pi(j) \in \mathcal{D}^*(l)$. See Figure 6 for an example of the process as it proceeds through different levels of the tree.

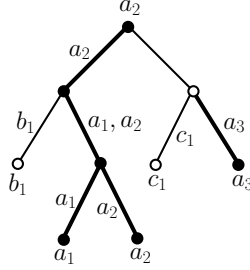


Figure 6: Example of the mapping. The leaves $\{b_1, a_1, a_2, c_1, a_3\}$ represent clients assigned to facility l in OPT. The different letters represent different facilities to which they are assigned in SOL. The labels on edges indicate which clients pass through them as they propagate up the tree. 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 and shaded nodes are included in $H(a)$.

Lemma 4.5 *The mapping π and the sets $H(i)$ defined by the above procedure satisfy properties (1)-(6).*

Proof. We have argued that properties 1-5 are satisfied in the description of the mapping procedure. For the proof of property 6, assume first that $\pi(j) \neq j$, and let k be the node at which $\pi(j)$ was assigned. Then k must be the least common ancestor of j and $\pi(j)$ in T . This is because j and $\pi(j)$ belong to different facilities in SOL, which means that they could not have propagated up the tree to node k from the same child of k (since only clients from the same facility in SOL propagate up any given edge of $T_{\mathcal{D}^*(l)}$), so they must have reached k from different children of k . Now, every node and edge that j crosses as it propagates up the tree from its corresponding leaf to k is included in $H(f(j))$. This is because j only propagates from k' to $\text{parent}(k')$ when $f(j)$ is the majority facility of $S(k')$, which is also when the node k' and the edge $(k', \text{parent}(k'))$ are added to $H(f(j))$. As a result, we get that the whole path from j to k (except k itself) is included in $H(f(j))$. If $\pi(j) = j$, then j must have propagated all the way to the root of T , and $f(j)$ was still the majority facility at the root, which again implies that the whole path, as well as the root, are included in $H(f(j))$. \square

4.3.2 Defining the tree-partition and the facility assignment.

Recall that a disperse move for facility i consists of a tree-partition and a facility assignment for each set of this partition. The set of disperse moves that we use for the analysis consists of one such move, call it (i, \mathcal{S}^i, f') , for each facility i that is used in SOL. We now define \mathcal{S}^i and f' for each i .

To define the tree-partition \mathcal{S}^i for facility i , we use the edges from the set $H(i)$ constructed while defining the mapping π . Recall that edges could have been added to the set $H(i)$ for our facility i which is used in SOL while π was being defined on different facilities l which are used in OPT. In the tree $T_{\mathcal{D}(i)}$ of clients currently at facility i , we retain the edges that are in $H(i)$ and cut all the other edges. This produces a set of connected components of the tree, and therefore defines a tree-partition of the clients $\mathcal{D}(i)$. This is the partition \mathcal{S}^i that we use.

To define the facility assignment $f'(S_h^i)$ for a set $S_h^i \in \mathcal{S}^i$, we make use of the mapping π . Recall that the idea of the mapping π was to find for each client j assigned to a facility $i = f(j)$ in SOL 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_h^i to one facility $f(\pi(j))$ for some $j \in S_h^i$. In particular, among these facilities we choose one which is closest to i . So the set of clients S_h^i in the tree-partition is assigned to the facility $f'(S_h^i) = i'$ from the set $\{f(\pi(j)) : j \in S_h^i\}$ that minimizes $\text{dist}(i, i')$. For example, if $\pi(j) = j$ for any $j \in S_h^i$, then we set $i' = i$.

4.4 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 partitions \mathcal{S}^i defined above are transferred to their assigned facilities, $f'(S_h^i)$.

Lemma 4.6 *The sum of incremental facility costs incurred when each set S_h^i is transferred to its new facility $f'(S_h^i)$ is at most the optimal facility cost,*

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

Proof. For each facility $i \in \mathcal{F}$, and each set $S_h^i \in \mathcal{S}^i$ that arises from a component of the tree-partition of $\mathcal{D}(i)$, we assign a budget B_h^i and show two inequalities:

1. for each $i \in \mathcal{F}$ and $S_h^i \in \mathcal{S}^i$, $\text{cost}_{f'(S_h^i)}(S_h^i) \leq B_h^i$
2. $\sum_{i \in \mathcal{F}} \sum_{S_h^i \in \mathcal{S}^i} B_h^i \leq F^*$

which together imply the lemma.

The way we define B_h^i is as follows. For a component $K(S_h^i)$ of the tree-partition, give it the amount of credit equal to the cost of the nodes which are included both in this component and the set $H(i)$. Formally, $B_h^i = \sum_{k \in K(S_h^i) \cap H(i)} \text{cost}(k)$.

Let us prove inequality 2 first. By property 4 in Section 4.3.1, each node of T is included in sets $H(i)$ at most as many times as it is paid for by OPT. Moreover, notice that each node in $H(i)$ can belong to at most one component of the tree-partition \mathcal{S}^i . Consequently, its cost was added to at most one budget B_h^i , proving the inequality.

To show that inequality 1 holds, consider a set of clients S_h^i and its tree-partition component $K(S_h^i)$. The facility cost for this set of clients, which is the cost of the union of their paths to the root, can be divided into two parts: the cost of the nodes that are in the component $K(S_h^i)$ and the cost of ones which are on the path between the highest node of this component, call it $k(S_h^i)$, and the root of T . We show that the cost of the nodes in the component (except for its highest node) is accounted for in the budget B_h^i , and the cost of the nodes on the path is already paid for at the new facility. The first part follows because the edges of the component are from the set $H(i)$, by property 5, and because the budget B_h^i is allocated for the nodes which are in the set $H(i)$. To show that the path between the component's highest node $k(S_h^i)$ and the root is already paid for at the new facility $f'(S_h^i)$, let $j \in S_h^i$ be the client whose partner's facility was chosen, i.e. such that $f'(S_h^i) = f(\pi(j))$. For the case that $j \neq \pi(j)$, by property 6, the path between j and $\text{lca}(j)$ is in $H(i)$, and therefore also in the component S_h^i . This means that in T , the client $\pi(j)$ is somewhere under the node $k(S_h^i)$, and since the path between $\pi(j)$ and the root of T is paid for at the new facility, so is the path between $k(S_h^i)$ and the root (inclusive). For the case that $j = \pi(j)$, by properties 5 and 6, the whole facility cost of the set S_h^i , including the root, is accounted for in B_h^i . \square

Lemma 4.7 *For the tree-partition $\{S_h^i\}$ and facilities $f'(S_h^i)$ defined above, the connection cost of transferring the sets of clients S_h^i to the facilities $f'(S_h^i)$ can be bounded as*

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

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

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

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

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

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

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

proving the lemma. □

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

Theorem 4.8 *A locally optimal solution SOL 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 disperse move (i, \mathcal{S}^i, f') defined above for a facility i in SOL, then the cost of the solution will change by an amount upper-bounded by expression (2). Because SOL is a locally optimal solution, the value of this move is non-negative:

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

Summing these inequalities over all i and applying Lemmas 4.6 and 4.7, we get that $F^* + 2C + 2C^* - F \geq 0$, or $F \leq F^* + 2C^* + 2C$. Combining this with the bound on C (Lemma 3.3), we get $F \leq 3F^* + 4C^*$, and $F + C \leq 4F^* + 5C^*$. □

Using standard scaling techniques, we improve the bound to $2 + \sqrt{5} < 4.237$. 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 3.3), and $\lambda F \leq 3\lambda F^* + 4C^*$ (see proof of Theorem 4.8). 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 $(4.237 + \epsilon)$ -approximation algorithm.

Theorem 4.9 *There is a polynomial time $(4.237 + \epsilon)$ -approximation algorithm for the facility location problem with hierarchical facility costs that is based on local search and uses aggregate and disperse moves.*

5 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. We define a more general class of such problems by using a facility cost that is an arbitrary monotone submodular function of the set of clients 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, we give a logarithmic-factor approximation algorithm. We also show how to implement the natural *aggregate* move of a local search in polynomial time, as it is a special case of minimizing a submodular function. 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 the general facility location problem with submodular facility costs 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 uncapacitated 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] N. Garg, R. Khandekar, and V. Pandit. Improved approximation for universal facility location. In *Proc. 16th ACM Symp. on Discrete Algorithms*, pages 959–960, 2005.
- [7] S. Guha and S. Khuller. Greedy strikes back: Improved facility location algorithms. *J. Algorithms*, 31(1):228–248, 1999.
- [8] M. T. Hajiaghayi, M. Mahdian, and V. S. Mirrokni. The facility location problem with general cost functions. *Networks*, 42:42–47, 2003.
- [9] A. Hayrapetyan, C. Swamy, and E. Tardos. Network design for information networks. In *Proc. 16th ACM Symp. on Discrete Algorithms*, pages 933–942, 2005.
- [10] D. S. Hochbaum. Heuristics for the fixed cost median problem. *Math. Programming*, 22(1):148–162, 1982.
- [11] S. Iwata, L. Fleischer, and S. Fujishige. A combinatorial strongly polynomial algorithm for minimizing submodular functions. *J. ACM*, 48(4):761–777, 2001.
- [12] 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.
- [13] 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.
- [14] M. Mahdian and M. Pál. Universal facility location. In *European Symposium on Algorithms*, pages 409–421, 2003.
- [15] M. Mahdian, Y. Ye, and J. Zhang. Improved approximation algorithms for metric facility location problems. In *Proc. 5th APPROX*, pages 229–242, 2002.
- [16] R. Ravi and A. Sinha. Multicommodity facility location. In *Proc. 15th ACM Symp. on Discrete Algorithms*, pages 342–349, 2004.
- [17] A. Schrijver. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *J. of Combinatorial Theory, Ser. B*, 80(2):346–355, 2000.
- [18] 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.
- [19] 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.

- [20] M. Sviridenko. An improved approximation algorithm for the metric uncapacitated facility location problem. In *IPCO*, pages 240–257, 2002.
- [21] V. V. Vazirani. *Approximation algorithms*. Springer-Verlag New York, Inc., 2001.
- [22] 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.