Universal Facility Location

Mohammad Mahdian [*] Martin Pál [†]

(Both authors are students.)

Abstract

In the Universal Facility Location problem we are given a set of demand points and a set of facilities. The goal is to assign the demands to facilities in such a way that the sum of service and facility costs is minimized. The service cost is proportional to the distance each unit of demand has to travel to its assigned facility, whereas the facility cost of each facility i depends on the amount of demand assigned to that facility and is given by a cost function $f_i(\cdot)$.

We present a $(7.88 + \varepsilon)$-approximation algorithm for the Universal Facility Location problem based on local search, under the assumption that the cost functions f_i are nondecreasing. The algorithm chooses local improvement steps by solving a knapsack-like subproblem using dynamic programming. This is the first constant-factor approximation algorithm for this problem. Our algorithm also slightly improves the best known approximation ratio for the capacitated facility location problem with non-uniform hard capacities.

[*]Laboratory for Computer Science, MIT, Cambridge, MA 02139. Email: mahdian@theory.lcs.mit.edu.

[†]Computer Science Department, Cornell University, Ithaca, NY 14853. Email: mpal@cs.cornell.edu. Research supported in part by ONR grant N00014-98-1-0589.

1 Introduction

In the facility location problem, we are given a set of demands and a set of possible locations for facilities. The objective is to open facilities at some of these locations and connect each demand point to an open facility in such a way that the total cost of opening facilities and connecting demand points to open facilities is minimized.

Many variants of the problem have been studied, such as uncapacitated facility location, in which we pay a fixed cost for opening each facility, and an open facility can serve any number of clients, hard-capacitated facility location, in which each facility has a upper bound on the amount of demand it can serve, or soft-capacitated facility location (each facility has a capacity, but we are allowed to open multiple copies of each facility). Facility location problems have occupied a central place in operations research since the early 60's [2, 12, 14, 20, 17, 5, 7]. Many of these problems have been studied extensively from the perspective of approximation algorithms.

Linear Programming (LP) based techniques have been applied successfully to uncapacitated and soft-capacitated variants of the facility location problem to obtain constant factor approximation algorithms for these problems (See [19] for a survey and [15, 16] for the latest results). However, LP based techniques have not been successful when dealing with hard capacities, and all known LP formulations have an unbounded integrality gap.

Using local search, Korupolu, Plaxton and Rajamaran [13] and Chudak and Williamson [6] gave a constant approximation algorithm for the hard capacitated problem, under the assumption that all capacities are the same. Pál, Tardos and Wexler [18] gave a local search algorithm for facility location with arbitrary hard capacities that achieves approximation ratio $9+\varepsilon$. Local search heuristics have been successfully used for other variants of facility location [14, 13, 6, 4, 1].

In this paper we consider a generalized version of the facility location problem in which the cost of opening each facility is an arbitrary given function of the amount of demand served by it. This problem, which we call the *universal facility location problem*, was defined and studied for the case of concave facility cost functions in [10]. Great number of well-studied problems such as uncapacitated facility location, capacitated facility location with soft and hard capacities, facility location with incremental facility costs (a.k.a. the linear-cost facility location problem), the concave-cost facility location are special cases of the universal facility location problem. In this paper we present the first constant factor approximation algorithm for the universal facility location problem with non-decreasing facility costs.

Our algorithm is a natural generalization of the local search algorithm of Pál et al. [18], and achieves an approximation factor of $8+\varepsilon$. This slightly improves the best approximation factor known for the hard-capacitated facility location problem. Furthermore, since our algorithm employs local transformations that are more powerful than the operation of Charikar and Guha [4] for uncapacitated facility location, as well as the transformations of Arya et al. [1] for soft-capacitated facility location, our algorithm inherits approximation guarantees of both algorithms in these special cases. In other words, our algorithm provides a common generalization of several local search algorithms that have been proposed for various facility location problems.

The rest of this paper is organized as follows. In Section 2 we define the universal facility location problem, introduce some notation and mention some important special cases of universal facility location. In Section 3, we describe the overall structure of the algorithm, define the local operations and show how to find them efficiently. In Section 4, the analysis, we prove the approximation guarantee of the algorithm. Section 5 contains some concluding remarks.

2 Definitions

In this section, we first introduce the universal facility location problem, and then define several well-studied facility location problems as special cases of the universal facility location problem.

2.1 The Universal Facility Location Problem

The *Universal Facility Location* (UniFL) Problem is defined as follows. We are given a set D of *demand points* (a.k.a. *clients*), a set F of *facilities*, and a weighted graph G on the vertex set $F \cup D$ with connection costs c on the edges. Each client $j \in D$ has d_j units of demand to be served by the facilities. The goal is to allocate certain capacity u_i at every facility $i \in F$ and assign all demands to the facilities subject to the constraint that each facility i can serve no more demand than the capacity u_i that has been allocated at it. We allow for splittable demands, that is, the demand of a client can be split among multiple facilities.

The cost $c(S)$ of a solution S to a UniFL instance is the sum of facility and service costs, denoted by $c_f(S)$ and $c_s(S)$, respectively. There is a cost associated with allocating capacity at each facility, depending on the amount allocated. Let us denote the cost of allocating u_i units of capacity at facility i by $f_i(u_i)$. We refer to the sum of costs of allocating capacity at all facilities as the *facility cost*. There is also a cost c_{ij} for shipping each unit of demand from client j to facility i. We assume that these connection costs form a metric, i.e., they are symmetric and obey the triangle inequality. The total shipping cost is referred to as the *service cost*.

A solution S to a UniFL instance can be specified by a pair (u, x), where u is the *allocation vector* (i.e., for $i \in F$, u_i is the capacity allocated at facility i), and x is the *assignment vector* (i.e., x_{ij} denotes the amount of demand of client j served by facility i). Using this notation, we can write the universal facility location problem as the following nonlinear optimization problem.

$$
\begin{array}{lrcll}
\text{minimize} & \multicolumn{3}{l}{\displaystyle\sum_{i \in F} f_i(u_i) + \sum_{i \in F, j \in D} c_{ij} x_{ij}} & \\[2ex]
\text{subject to} & \displaystyle\sum_{i \in F} x_{ij} & = & d_j & \forall j \in D \\
& \displaystyle\sum_{j \in D} x_{ij} & \leq & u_i & \forall i \in F \\
& u_i, x_{ij} & \geq & 0 & \forall i \in F,\ j \in D.
\end{array}
$$

Note that it makes sense to assume that the functions $f_i(\cdot)$ are non-decreasing (otherwise we can always allocate more capacity if it costs us less).[1] To model hard upper bounds on capacities, we allow f_i to take the value $+\infty$. We make the following assumptions about the functions f_i.

Assumption 2.1 *We assume that each f_i is a non-decreasing and left-continuous mapping from non-negative reals to non-negative reals with infinity. That is, $f_i : \mathbb{R}^{\geq 0} \to \mathbb{R}^{\geq 0} \cup \{+\infty\}$, $f_i(x) \leq f_i(y)$ for every $x \leq y$, and $\lim_{x \to u^-} f_i(x) = f_i(u)$ for every u.*

The above assumption is not only helpful in designing an approximation algorithm, it also guarantees the existence of a globally optimal solution.

Theorem 2.2 *Any instance of UniFL with the cost functions satisfying Assumption 2.1 has an optimal solution.*

[1] In order to define the problem for facility cost functions that are not necessarily non-decreasing, we need to change the second constraint of the optimization program to equality. There are some cases where having a decreasing cost function is useful. See Section 5 for an example.

Proof. If no solution has finite cost, any solution is optimal. Therefore, the infimum C^* of the set of costs of solutions to the UniFL problem is a well-defined finite number. There must be a sequence of solutions $\{(u^k, x^k)\}_{k=1}^{\infty}$ such that $\lim_{k\to\infty} \mathrm{cost}(u^k, x^k) = C^*$ and u^k and x^k are bounded. Since C^* is finite, $f_i(u_i^k)$ must be bounded for all $k > k_0$ as well. Hence we can pick an infinite subsequence $\{(u^{k_l}, x^{k_l})\}_{l=1}^{\infty}$ such that the limits $u^* = \lim_{l\to\infty} u^{k_l}$, $x^* = \lim_{l\to\infty} x^{k_l}$ and $f_i^* = \lim_{l\to\infty} f_i(u_i^{k_l})$ for $i \in F$ exist. Since f_i are non-decreasing and left continuous by assumption, $f_i(\lim_{l\to\infty} u_i^{k_l}) \leq f_i^* = \lim_{l\to\infty} f_i(u_i^{k_l})$, and hence (u^*, x^*) must be an optimal solution. ■

In addition to the Assumption 2.1, it usually makes sense to assume that for every i, $f_i(0) = 0$, that is, we do not pay for facility we do not use. However, we do not need this assumption in our analysis.

2.2 Special Cases

The Universal facility location generalizes several important variants of the facility location problem that have been studied in the literature.

The *uncapacitated facility location* problem is a special case of UniFL problem where cost functions are of the form $f_i(u) = \bar{f}_i \cdot [u > 0]$. That is, every facility serving positive demand must pay the opening cost $\bar{f}_i$, and any open facility can serve an unlimited amount of demand. The best approximation algorithm known for this problem is LP-based and achieves an approximation ratio of 1.52 [15]. The best local search algorithm for this problem is due to Charikar and Guha [4], and achieves a factor of $3 + \varepsilon$ ($1 + \sqrt{2} + \varepsilon$ with scaling). It is easy to observe that the local steps we use in our algorithm generalize the local steps of Charikar and Guha [4], and hence their analysis also shows that our algorithm achieves the same approximation guarantees for the uncapacitated facility location problem.

The *capacitated facility location problem with soft capacities* has cost functions of the form $f_i(u) = \bar{f}_i \cdot \lceil u/\bar{u}_i \rceil$. In words, at each site i we can open an arbitrary number of "copies" of a facility at cost $\bar{f}_i$ each. Each copy can serve up to $\bar{u}_i$ units of demand. This problem is also known as the capacitated facility location problem with integer decision variables in the operations research literature [3]. The best approximation algorithm known for this problem is LP-based and achieves a factor of 2 [16]. Arya et al. [1] give a local search algorithm with an approximation guarantee of $4 + \varepsilon$ for this problem. Since our local steps are at least as powerful as theirs, their approximation guarantee carries over to our algorithm in the soft-capacitated case.

The *facility location problem with incremental costs* (a.k.a. the *linear-cost facility location problem*) is a special case of the UniFL problem in which the facility costs are of the form $f_i(u) = \bar{f}_i \cdot [u > 0] + \sigma_i \cdot u$, i.e., in addition to opening costs $\bar{f}_i$, there is also cost σ_i per unit of demand served. This can be easily reduced to the uncapacitated problem by increasing all c_{ij} distances by σ_i.

In the *concave-cost facility location problem*, cost functions $f_i(\cdot)$ are arbitrary concave functions. This problem has been studied in the operations research literature [7]. A concave function can be well approximated by a collection of linear functions from its upper linear envelope. This suggests that UniFL with arbitrary concave cost functions can be reduced to facility location with incremental costs, and further to uncapacitated facility location (See [10] for details). Without much effort we can show that our algorithm performs these reductions implicitly, hence the $3 + \epsilon$ guarantee of [4] carries over to instances with concave facility cost as well.

In the *capacitated facility location problem with hard capacities* each facility has an opening cost $\bar{f}_i$ and a hard capacity $\bar{u}_i$. An open facility can serve up to $\bar{u}_i$ demand, and this cannot be exceeded at any cost. Hence the cost function is $f_i(u) = \bar{f}_i \cdot [u > 0] + \infty \cdot [u > \bar{u}_i]$. Hard capacitated facility location is perhaps the most difficult special case of UniFL, in that it captures much of the hardness of UniFL with respect to approximation. The only known approximation algorithm for this problem is due to Pál, Tardos, and Wexler [18], and achieves an approximation factor of $8.53 + \varepsilon$. Our algorithm is an extension of this algorithm.

3 The algorithm

The basic scheme of the algorithm is very simple. Start with an arbitrary feasible solution and repeatedly look for local transformations that decrease the cost. If no such operation can be found, output the current solution and stop. Otherwise, pick the operation that decreases the cost by the greatest amount, apply it to the current solution and continue with the modified solution.

This simple scheme guarantees that (if it stops), the algorithm arrives at a *locally optimal* solution, i.e., a solution that cannot be improved by any local transformation. In Section 4 we will show that any locally optimal solution with respect to our operations is a good approximation for the (globally) optimal solution. To obtain polynomial running time, we need to address two issues with this scheme.

Significant improvements. We shall require that every local transformation we apply improves the cost $c(S)$ of the current solution S significantly, that is, by at least $\frac{\epsilon}{5n}c(S)$ for some $\epsilon > 0$. We call such an operation *admissible*. If no admissible operation can be found, the algorithm stops. Note that since the optimum cost $c(S^*)$ is a lower bound on the cost of any solution, the algorithm stops after at most $\frac{5n}{\epsilon} \ln \frac{c(S)}{c(S^*)}$ iterations. This means that the solution we output may be only approximately locally optimal, not a true local optimum. The bound on the cost of solutions that are only approximately optimal is only by an ϵ factor worse than the bound for true local optima.

Efficient operation finding. We need to show that in each iteration we can find an admissible operation in polynomial time, if one exists. We do not know how to find the best local operations efficiently, because even for very simple functions f_i (such as the functions with two steps arising from capacitated facility location), finding an optimal local step is NP-hard. However, we are able to find a local operation with cost within a small additive factor, say $\frac{\varepsilon}{10n}c(S)$ of the best operation in polynomial time by solving a knapsack-like subproblem using dynamic programming. This guarantees that if an admissible operation exists, our algorithm finds an operation that improves the cost by at least $\frac{\epsilon}{10n}c(S)$. It is still enough to drive the cost down sufficiently fast.
Our algorithm employs the following two types of local transformations.

- $\mathsf{add}(s, \delta)$. Increase the allocated capacity u_s of facility s by δ, and find the minimum cost assignment of demands to facilities, given their allocated capacities. A variant of this operation has been considered by many previous papers [1, 13, 6, 18]. This operation allows us to bound the service cost (Lemma 4.1).

  The cost of this operation is $f_s(u_s+\delta_s)-f_s(u_s)+c_s(S')-c_s(S)$, where $c_s(S)$ and $c_s(S')$ indicate the service cost of the solution before and after the operation, respectively. These costs can be computed by solving a minimum cost network flow problem.

- $\mathsf{pivot}(s, \Delta)$. This operation is a combination of the open and close operations of [18]. It is strictly more powerful, however, as it allows us to prove a somewhat stronger approximation guarantee of $8 + \varepsilon$.

  In the operation $\mathsf{pivot}(s, \Delta)$, we adjust the amount of demand served by each facility i by Δ_i. For a facility i with $\Delta_i < 0$ this means that we must ship $|\Delta_i|$ units of excess demand out of i. We gather all this excess demand at a location s, which is the *pivot* for the operation. Subsequently we distribute the demand gathered at s to facilities with $\Delta_i > 0$. Finally, we adjust the allocated capacity of each facility to be equal to the actual amount of demand served by the facility. Note that since the amount of demand in the system must be conserved, the operation is feasible only if $\sum_i \Delta_i = 0$.

Assuming that before the operation the allocated capacity u_i of each facility i was equal to the actual demand served by the facility, the cost of the $\mathsf{pivot}(s, \Delta)$ operation can be written as $\sum_{i \in F} f_i(u_i + \Delta_i) - f_i(u_i) + c_{si} \cdot |\Delta_i|$.

Recall that an admissible local step is defined as a local step with cost at most $-\frac{\epsilon}{5n} c(S)$, where $c(S)$ is the cost of the solution before performing the operation. To build a local search algorithm on these operations, we must be able to find admissible local steps efficiently. In the remainder of this section, we first show how to find optimal operations under the following *discretization assumption*: we shall assume that all demands d_j are "small" integers and that the cost functions are step functions with steps occurring only at integer points (i.e. $f_i(u) = f_i(\lceil u \rceil)$ for any real u). At the end of the section we show how to remove this assumption.

Finding the best $\mathsf{add}(s, \delta)$ operation under this assumption is straightforward: there are only $n = |F|$ possibilities for s and at most $|d| = \sum_j d_j$ possibilities for δ, so we can try them all.

Finding a good $\mathsf{pivot}(s, \Delta)$ operation is more involved; the vector Δ has n coordinates, each of them ranging between $-|d|$ and $|d|$, hence we cannot hope to try all possibilities in polynomial time. Instead, we give a procedure that finds an optimal vector Δ for a given pivot point s. This problem is essentially a covering knapsack problem.

Lemma 3.1 *Given a solution (u, x) and a point $s \in G$, we can find an operation* $\mathsf{pivot}(s, \Delta)$ *with minimum cost in time polynomial in the number of facilities n and the total demand $|d|$.*

Proof. Consider the current solution (u, x) and fix a pivot $s \in G$. First, for every i, adjust u_i so that it is equal to the actual amount of demand served by the facility i. Let $g_i(\delta)$ denote the cost of shipping $\delta \geq 0$ extra units of demand from the pivot *to* the facility i, and adjusting the capacity of i accordingly. The function g_i can be expressed as $g_i(\delta) = c_{si} \cdot \delta + f_i(u_i + \delta) - f_i(u_i)$. Similarly, let $g_i(-\delta)$ denote the cost of moving δ units of demand *from* facility i to the pivot s (this makes sense only if $\delta < u_i$). Analogously, $g_i(-\delta) = c_{si} \cdot \delta + f_i(u_i - \delta) - f_i(u_i)$.

We use dynamic programming to build a two-dimensional table a, where $a(i, w)$ is the minimum cost of having w units of excess (or deficiency) of demand at the pivot s, after considering the facilities $1, \ldots, i$. We can easily derive the recurrence

$$a(i, w) = \begin{cases} g_1(-w) & i = 1 \\ \min_\delta \{g_i(\delta) + a(i-1, w+\delta)\} & i = 2, \ldots, n \end{cases}$$

for $w = -|d|, \ldots, |d|$. The best pivot operation corresponds to the entry $a(n, 0)$, that is, the solution that leaves no excess or deficit demand at s. ■

We can do a little better when computing the cost of reassigning demand from a facility i to the pivot s. Since each unit of demand at i originated from some client j, instead of shipping the demand from i to s at cost c_{is}, we can reroute it directly from j to s at cost $c_{js} - c_{ji}$ (if this quantity is negative, we in fact save on shipping). The functions g_i can be redefined to reflect this saving, reassigning demand greedily by ordering clients by increasing $c_{js} - c_{is}$. In our analysis, it is enough to assume the original definition of g_i. However, to be able to claim that our algorithm generalizes the algorithms of [4, 1], we need to use the refined cost estimate.

Finally we return to the question of removing the discretization assumption. It is not possible to simply round all demands to the nearest integer and replace each f_i by a step function, as this may distort the cost of a solution considerably. Instead, we discretize costs, at the expense of an additional ε factor in the approximation ratio.

Let B_i denote the budget of facility i, i.e. $B_i = f_i(u_i)$. For the add operation, we try to increase the budget of each facility i in multiples of $\frac{\varepsilon}{n}c(S)$, up to a maximum $c(S)$. To find a suitable pivot operation, we apply dynamic programming over costs instead of capacities. We build a table p' that is an "approximate inverse" of the table p: the entry $p'(i, C)$ is the minimum excess (or maximum deficiency) of demand at the pivot s that can be achieved at a cost of at most C. The table is computed for values of C between $-c(S)$ and $c(S)$ in increments of a suitable fraction of $\frac{\varepsilon}{n^2}c(S)$.

Lemma 3.2 *An* add *or* pivot *operation with cost within an additive factor of $\frac{\varepsilon}{8n}c(S)$ of the best* add *or* pivot *operation can be found in polynomial time.*

4 The analysis

The goal of this section is to show that if for some solution $S = (u, x)$ our algorithm cannot find an operation that would significantly improve its cost, then the cost of S is not too large in comparison to the optimal solution S^*. Our analysis roughly follows the analysis in [18].

For the most of this section we shall assume that S in fact is a locally optimal solution, i.e. no operation can improve its cost. We show that the cost of any locally optimal solution S is within a factor of 8 of the cost of the optimal solution S^*. In the end of the section we extend this argument to yield a bound of $8 + \varepsilon$ for solutions for which no significant improvement can be found.

4.1 Bounding the service cost

Any local search algorithm that has the add operation in its repertoire can guarantee low service cost. A variant of the following lemma has been first proved in [13] and [8], and in various versions it has become folklore of the field since.

Lemma 4.1 *The service cost $c_s(S)$ of a locally optimal solution $S = (u, x)$ is at most the cost of the optimal solution $c(S^*)$.*

Proof. We only sketch the idea. Suppose that $c_s(S) > c_f(S^*) + c_s(S^*)$, where $S^* = (u^*, x^*)$ is an optimal solution. Consider a fictitious operation in which we were to increase the allocated capacity of each facility i to $\max(u_i, u_i^*)$, paying at most $c_f(S^*)$ on allocation costs. With the new allocations, the assignment vector x^* becomes feasible, and hence the new assignment cost is at most $c_s(S^*)$. Thus the cost of the solution is improved by $c_s(S) - c_s(S^*) - c_f(S^*) > 0$ by this fictitious operation. Using submodularity of assignment costs we can argue that since increasing the capacities of all facilities simultaneously improves the cost, there must be a facility i such that increasing its capacity to u_i^* improves the cost of S, contradicting the local optimality of S. ∎

4.2 Bounding the facility cost

The bulk of work we have to do in the analysis goes to proving a bound on the facility cost. To do this, we start with a solution S that is locally optimal with respect to the add operation, and hence has small service cost. We argue that if the facility cost of S is large, then there must be a pivot operation that improves the cost of S. In order to illustrate the technique, we start by imagining that instead of the pivot operation, we have a stronger global operation swap, as defined below.

Definition. Let u denote the capacities allocated in the current solution. The operation swap$(u^* - u)$ adjusts the capacity of each facility i from u_i to u_i^* and reroutes excess demand from facilities with $u_i > u_i^*$ to facilities with $u_i^* > u_i$ which after adjustment have excess free capacity. The cost of this operation is equal to the total facility cost at capacities u^* ($c_f(S^*)$) minus the total facility cost at capacities u ($c_f(S)$) plus the *rerouting cost* $\sum_{i,j} c_{ij}\delta_{ij}$, where δ_{ij} denotes the amount of flow rerouted from facility i to facility j.

The plan for the rest of this section is as follows: We first show how to reroute the demand in the operation $\mathsf{swap}(u^* - u)$ so that the rerouting cost is small. Then we show how to replace this fictitious swap operation with a list of pivot operations and bound the total cost of these operations in terms of the global swap operation. Finally, if S is locally optimal, we know that the cost of each pivot operation must be greater than or equal to zero. Summing these inequalities over all pivot operations in our list gives us a bound on the facility cost of S.

The exchange graph. Without loss of generality we shall assume that the allocated capacity at each facility is equal to the demand served in both S and S^*: $u_i = \sum_j x_{ij}$ and $u_i^* = \sum_j x_{ij}^*$. Let $\delta_i = u_i - u_i^*$ be the amount of excess or deficit of demand of each facility in our fictitious swap operation; note that $\sum_i \delta_i = 0$. We have not yet specified how we reassign demand. To do this, we set up a transshipment problem on the graph G: facilities $U = \{i \in F \mid \delta_i > 0\}$ are the sources, and facilities $U^* = \{i \in F \mid \delta_i < 0\}$ are the sinks. The goal is to find a flow of small cost such that exactly δ_s units of flow emanate from every source s and $-\delta_t$ units enter into each sink t. The cost of shipping a unit of flow between two vertices s and t is equal to their distance c_{st}, and there are no capacities on the edges.

Note that each flow that is feasible for the transshipment problem immediately gives a way of reassigning demand in the imaginary swap operation, and the cost of the flow is equal to the rerouting cost of the swap operation. We claim that there is a solution to the transshipment problem with low cost.

Lemma 4.2 *The transshipment problem defined above has a solution of cost at most $c_s(S) + c_s(S^*)$.*

Proof. Take the assignment vectors x and x^* and interpret them as flows, sending x_{ij} and x_{ij}^* units of flow respectively from each client j to each facility i. Note that $x^* - x$ is then also a flow (sending x_{ij} units of flow from j to i means the same as sending $-x_{ij}$ units from i to j). This flow satisfies flow conservation at every client (amount of demand d_j flowing out is equal to the amount d_j flowing in), and also carries the correct amounts δ_i out of sources and into sinks. The cost of this flow is $\sum_{i,j} x_{ij}c_{ij} + \sum_{i,j} x_{ij}^* c_{ij} = c_s(S) + c_s(S^*)$. The optimal flow can only be cheaper. ■

We do not know how to find a good swap operation efficiently. Therefore, we cannot design an efficient algorithm based on this operation. However, we would like to illustrate the style of argument we will be using with the following simple lemma.

Lemma 4.3 *Let S be a solution that can not be improved by any* swap *or* add *operation. Then $c_f(S) \leq 2c(S^*)$.*

Proof. The facility cost of the $\mathsf{swap}(u^* - u)$ operation is $\sum_{i \in F} f_i(u_i^*) - f_i(u_i) = c_f(S^*) - c_f(S)$, while the rerouting cost by Lemma 4.2 is at most $c_s(S) + c_s(S^*)$. Since S is locally optimal with respect to swaps, the cost of the swap must be nonnegative:

$$0 \leq -c_f(S) + c_f(S^*) + c_s(S^*) + c_s(S). \quad (1)$$

By Lemma 4.1, we have $c_s(S) \leq c(S^*)$. Plugging this into the above inequality and rearranging we get the claimed bound. ■

The exchange forest. Let us consider a flow y that is an optimal solution to the transshipment problem defined above. We claim that without loss of generality, the set of edges with nonzero flow forms a forest. Indeed, suppose that there is a cycle of nonzero edges. Augmenting the flow along the cycle in either direction must not increase the cost of the flow (otherwise augmenting in the opposite direction would decrease it, contradicting the optimality of y). Hence we can keep augmenting until the flow on one of the edges becomes zero; this way we can remove all cycles one by one. Similarly we can assume that there is no flow between a pair of sources (facilities with $\delta_i > 0$) or sinks (facilities with $\delta_i < 0$). If there was a positive flow from a source s_1 to another source s_2, it must be the case that it eventually arrives at some sink (or sinks) t. By triangle inequality, the flow can be sent directly from s_1 to t without increasing the cost.

Hence, the flow y can be thought of as a collection of bipartite trees, with edges leading between alternating layers of vertices from U^* and U. We root each tree at an arbitrary facility in U^*.

A list of pivot **operations.** We are ready to proceed with our plan and decompose the imaginary swap operation into a list of local pivot operations. Each of these pivot operations decreases the capacity of some facilities from u_i to u_i^* (we say that such a facility is *closed*) and increases some others from u_i to at most u_i^* (we say that such a facility is *opened*). Note that only facilities in U^* can be opened, and only facilities in U can be closed by the operations in the decomposition. Also, for each pivot operation, we specify a way to reroute the demand from the facilities that are closed to facilities that are open *along the edges of the exchange forest*.

Therefore, since f_i's are non-decreasing, the cost of a pivot operation in our list is at most

$$\sum_{i \in A} (f_i(u_i^*) - f_i(u_i)) + \sum_{e} c_e \delta_e, \tag{2}$$

where A is the set of facilities affected (i.e., either opened or closed) by the operation, and δ_e is the amount of demand rerouted through edge e of the exchange forest.

To get an upper bound on the amount $\sum_{s \in U} f_s(u_s) - f_s(u_s^*)$ by which the facility cost of S exceeds the facility cost of S^*, we choose the list of operations in such a way that they satisfy the following three properties.

1. Each facility in U is closed by exactly one operation.
2. Each facility in U^* is opened by at most 4 operations.
3. For every edge e of the exchange graph, the total amount of demand rerouted through e is at most 3 times the amount of flow y along e.

The following lemma shows that finding a list of pivot operations with the above properties is enough for bounding the facility cost of S.

Lemma 4.4 *Assume there is a list of* pivot *operations satisfying the above properties. Then, the facility cost of the solution S is at most $c_f(S) \leq 4 \cdot c_f(S^*) + 3 \cdot (c_s(S) + c_s(S^*))$.*

Proof. Since S is a local optimum, the cost of every local operation, and therefore the upper bound given in (2) for the cost of each operation in our list must be non-negative. Adding up these inequalities, and using the above properties and the definition of U and U^*, we get

$$4 \sum_{i \in U^*} (f_i(u_i^*) - f_i(u_i)) + 3 \sum_{s,t} c_{st} y_{st} \geq \sum_{i \in U} (f_i(u_i) - f_i(u_i^*)),$$

where $\sum_{s,t} c_{st} y_{st}$ is the cost of flow y, which is at most $c_s(S) + c_s(S^*)$ by Lemma 4.2. Adding $\sum_{i \in U} f_i(u_i^*) + \sum_{i \notin U} f_i(u_i)$ to both sides and rearranging, we get

$$4c_f(S^*) + 3(c_s(S) + c_s(S^*)) \geq \sum_{i \in F} f_i(u_i^*) + \sum_{s,t} c_{st} y_{st} \geq \sum_{i \in F} f_i(u_i) = c_f(S).$$

■

In the rest of this section, we will present a list of operations satisfying the above properties.

Decomposing the trees. Recall that the edges on which the flow y is nonzero form a forest. Root each tree $\mathcal{T}$ at some facility $r \in U^*$. For a vertex $t \in U^*$, define $C(t)$ to be the set of children of t. Since the flow y is bipartite, the children $C(t) \subseteq U$. For a $t \in U^*$ that is not a leaf of $\mathcal{T}$, let $\mathcal{T}_t$ be the subtree of depth at most 2 rooted at t containing all children and grandchildren of t. (See Figure 1).

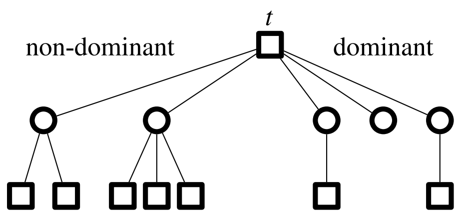

Figure 1: The subtree $\mathcal{T}_t$. Circles denote facilities from U, squares facilities from U^*.

We define a set of operations for every such subtree $\mathcal{T}_t$. In case t has no grandchildren, we consider a single pivot(t, Δ) operation that has t as its pivot, opens t, closes the children $C(t)$, and reroutes the traffic from facilities in $C(t)$ to t through the edges of the exchange graph. This operation is feasible, as the total capacity closed is less than or equal to the capacity opened (i.e., $u_t^* - u_t$). Also, in this case the reassignment cost along each edge in $\mathcal{T}_t$ is bounded by the cost of the flow on that edge.

We now consider the general case in which the tree $\mathcal{T}_t$ has depth 2. We divide the children of t into two sets. A node $s \in C(t)$ is *dominant*, if at least half of the total flow emanating from s goes to t (i.e. $y_{st} \geq \frac{1}{2} \sum_{t' \in U} y_{st'}$) and *non-dominant* otherwise. Let Dom and $NDom$ be the set of dominant and non-dominant facilities respectively. (See Figure 1).

We close all dominant nodes in a single operation. We let t to be the pivot point, close all facilities in Dom and open t and all children of Dom. Note that since we decided to open all neighbors of Dom, there will be enough free capacity to accommodate the excess demand. Figure 2 (left panel) shows how the demand is rerouted.

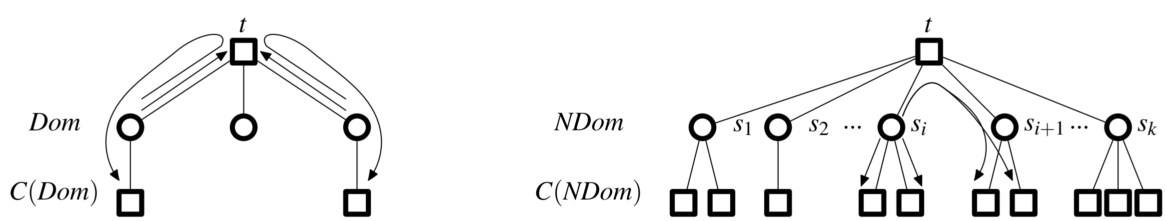

Figure 2: Reassigning demand. The operation pivot(t, Δ) shown on the left closes all facilities in Dom. The pivot operation on the right closes a facility $s_i \in Dom$.

We can not afford to do the same with the non-dominant nodes, because the pivot operation requires that all the affected demands are first shipped to the pivot. Since the flow from a node $s \in C(t)$ to its children may be arbitrarily larger than the flow from s to t, the cost of shipping it to t and then back to the children of s might be prohibitively large. Instead, we use a separate pivot operation for each $s \in C(t)$ that closes s and opens all children of s. The operation may not be feasible, because we still may have to deal with the leftover demand that s was sending to t. The opening cost of t may be large, hence we need to avoid opening t in a pivot operation for every $s \in NDom$.

We order the elements $s_1, s_2, \ldots, s_k$ of $NDom$ by the amount of flow they send to the root t, i.e., $y_{s_1,t} \leq y_{s_2,t} \leq \ldots \leq y_{s_k,t}$. For s_i, $1 \leq i < k$, we consider the operation with s_i as the pivot, that closes s_i and opens the set $C(s_i) \cup C(s_{i+1})$. By our ordering, the amount of flow from s_i to t is no more than the amount of flow from s_{i+1} to t, and since s_{i+1} is non-dominant, this is also less than or equal to the total capacity opened in $C(s_{i+1})$. Hence the opened capacity at $C(s_i) \cup C(s_{i+1})$ is enough to cover the excess demand arising from closing capacity at s_i. For the last facility s_k we consider the operation with s_k as a pivot, that closes s_k and opens facility t as well as facilities in $C(s_k)$. (See Figure 2).

We have defined a set of pivoting operations for each tree $\mathcal{T}_t$ for $t \in U^*$. To finish the analysis, we need to show that these operations satisfy the properties 1–3 above.

Lemma 4.5 *In the set of operations defined above, each facility $s \in U$ is closed once, and each facility $t \in U^*$ is opened at most 4 times.*

Proof. Every facility $s \in U$ is a child in some subtree $\mathcal{T}_t$, and the operations defined for each subtree ensure that each child is closed once. Each facility $t \in U^*$ is the root of a subtree $\mathcal{T}_t$ at most once, and can also be a leaf in at most one subtree. As a root, each facility is opened at most twice: once for the dominant and once for the non-dominant facilities. As a leaf, t can be opened twice if it is a child of a non-dominant facility or once if its parent is dominant. ∎

Lemma 4.6 *For every edge e of the exchange graph, the amount of demand that is being rerouted by the operations on e does not exceed three times the amount of flow y on e.*

Proof. Each edge e belongs to exactly one subtree $\mathcal{T}_t$. Consider all operations in $\mathcal{T}_t$. Consider the edge (s,t) between a dominant node s and the root t. The pivot operation that closes s routes $y_s := \sum_{t' \in U^*} y_{s,t'}$ demand from s to the pivot point t, leaves $y_{s,t}$ demand at t and then ships $y_s - y_{s,t}$ demand back to the children of s. Hence the total amount of demand shipped back and forth across the edge (s,t) is $2y_s - y_{st}$. Because s is dominant (i.e., $y_s \leq 2y_{st}$), this is at most $3y_{st}$.

Now consider the edges between t and the facilities $NDom = \{s_1, s_2, \ldots, s_k\}$, ordered by $y_{s_i,t}$. Each edge (s_i, t) is used by the operation with pivot s_i. This operation reroutes $y_{s_i,t}$ demand from s_i either directly to t (if $i = k$), or via t to the children of s_{i+1}. For $i > 1$, the edge can also be used by the pivot operation on s_{i-1} to route $y_{s_{i-1},t}$ units of demand from s_{i-1} via t and s_i to the children of s_i. By our ordering, this is no more than the flow $y_{s_i,t}$ on the edge. Since the edge is used by at most two operations, the bound holds in this case as well. The edges between $C(t)$ and the leaves can be checked similarly. ∎

Lemmas 4.1, 4.4, 4.5, and 4.6 imply the following.

Theorem 4.7 *Consider a UniFL instance. Let S^* be an optimal solution for this instance, and S be a locally optimal solution with respect to the* add *and* pivot *operations. Then $c_s(S) \leq c_s(S^*) + c_f(S^*)$ and $c_f(S) \leq 6c_s(S^*) + 7c_f(S^*)$.*

4.3 Establishing polynomial running time

As discussed in Section 3, in order to be able to guarantee a polynomial running time, we need to make a significant improvement in every step. Therefore, at the end, we will find a solution S such that no local operation can improve its cost by more than a factor of $\frac{\varepsilon}{5n}c(S)$, whereas in our analysis we assumed the S cannot be improved by local operations at all.

The proof of Lemma 4.1 as well as Lemma 4.4 is based on selecting a set of at most n operations and summing up the corresopnding inequalities saying that the cost of each operation is nonnegative. By relaxing local optimality we must add up to n times the term $\frac{\varepsilon}{5n}c(S)$ to the resulting bound; that is, the claim of Lemma 4.1 becomes $c_s(S) \leq c_f(S^*) + c_s(S^*) + \frac{\varepsilon}{5}$ and the bound in Lemma 4.4 changes to $c_f(S) \leq 3(c_s(S^*) + c_s(S)) + 4c_f(S^*) + \frac{\varepsilon}{5}$. Combining these inequalities we get $c(S) \leq 8c(S^*) + \varepsilon c(S)$.

Theorem 4.8 *The algorithm described in Section 3 terminates after at most* $\frac{10n}{\epsilon} \ln \frac{c(S)}{c(S^*)}$ *iterations and outputs a solution of cost at most* $8/(1-\varepsilon)$ *times the optimum.*

4.4 Improving the approximation ratio by scaling

We can use the standard scaling techniques to slightly improve the approximation factor of our algorithm. Assume we scale all the facility costs by a factor of λ, and run the local search algorithm on the modified instance. If S denotes the locally optimal solution, by Theorem 4.7 we have $c_s(S) \leq c_s(S^*) + \lambda c_f(S^*)$ and $\lambda c_f(S) \leq 6c_s(S^*) + 7\lambda c_f(S^*)$. Combining these inequalities shows that the cost of S is at most $(1+6/\lambda)c_s(S^*)+(7+\lambda)c_f(S^*)$. Setting $\lambda = \sqrt{15}-3$, we get an approximation factor of $\sqrt{15}+4 \approx 7.873$.

5 Conclusion

In this paper, we presented the first constant-factor approximation algorithm for the universal facility location problem, generalizing many of the previous results on various facility location problems and slightly improving the best approximation factor known for the hard-capacitated facility location problem. We proved that the approximation factor of our algorithm is at most $8 + \varepsilon$, however, we do not know of any example on which our algorithm performs worse than 4 times the optimum (the example given in [18] shows that our algorithm sometimes performs 4 times worse than the optimum). A tight analysis of our algorithm remains an open question. Furthermore, the only inapproximability lower bound we know for the universal facility location problem is a bound of 1.463 proved by Guha and Khuller [8] for the uncapacitated facility location problem. Finding a better lower bound for the more general case of universal facility location is an interesting open question. In the non-metric case (i.e., when connection costs do not obey the triangle inequality), the UniFL problem (even in the uncapacitated case) is hard to approximate within a factor better than $O(\log n)$. Another open question is to find a $O(\log n)$-approximation algorithm for the non-metric UniFL problem.

Our algorithm was based on the assumption that the facility cost functions are non-decreasing. While this assumption holds in most practical cases, there are cases where solving the universal facility location problem with decreasing cost functions can be helpful. The load balanced (a.k.a. lower bounded) facility location problem in an example. This problem, which is a special case of UniFL with $f_i(u) = \bar{f}_i \cdot [u > 0] + \infty \cdot [u < \bar{l}_i]$ for given $\bar{f}_i$ and $\bar{l}_i$, was first defined by Karger and Minkoff [11] and Guha et al. [9] and used to solve other location problems. We still do not know of any constant factor approximation algorithm for this problem, and more generally for UniFL with decreasing cost functions. As in the hard-capacitated facility location problem, the integrality gap of the natural LP relaxation of this problem is unbounded, and therefore local search seems to be a natural approach. However, our analysis does not work for this case, since `add` operations are not necessarily feasible, and therefore Lemma 4.1 does not work.

References

[1] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit. Local search heuristics for k-median and facility location problems. In *Proceedings of 33rd ACM Symposium on Theory of Computing*, 2001.

[2] M.L. Balinski. On finding integer solutions to linear programs. In *Proc. IBM Scientific Computing Symposium on Combinatorial Problems*, pages 225–248, 1966.

[3] P. Bauer and R. Enders. A capacitated facility location problem with integer decision variables. In *International Symposium on Mathematical Programming (ISMP)*, 1997.

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

[5] N. Christofides and J.E. Beasley. An algorithm for the capacitated warehouse location problem. *European Journal of Operational Research*, 12:19–28, 1983.

[6] F.A. Chudak and D.P. Williamson. Improved approximation algorithms for capacitated facility location problems. In *Integer Programming and Combinatorial Optimization (Graz, 1999)*, volume 1610 of *Lecture Notes in Computer Science*, pages 99–113. Springer, Berlin, 1999.

[7] E. Feldman, F.A. Lehrer, and T.L. Ray. Warehouse locations under continuous economies of scale. *Management Science*, 12:670–684, 1966.

[8] S. Guha and S. Khuller. Greedy strikes back: Improved facility location algorithms. *Journal of Algorithms*, 31:228–248, 1999.

[9] S. Guha, A. Meyerson, and K. Munagala. Hierarchical placement and network design problems. In *Proceedings of the 41th Annual IEEE Symposium on Foundations of Computer Science*, 2000.

[10] M. Hajiaghayi, M. Mahdian, and V.S. Mirrokni. The facility location problem with general cost functions. unpublished manuscript, 2002.

[11] D. Karger and M. Minkoff. Building Steiner trees with incomplete global knowledge. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science*, 2000.

[12] L. Kaufman, M.V. Eede, and P. Hansen. A plant and warehouse location problem. *Operations Research Quarterly*, 28:547–554, 1977.

[13] M.R. Korupolu, C.G. Plaxton, and R. Rajaraman. Analysis of a local search heuristic for facility location problems. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1–10, January 1998.

[14] A.A. Kuehn and M.J. Hamburger. A heuristic program for locating warehouses. *Management Science*, 9:643–666, 1963.

[15] M. Mahdian, Y. Ye, and J. Zhang. Improved approximation algorithms for metric facility location problems. In *Proceedings of 5th International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX 2002)*, 2002.

[16] M. Mahdian, Y. Ye, and J. Zhang. A 2-approximation algorithm for the soft-capacitated facility location problem. unpublished manuscript, 2003.

[17] R.M. Nauss. An improved algorithm for the capacitated facility location problem. *Journal of Operational Research Society*, 29:1195–1202, 1978.

[18] M. Pál, E. Tardos, and T. Wexler. Facility location with hard capacities. *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science*, 2001.

[19] D.B. Shmoys. Approximation algorithms for facility location problems. In K. Jansen and S. Khuller, editors, *Approximation Algorithms for Combinatorial Optimization*, volume 1913 of *Lecture Notes in Computer Science*, pages 27–33. Springer, Berlin, 2000.

[20] J.F. Stollsteimer. A working model for plant numbers and locations. *J. Farm Econom.*, 45:631–645, 1963.