Facility Location with Nonuniform Hard Capacities

Martin Pál *　　Éva Tardos †　　Tom Wexler ‡
Cornell University

Abstract

In this paper we give the first constant factor approximation algorithm for the Facility Location Problem with nonuniform, hard capacities. Facility location problems have received a great deal of attention in recent years. Approximation algorithms have been developed for many variants. Most of these algorithms are based on linear programming, but the LP techniques developed thus far have been unsuccessful in dealing with hard capacities.

A local-search based approximation algorithm [11, 6] is known for the special case of hard but uniform capacities. We present a local-search heuristic that yields an approximation guarantee of $9 + \epsilon$ for the case of nonuniform hard capacities. To obtain this result we introduce new operations that are natural in this context. Our proof is based on network flow techniques.

1 Introduction

We consider the following metric capacitated facility location problem. We are given a set of clients and a set of facilities. Each client j has some demand d_j. Each facility i has a capacity u_i, which is the maximum amount of demand it can satisfy, as well as a facility cost f_i for opening i. A facility can only serve demand if it is opened. The service cost is based on a metric c_{ij} on the set of clients and facilities. We incur a service cost of c_{ij} for facility i to serve one unit of demand to client j. We do not require that all demand from a client be served by a single facility. We are interested in finding a set of facilities to open and an assignment of demand to facilities such that all client demand is satisfied, facility capacities are not violated, and the total cost of opening facilities and assigning demand is minimized. In this problem we say that the capacities are *hard* in that facility i can be opened at most once to serve at most u_i demand. This is in contrast to problems with *soft* capacities considered in several previous papers [9, 3, 1, 5], where facility i may serve a bit more than u_i demand, or may be opened k times to serve up to ku_i demand at a cost of kf_i.

There has been a large amount of work done in the study of approximation algorithms for metric facility location problems. A number of techniques, including LP rounding, primal-dual algorithms, and local search heuristics, have been used to yield constant factor approximations for various facility location problems. Linear programming techniques have proved to be very successful in solving uncapacitated problems, and lead to the first as well as the best known approximation algorithms for many of these problems. Linear programming based techniques has also been extended to capacitated problems with soft capacities. Chudak and Shmoys [5] gave an approximation algorithm for facility location with uniform, soft capacities. Jain and Vazirani [9] gave a technique for taking LP based approximation algorithms for the uncapacitated problem and extending them to handle soft capacities.

In this paper we consider the facility location problem with *hard* capacities. Hard capacities arise naturally in many applications (see, for example, [2]). However, hard capacities are known to be problematic for techniques that rely on linear programs (such as LP rounding and primal-dual algorithms). These techniques involve writing the facility location problem as a natural integer program, and then using the corresponding linear programming optimum as a lower-bound on the optimum solution cost. However, the natural linear program has a large integrality gap for the case of hard capacities, i.e., the fractional solution can have an arbitrarily small value compared to the best integer solution. The idea of soft capacities has been introduced precisely to get around this integrality gap. For problems with hard capacities, we would need a different, and probably less natural integer programming formulation to be able to use these techniques. There have been some attempts to find such an improved formulation for a closely related network design problem [7].

Local search heuristics have been popular algorithms for hard optimization problems. Korupolu, Plaxton, and Raja-

*Email: mpal@cs.cornell.edu
†Research supported in part by NSF grant CCR-9700163 and ONR grant N00014-98-1-0589. Email: eva@cs.cornell.edu
‡Research supported in part by an NSF graduate fellowship. Email: wexler@cs.cornell.edu

maran [10, 11] showed that a simple local search heuristic proposed by Kuehn and Hamburger [12] provides a constant factor approximation for several facility location problems. Quite surprisingly, their results extend to problems with uniform hard capacities. Chudak and Williamson [6] improved on the analysis of the same heuristic to show that this local search algorithm gives a $6+\epsilon$ approximation. Arya et al [1] consider a local search algorithm for a facility location problem with nonuniform capacities. However, their techniques can only handle soft capacities. In this paper we show how a more powerful version of the local search heuristic yields a $9+\epsilon$ approximation algorithm for the problem with nonuniform, hard capacities. Using scaling this bound can be slightly improved to $8.53+\epsilon$. This is the first constant approximation algorithm for this problem.

The local search algorithms for the case of uniform capacities use operations that open and/or close at most one facility at a time. To be able to deal with nonuniform capacities we need more general operations that allow us to trade off one large facility for many small ones. This suggests two natural operations: open one (large) facility, and close many smaller ones; and vice-versa, close one (large) facility, and open many smaller ones.

The main tool in our analysis is a transshipment problem, that arises in routing clients from the set of open facilities in a given solution to the set of facilities open in the optimum. Chudak and Williamson [6] use a analogous flow problem in the case of uniform capacities, which is equivalent to a bipartite matching problem in this case. They use the integrality of the bipartite matching polytope, and show that if the current solution is sufficiently expensive, then one of the edges of the optimum matching provides an improving swap move. In the case of nonuniform capacities, the analogous flow problem is an uncapacitated transshipment problem. However, the integrality of the transshipment polytope does not appear to be useful in our context. We are able to use the tree structure of the optimal solution to a transshipment problem to define a set of local moves, one of which must improve our current solution.

The remainder of this paper proceeds as follows. Section 2 gives a formal definition of the problem. Section 3 describes the algorithm. Section 4 defines the network flow tools used in bounding the cost of such a solution, including the *exchange graph*, which roughly corresponds to the swap graph of [6], except that our graph handles nonuniform capacities. Section 5 bounds the facility cost of a solution produced by the algorithm. Section 6 combines the results of the previous sections to bound the total cost of our solution.

2 Definitions

The input to the nonuniform capacitated facility location problem consists of a set F of *facilities* and a set D of *clients*. Each client j has an integer demand d_j that must be served by one or more open facilities. Every facility i has a cost f_i that is incurred if i is opened and an integer capacity u_i. There is also a cost c_{ij} to have an open facility i serve one unit of demand from client j. We allow for *splittable* demand, meaning we do not require that all demand of a client be served by a single facility. The metric version of the problem assumes that the costs c_{ij} are symmetric and satisfy the triangle inequality. For this problem we extend c to be a metric on the space $F \cup D$ and refer to it as *distance*.

A solution (S, x) to the facility location problem consists of a set of facilities S to be opened and an assignment x of demand to facilities. By $x(i,j)$ we denote the amount of demand of client j served by facility i. We require that the demand of every client j be fully served and no facility is used to more than its capacity. We use $x(i,\cdot)=\sum_j x(i,j)$ to denote the total demand served by facility i. The cost of the solution (S,x) is a sum of two parts: the *facility cost* $\sum_{i\in S} f_i$, and the *service cost* $\sum_{i\in F, j\in D} c_{ij}x(i,j)$. The problem is to find a set of facilities $S\subseteq F$ and an assignment x of demand to facilities that form a solution of minimum cost. In other words, the goal is to choose S and x to minimize

$$\sum_{i\in S} f_i + \sum_{i\in S, j\in D} c_{ij}x(i,j)$$

$$\begin{array}{llcll} \text{subject to} & \sum_{i\in S} x(i,j) & = & d_j & \forall j\in D \\ & \sum_{j\in D} x(i,j) & \leq & u_i & \forall i\in S \\ & x(i,j) & \geq & 0 & \forall i\in S,\ j\in D. \end{array}$$

Given a set of facilities S, the minimum cost assignment x using these facilities can be computed in polynomial time using a minimum cost transshipment algorithm, and hence x will be integral so long as demands and capacities are integral as well. We use $c_f(S)=\sum_{i\in S} f_i$ to denote the facility cost of solution S, and $c_s(S)$ to denote the minimum possible service cost $\sum_{i\in S, j\in D} c_{ij}x(i,j)$ for an assignment x of demand to facilities in S. We define $c(S)=c_f(S)+c_s(S)$.

3 The Algorithm

Our local search algorithm follows roughly the same scheme as originally proposed by Kuehn and Hamburger [12]. We start with an arbitrary solution (S,x) satisfying our constraints, and repeatedly perform one of three types of local transformations to improve the solution. We begin by describing the local steps that are available to the algorithm. We then discuss how the algorithm can efficiently select useful steps to take. For simplicity, we initially assume that all capacities and demands are small integers. This assumption will be lifted at the end of the section.

Our algorithm considers the following three types of operations.

- $\mathrm{add}(s)$ – Open a facility $s \in F$ and find the minimum cost assignment of clients to open facilities in $S \cup \{s\}$. This operation is also used by [11, 6]. The goal of this operation is to keep the service cost low.

- $\mathrm{open}(s, T)$ – Open one facility $s \in F$ and close a set of facilities $T \subseteq S - \{s\}$. With this operation we simply reroute all demand served by T to s. Any demand served by $s' \in S - T$ is not affected by this operation. We can think of this as finding a large facility to open that can handle the demand assigned to a set of small facilities, all of which may then be closed. Arya et al [1] considered a variant of this operation for facility location with soft capacities, with the modification that s could be opened any number of times.

- $\mathrm{close}(s, T)$ – Close one facility $s \in S$ and open a set of facilities $T \subseteq F - \{s\}$. Here we only reassign demand from s to facilities in T. Any demand served by $s' \in S - \{s\}$ is not affected by this operation. Intuitively, we are closing a big facility and opening a set of small facilities to handle the resulting demand.

Note that in these operations, we do not require that the facilities we select to open be closed initially. Opening an already open facility i allows us to reassign demand to i if i had not yet been used up to its full capacity. We say that an operation is *feasible* if the total free capacity in the opened facilities is at least as large as the total amount of demand served by the facilities being closed (i.e. $\mathrm{add}(s)$ is always feasible).

To use these steps in an effective local search algorithm we need to be able to select a good local step efficiently. The operations $\mathrm{close}(s, T)$ and $\mathrm{open}(s, T)$ involve selecting a set of facilities T and so we cannot try all possible operations in polynomial time. Instead, we will try all facilities s, and for each choice of s, in polynomial time decide if there is a subset T such that either $\mathrm{open}(s, T)$ or $\mathrm{close}(s, T)$ can be performed.

We need to estimate the affect that performing an operation has on the cost of a solution so that our algorithm can select an operation to apply. Ideally we would simply calculate the exact change in the cost of the solution for each operation. However, if we use this measure for $\mathrm{close}(s, T)$, it is impossible to efficiently determine whether or not there are any good operations (i.e. operations that decrease the cost of the current solution sufficiently). Given a facility s, the problem of finding a set of facilities T such that the operation $\mathrm{close}(s, T)$ yields the maximum decrease in cost is in itself the facility location problem (restricted to the clients served by facility s in the current solution). Therefore we need a more manageable estimation. With this in mind, we define the *estimated cost* $\mathrm{ec}(\mathrm{op})$ of an operation op based on the operation type.

- The estimated cost of $\mathrm{add}(s)$ is in fact the true change in cost caused by opening facility s and optimally assigning all demand to facilities in $S \cup \{s\}$.

- The estimated cost of $\mathrm{open}(s, T)$ is the true cost of opening s and reassigning demand from T to s, minus the facility cost of T. The cost of reassigning one unit of demand of client j from facility t to s is $c_{js} - c_{jt}$.

- For the estimated cost of $\mathrm{close}(s, T)$, we use a rough upper bound on the cost of rerouting demand from s to T. In computing the service cost of a client j that is rerouted from s to a facility $t \in T$ we use $c_{js} + c_{st}$ instead of the actual cost of c_{jt}. By the triangle inequality, $c_{js}+c_{st}$ is an upper bound on the actual cost. Using this upper bound, we estimate the increase in service cost of rerouting the demand of client j from s to a facility $t \in T$ by c_{st}. Let γ be the above estimate for the increase in the cost of rerouting all demand from s to T. This yields the following formula for the estimated cost: $\mathrm{ec}(\mathrm{close}(s, T)) = -c_f(s) + c_f(T - S) + \gamma$. In other words, $\mathrm{ec}(\mathrm{close}(s, T))$ is the net cost of closing s, opening T, and rerouting demand from s to T if sending one unit of demand to t increases the service cost by c_{st}.

We say that a solution (S, x) is *locally optimal* if there are no feasible operations with negative estimated cost. In order to guarantee that our local search algorithm terminates in polynomial time, we must make sure that the operations selected decrease the cost of the solution significantly. We define a feasible operation to be *admissible* if the operation has an estimated cost of at most $-c(S)/p(n, \epsilon)$ where $p(n, \epsilon)$ is a suitably chosen polynomial in n and $1/\epsilon$. If no transformation is found that yields such an improvement, the algorithm stops.

Using this definition we can now describe the algorithm as follows: repeatedly apply an admissible operation of one of the above three types until no admissible operation exists.

Lemma 3.1 *The algorithm terminates after at most* $O(p(n, \epsilon) \log \frac{c(S)}{c(S^*)})$ *operations.*

The algorithm insists that each local step lowers the cost of the current solution S by at least $c(S)/p(n, \epsilon)$. Doing so ensures termination in polynomial time. The drawback is that when the algorithm terminates the solution is not locally optimal: there may be operations that could improve the cost of the solution, but they only do so marginally. However, for the sake of simplifying the discussion, we will prove that any locally optimal solution has cost at most 9 times the optimum. At the end of Section 5 we show that dropping the assumption of local optimality yields a $(9+\epsilon)$ approximation ratio.

We now show that the algorithm can efficiently find admissible operations if any exist. The $\mathrm{add}(s)$ operation is fairly straight-forward, and we can easily determine whether any choice of s yields an admissible operation. However, for the remaining two operations $\mathrm{close}(s,T)$ and $\mathrm{open}(s,T)$, the brute force approach of trying all possible sets T can not be done in polynomial time. Fortunately, given s, there is an efficient way to find a good T for each operation. This is stated more precisely in the following lemmas.

Lemma 3.2 *Given $s \in S$, we can find a set T that has the minimum estimated cost $\mathrm{ec}(\mathrm{open}(s,T))$ among all subsets $T \in F - \{s\}$ in time polynomial in the number of facilities n and the capacity u_s.*

Proof. Consider a facility t in the current solution S, and for this proof, let $\mathrm{value}(t)$ denote the decrease in cost obtained by rerouting all demands from t to s. More precisely, $\mathrm{value}(t)$ is f_t, the facility cost saved by closing t, plus the decrease in service cost caused by rerouting all clients served by t to s (where negative decrease reflects an increase). With this definition of value the problem of finding the best set T is exactly a knapsack problem, where the newly opened facility s is the knapsack, its size is u_s. The items that may be packed in the knapsack are the facilities in the current solution S, where facility t has value $\mathrm{value}(t)$ and size $x(t,\cdot)$, the amount of demand served by t in the current solution. Under the assumption of integer demands and capacities, x is integral, and hence the knapsack problem can be solved in (weakly) polynomial time by dynamic programming. ∎

The algorithm for finding a good set T for the $\mathrm{close}(s,T)$ operation is in a way analogous to the one used to prove Lemma 3.2. To make the operation $\mathrm{close}(s,T)$ we need to select a set of facilities T such that the unused capacity of these facilities is at least the demand $x(s,\cdot)$ currently served by facility s. The true cost of this operation depends on exactly which clients of s get reassigned to t. Recall, however, that in computing the estimated cost $\mathrm{ec}(\mathrm{close}(s,T))$ of the operation, we estimate the change in service cost by c_{st} for each client. In essence, the problem we have to solve of finding the best set T for minimizing $\mathrm{ec}(\mathrm{close}(s,T))$ for a given facility s is the special case of the facility location problem, where there is only a single client node corresponding to s, and it has $x(s,\cdot)$ demand.

Lemma 3.3 *The special case of the facility location problem with one client j that has demand d_j can be solved in time polynomial in the demand d_j and the number of facilities n.*

Proof. Suppose we know the set S^* of facilities in the optimal solution (S,x). Let $t \in S^*$ be a facility that has maximal distance from j. It is clear that assignment x uses all facilities $i \in S - \{t\}$ to their full capacity u_i, and sends the remaining demand to facility t.

If we are given t, the furthest facility from j open in the optimum, we can find the optimal set of facilities by solving the following covering knapsack problem. Let $\mathrm{cost}(s) = f_s + u_s c_{sj}$ denote the cost of opening facility s and assigning it u_s units of j's demand. Our goal is to select a set $S' \subseteq F - \{t\}$ of facilities with a total capacity of at least $d_j - u_t$ such that $\sum_{s\in S'} \mathrm{cost}(s) + c_{tj}(d_j - \sum_{s\in S'} u_s)$ is minimized. This covering knapsack problem can be solved by dynamic programming in time polynomial in n and d_j. There are only n possible values of t to try, and so the special case of the facility location problem can be solved by solving n covering knapsack problems. ∎

Using this algorithm, we can find a good set T for the $\mathrm{close}(s,T)$ operation, giving us

Lemma 3.4 *Given $s \in S$, we can find a set T that has the minimum estimated cost $\mathrm{ec}(\mathrm{close}(s,T))$ among all subsets $T \in F - \{s\}$ in time polynomial in the number of facilities n and the capacity u_s.*

Finally, we have to return to the case when the demands and capacities are not required to be "small" integers. This assumption was used in Lemmas 3.2 and 3.4. In both cases, we used the assumption that we could solve knapsack problems exactly. In the case of general demands and capacities we will have to settle for solving these knapsack problems approximately. However, given a solution S that can be improved by an admissible operation, we can still find an operation that improves its cost by at least $\frac{1}{2}c(S)/p(n,\epsilon)$. Thus we still decrease cost of the solution "fast enough" so that the algorithm terminates after polynomial number of operations.

Theorem 3.5 *The local search algorithm described in this section finds a solution (S,x) that has no admissible operations in polynomial time.*

4 The exchange graph and bounding the service cost

The goal of the next two sections is the prove a bound on the cost of a solution obtained by the local search algorithm of the previous section. For most of these sections we will consider a locally optimal solution (S,x). At the end of Section 5 we show how to extend the arguments to yield a $(9+\epsilon)$ approximation ratio for a solution with no admissible local operations. Let (S^*,x^*) denote an optimum solution.

The primary tool we will use is the *exchange graph*, a transshipment graph that bounds the cost of reassigning clients from the currently open facilities to the facilities open in the optimum. Flows in the exchange graph describe a feasible way to exchange facilities in S for facilities in S^*. Chudak and Williamson [6] use an analogous construction called a swap graph for the case of uniform capacities.

We will use a natural path decomposition that shows how to transfer clients from all facilities in $S-S^*$ to the facilities in S^* in the optimal solution. Such a path decomposition has been used in several previous papers [10, 8, 6]. The path decomposition allows us to bound the service cost $c_s(S)$ of the locally optimal solution.

Furthermore, the path decomposition defines a flow of low cost in the exchange graph. In Section 5 we use the tree structure of a minimum cost flow in the exchange graph to define a set of open and close operations, which allow us to prove the claimed upper bound on the cost $c(S)$ of a locally optimal solution.

The path decomposition. We think of the assignment x as a flow in a bipartite graph with vertices corresponding to facilities and clients, where $x(s, j)$ units flows from each facility s to each client j. We want to compare an arbitrary solution (S, x) with the optimal solution (S^*, x^*). To do this, we consider the flow $x - x^*$, i.e. the flow where each edge (s, j) carries $x(s, j) - x^*(s, j)$ units of flow. Negative flow indicates flow in the reverse direction, i.e. from a client to a facility. Note that there is flow conservation at the clients: the amount of flow into each client j is d_j, as is the amount of flow out of j. In addition, there is no flow entering any $s \in F - S^*$, and no flow going out of any $s \in F - S$.

By standard path-stripping arguments, we decompose the flow $x-x^*$ into a set of flow paths $\mathcal{P}$ and cycles $\mathcal{C}$. Note that every path starts in a vertex in S and ends in a vertex in S^* and no cycle passes through a facility outside $S^* \cap S$. We will assume without loss of generality that there are no cycles in the decomposition. To see this, consider a cycle $C \in \mathcal{C}$. Both x and x^* are minimum cost transshipment, hence the cycle C must have 0 cost when viewed as an augmenting cycle for the flow x or x^*. Augmenting the flow x^* along each of these cycles yields a new optimal transshipment $\hat{x}^*$ and eliminates all cycles from the decomposition.

Let $w(P)$ be the amount of flow on path P and let $c(P) = \sum_{(s,j)\in P} c_{sj}$ be its cost. Note that $\sum_{P\in\mathcal{P}} c(P)w(P) \leq c_s(S) + c_s(S^*)$. Let $\mathcal{P}(s, t)$ be the set of paths that start at vertex s and end at vertex t. We use $\mathcal{P}(s, \cdot)$ to denote the set of paths that start at vertex s and $\mathcal{P}(\cdot, t)$ to denote the set of paths that end at vertex t.

Korupolu et al [11] and Guha and Khuller [8] independently proved a version of the following lemma. We include the proof for completeness.

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

Proof. Let (S, x) be a solution that cannot be improved by any $\mathrm{add}(s)$ operation and let (S^*, x^*) be an optimal solution. Consider the path decomposition of $x - x^*$. Slightly abusing the notation, we may think of every path P in the decomposition as a flow that sends $w(P)$ units on every edge $e \in P$, and any set of paths $\mathcal{P}$ as just a sum of the flows $P \in \mathcal{P}$. Using this notation, $x = x^* + \mathcal{P}$.

Consider the flow $x - \mathcal{P}(\cdot, t)$ for $t \in S^*$. This flow defines a feasible way to reassign clients after adding facility t to S. Namely, the total flow into every client j is d_j, the flow out of every facility s is nonzero only if $s \in S \cup \{t\}$, the flow out of every facility t' is at most $u_{t'}$, and flow on each edge (s, j) is nonnegative. The service cost of a solution is just the cost of the corresponding flow. Therefore, the service cost resulting from the $\mathrm{add}(t)$ operation is at most the cost of this flow. Let $\delta_t = c_s(x) - c_s(x - \mathcal{P}(\cdot, t))$ be this lower bound for the change in service cost. Since no add operation improves the cost of the solution, δ_t must be at most f_t for any $t \in S^*$.

We write $x = x^* + \sum_{t\in S^*} \mathcal{P}(\cdot, t)$. By linearity of cost, we have $c_s(x) = c_s(x^*) + \sum_{t\in S^*} \delta_t$. Combining this with the inequalities that $\delta_t \leq f_t$ for all $t \in S^*$ gives the lemma. ∎

The exchange graph. The exchange graph is a bipartite graph corresponding to a transshipment problem in which flow is shipped from facilities in $S - S^*$ to facilities in S^*. Intuitively, the flow corresponds to clients. The cost of an edge (s, t) is c_{st}. This cost is a simple upper bound on the increase in service cost incurred by reassigning one unit of demand from s to t. The idea is that we want to close every facility in $S - S^*$ and transfer the demand it serves to facilities in S^*. Therefore we require the flow out of every such facility s to be exactly $x(s, \cdot)$, and the flow entering a facility $t \in S^*$ to be at most the free capacity of t, $u_t - x(t, \cdot)$. We use $y(s, t)$ to denote the amount of flow on edge (s, t). We formulate the problem as follows:

$$\text{minimize} \quad \sum_{s\in S-S^*, t\in S^*} c_{st}\, y(s, t)$$

subject to

$$\begin{array}{rcll} \displaystyle\sum_{t\in S^*} y(s,t) & = & x(s,\cdot) & \forall s \in S - S^* \\ \displaystyle\sum_{s\in S-S^*} y(s,t) & \leq & u_t - x(t,\cdot) & \forall t \in S^* \\ y(s,t) & \geq & 0 & \forall s \in S - S^*, t \in S^* \end{array}$$

First we claim that there is a low cost feasible flow, despite

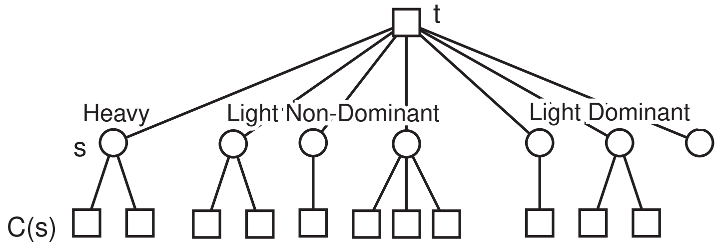

Figure 1. A subtree $\mathcal{T}_t$ rooted at $t \in S^*$ with children in H, LN and LD.

the fact that we used the simple upper bound on the cost of transferring demand between facilities.

Lemma 4.2 *There is a flow in the exchange graph with cost at most $c_s(S) + c_s(S^*)$.*

Proof. We use the path decomposition described above to construct such a flow. For a pair of facilities $s \in S - S^*$ and $t \in S^*$ send $y(s,t) = \sum_{P \in \mathcal{P}(s,t)} w(P)$ units of flow from vertex s to vertex t. The total flow leaving every vertex $s \in S - S^*$ is exactly $x(s,\cdot)$, as all of the paths in $\mathcal{P}(s,\cdot)$ end in a node $t \in S^*$. Furthermore, the total flow entering any vertex $t \in S^*$ is at most $\max(0, x^*(t,\cdot) - x(t,\cdot))$, which is at most $u_t - x(t,\cdot)$. This proves that the flow y is feasible.

Consider edge (s,t). The cost of the flow on this edge is $c_{st}y(s,t)$. We would like to bound this cost in terms of the costs of flows x and x^*. By the triangle inequality, $c_{st} \leq c(P)$ for every path $P \in \mathcal{P}(s,t)$. Thus the cost of the flow on edge (s,t) is at most the total cost of the flows on (s,t)-paths in the decomposition. More formally, we get that $c_{st}y(s,t) \leq \sum_{P \in \mathcal{P}(s,t)} c(P)w(P)$. Summing over all edges we get the claimed bound on the total cost of flow y:

$$\begin{aligned} \sum_{s \in S - S^*, t \in S^*} c_{st}y(s,t) &\leq \sum_{P \in \mathcal{P}} c(P)w(P) \\ &\leq c_s(S) + c_s(S^*). \end{aligned}$$

■

A flow is said to be acyclic if the graph consisting of edges carrying nonzero flow does not contain an (undirected) cycle. By augmenting the flow along cycles, we can modify any flow y to obtain an acyclic flow y' of no greater cost. In particular,

Lemma 4.3 *There is a minimum cost flow y in the exchange graph whose nonzero edges form a forest.*

5 Bounding facility costs

The goal of this section is to bound the facility cost $c_f(S)$ for a locally optimal solution S. To do this we will define a set of feasible local operations of the form close and open. Since S can not be improved by any operation, none of the operations we define can have negative estimated cost. Recall that the estimated cost of an operation is nonnegative if the amount saved by closing facilities is at most the cost of opening facilities plus the estimated cost of reassigning demand. We will add the resulting inequalities to obtain a bound on the overall facility cost $c_f(S)$.

We use the flow given by Lemma 4.3 to generate a list of feasible local operations. Every facility in $S - S^*$ is closed in exactly one such operation, and hence summing the inequalities gives a bound on $c_f(S)$. We are able to choose a set of operations such that every facility in S^* is opened by only a constant number of operations. We keep the estimated reassignment cost low by requiring that all the demand is transfered along flow edges and thus the flow "pays for" the reassignment.

We will begin by explaining how to generate this list of operations. Consider the forest formed by edges with nonzero flow in a minimum cost flow y from Lemma 4.3. Root each tree $\mathcal{T}$ at some vertex $r \in S^*$. For a vertex t we define $C(t)$ to be the set of children of t. For a $t \in S^*$ that is not a leaf of $\mathcal{T}$, let $\mathcal{T}_t$ be the subtree of depth at most 2 rooted at t and containing all children and grandchildren of t. See Figure 1 for an example of such a tree $\mathcal{T}_t$. Note that since the exchange graph is bipartite, for a node $t \in S^*$ the children $C(t) \subseteq S - S^*$.

We define a set of operations for every such subtree $\mathcal{T}_t$. In the simple case when all children of t are leaves, we consider the single operation $\mathrm{open}(t, C(t))$. This operation is feasible, since the total flow out of all children of t cannot exceed the total flow into t, which is at most $u_t - x(t,\cdot)$. The increase in service cost caused by this operation is bounded by the cost of the flow on the edges between t and its children.

Now consider the case in which not all children of t are leaves, as shown in Figure 1. One would be tempted to use operations $\mathrm{close}(s, C(s) \cup t)$ for all nodes $s \in C(t)$. Each operation is feasible since we open all neighbors of s. However, the facility t may be expensive and may have many children, so we need to avoid opening it $|C(t)|$ times.

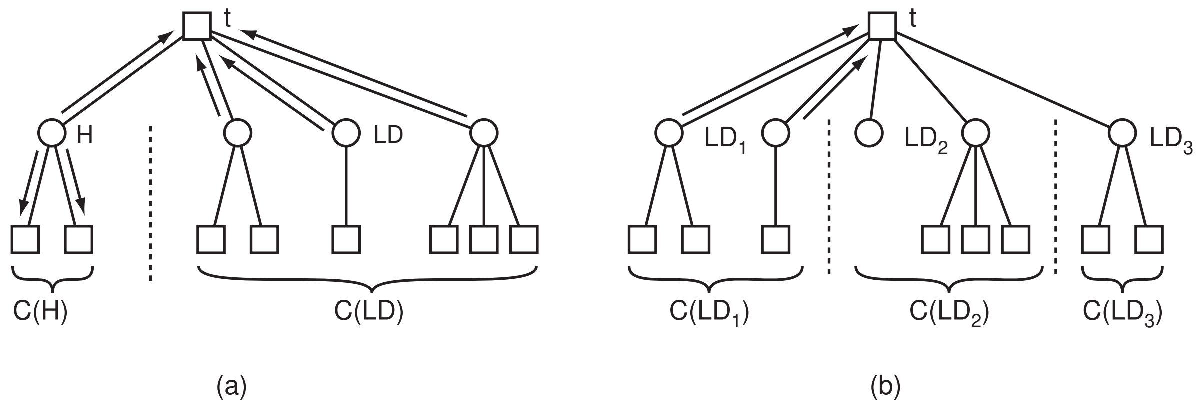

Figure 2. The two operations needed to reassigning demand from nodes in H and LD when $H \neq \emptyset$ (a) and one of the three operations needed when $H = \emptyset$ (b). Nodes in LN are not shown.

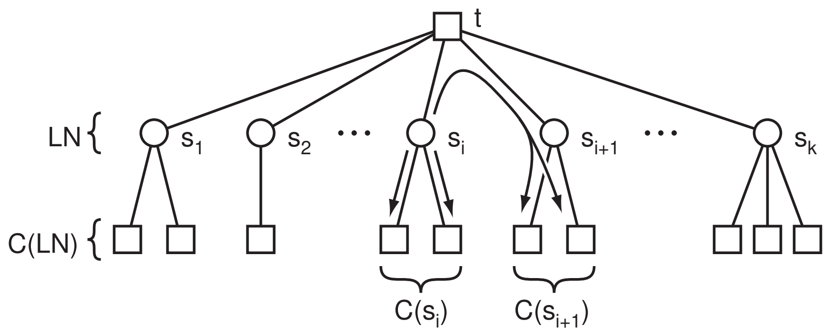

Figure 3. Reassigning demand from $s_i \in LN$. Nodes in H and LD are not shown.

To do this we need to distinguish a few cases.

For $s \in C(t)$, we say that s is *heavy* if $y(s,t) > \frac{1}{2}y(\cdot,t)$ and it is *light* otherwise. A light facility s is *dominant* if $y(s,t) \geq \frac{1}{2}y(s,\cdot)$, and otherwise it is *non-dominant*. We partition $C(t)$ into three sets. H will be the set of heavy facilities, LN the set of light non-dominant facilities, and LD is the set of light dominant facilities. We will define operations for each set separately.

Note that H contains at most one facility. For $s \in H$ we consider the operation $\mathrm{close}(s, C(s) \cup t)$.

For each $s \in LD$, the edge (s,t) carries at least a half of the flow out of s. If t has a heavy child (i.e. $H \neq \emptyset$), then the operation $\mathrm{open}(t, LD)$ is feasible: at most half of the flow entering t comes from the set LD, and hence the total demand of LD is at most the capacity of t. For the case when $H \neq \emptyset$, the two operations needed to close facilities from H and LD are shown in Figure 2(a).

If t has no heavy children, as in Figure 2(b), the total demand of LD can be as much at twice the capacity of t. However, for any $s \in LD$, $y(s,t)$ is at most half of the capacity of t, by the definition of light nodes. Thus the total demand at s does not exceed the capacity of t, and hence the set LD can be divided into at most three sets LD_1, LD_2, and LD_3 such that the total demand in each set is no more than the capacity of t. Then we consider operations $\mathrm{open}(t, LD_i)$ for $i = 1, 2$ and 3. Figure 2(b) demonstrates the operation $\mathrm{open}(t, LD_1)$.

It is a bit harder to deal with the light non-dominant nodes. For each $s \in LN$, we define a $\mathrm{close}(s,\cdot)$ operation, but excluding a single $s \in LN$, none of these operations will open t.

Order the elements $s_1, s_2, \ldots, s_k$ of LN by the amount of flow in their edge to t, i.e. $y(s_1,t) \leq y(s_2,t) \leq \ldots \leq y(s_k,t)$. Here we define a close operation for each $s \in LN$. For node s_k we consider the operation $\mathrm{close}(s_k, C(s_k) \cup t)$.

For s_i, $1 \leq i < k$ we have a more sophisticated operation $\mathrm{close}(s_i, C(s_i) \cup C(s_{i+1}))$ (see Figure 3). By our ordering, the amount of flow on the edge (s_i,t) is less than or equal to the amount of flow on the edge (s_{i+1},t) and, since s_{i+1} is non-dominant, it is also less than or equal to the total capacity of facilities in $C(s_{i+1})$. Hence $C(s_i) \cup C(s_{i+1})$ has enough capacity to compensate for the closing of s_i and thus the proposed operation is feasible.

We have now defined a set of operations for each tree $\mathcal{T}_t$ for $t \in S^*$. We use the fact that S is locally optimal, and hence each operation must have nonnegative estimated cost. The next two lemmas bound the facility costs and the estimated reassignment costs involved in these operations.

Lemma 5.1 *In the set of feasible operations defined above, each facility $s \in S - S^*$ is closed exactly once, and every facility $t \in S^*$ is opened at most 6 times.*

Proof. Every facility $s \in S - S^*$ belongs to exactly one subtree $\mathcal{T}_t$, and is a child of t in its subtree. The operations defined for a subtree $\mathcal{T}_t$ close every facility $s \in C(t)$ exactly once.

Each facility $t \in S^*$ is the root of a subtree $\mathcal{T}_t$ at most once. As the root, it can be opened once for the facility in H and once for facilities in LD, or three times for facilities in LD. It can also be opened once for facilities in LN. Every facility t is also a leaf of at most one subtree $\mathcal{T}_{t'}$. In this case, $t \in C(s)$ for some $s \in \mathcal{T}_{t'}$. If s is dominant in $\mathcal{T}_{t'}$, t is opened once. If $s \in LN$, t is opened by one or two operations. If $s \in LD$, t is not opened at all. ∎

Next we want to bound the increase in service cost incurred by the set of operations. For an operation $\mathrm{open}(s, T)$ let the *reassignment cost* of the operation be the cost of reassigning clients from facilities T to the facility s. For the operation $\mathrm{close}(s, T)$, let its *reassignment cost* be the cost of optimally reassigning of demand from s to facilities in T, where the cost of reassigning one unit of demand from s to t is c_{st}. Note that for each operation, the estimated cost used by the algorithm, is exactly the reassignment cost plus the cost of facilities opened minus the cost of facilities being closed. Further, note that the reassignment cost of operation $\mathrm{open}(s, T)$ can be bounded from above by using c_{st} as an estimate for the cost of reassigning one unit of demand from t to s.

Let $\alpha = \sum_{s \in S - S^*, t \in S^*} c_{st} y(s,t)$ denote the total cost of the flow y in the exchange graph.

Lemma 5.2 *The total reassignment cost of all operations is at most 2α.*

Proof. Consider all operations in some subtree $\mathcal{T}_t$. For $s \in H$, the operation $\mathrm{close}(s, C(s) \cup \{t\})$ reassigns $y(s, t')$ units of demand to every neighbor t', and hence the cost of the reassignment is paid for by the flow in edges adjacent to s.

The operation $\mathrm{open}(t, LD_i)$ for $LD_i \subseteq LD$ reassigns at most $2y(s,t)$ units of demand from s to t for each $s \in LD_i$. Thus the edge (s,t) pays at most twice the cost of the flow adjacent to nodes in LD.

Finally, consider the facilities in $LN = \{s_1, s_2, \ldots s_k\}$, ordered by $y(s_i, t)$. The reassignment cost of operation $\mathrm{close}(s_k, C(s_k) \cup \{t\})$ is paid for by the edges adjacent to s_k.

For $i < k$, the operation $\mathrm{close}(s_i, C(s_i) \cup C(s_{i+1}))$ reroutes $y(s_i, t')$ demand to each facility $t' \in C(s_i)$. The flow in the edge (s_i, t') pays for this. The operation has also to reroute $y(s_i, t)$ units of demand to facilities in $C(s_{i+1})$ (see Figure 3.) Divide the demand between facilities $t_1, t_2, \ldots, t_l \in C(s_{i+1})$ so that every facility t_j gets at most $y(s_{i+1}, t_j)$ demand. By the triangle inequality, $c_{s_i t_j} \le c_{s_i t} + c_{t s_{i+1}} + c_{s_{i+1} t_j}$, and hence this part of rerouting is paid for by the flow in edges adjacent to s_{i+1} plus the edge (s_i, t).

Since every edge pays for at most two operations, the claim follows. ∎

6 Approximation Guarantee

We can now bound the cost of a solution produced by our algorithm in terms of the cost of an optimal solution.

Theorem 6.1 *A locally optimal solution S has cost at most 9 times the optimum cost for the facility location problem with hard nonuniform capacities.*

Proof. By Lemmas 5.1 and 5.2 the total estimated cost of all operations is at most $6c_f(S^* - S) + 2\alpha - c(S - S^*)$. Lemma 4.2 bounds $\alpha \le c_s(S^*) + c_s(S)$. All operations have nonnegative estimated cost, and therefore

$$c_f(S - S^*) \le 6c_f(S^* - S) + 2\left[c_s(S) + c_s(S^*)\right].$$

Adding $c_s(S) + c_f(S \cap S^*)$ to both sides, using Lemma 4.1 to bound the $c_s(S)$ term by $c_s(S^*) + c_f(S^*)$, and rearranging we get

$$c(S) \le 9c_f(S^*) + 5c_s(S^*).$$ ∎

In order to guarantee polynomial running time, we had to insist that each operation lower the cost of the current solution by at least $c(S)/p(n, \epsilon)$. As a result, when the algorithm terminates, there may be operations that could improve the cost of the solution, albeit marginally. The proof of Theorem 6.1 can be easily modified to handle the solution (S, x) obtained by the algorithm. First, we modify the proof of Lemma 4.1 to show that if no add operation is admissible then $c_s(S)$ is at most $c(S^*) + \frac{nc(S)}{p(n,\epsilon)}$. Now a local operation may have negative estimated cost, but it cannot be admissible. Adding the resulting inequalities, as was done in the proof above and choosing $p(n, \epsilon) = \frac{3n}{\epsilon}$ we get the following slightly weaker bound.

Theorem 6.2 *The local optimization algorithm in Section 3 is a* $9 + \epsilon$ *approximation algorithm for facility location with hard nonuniform capacities.*

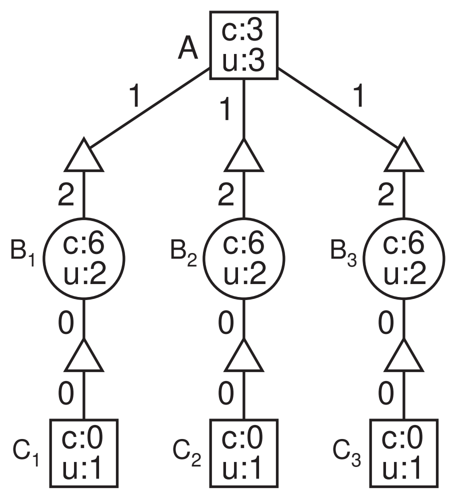

Figure 4. A locally optimal solution with four times the cost of the optimum solution.

We also present a lower bound on the approximation ratio of our algorithm.

Theorem 6.3 *The approximation ratio of our algorithm is at least 4.*

Sketch of Proof. Consider Figure 4. Triangles represent clients with one unit of demand each. Circles and squares represent facilities with costs c and capacities u. Numbers on edges represent distances, and edges which are not drawn have lengths as induced by the given edges. The set of facilities shown as squares (A, C_1, C_2, C_3) is the optimum solution, with facility cost 3 and service cost 3. However, the set of facilities shown as circles (B_1, B_2, B_3) represents a valid solution with a facility cost of 18 and a service cost of 6. Furthermore, it is easily verified that this solution is locally optimal. Note that although a swap in which facility B_1 is closed and facilities A and C_1 are opened could improve the cost of our solution, the net gain that our algorithm calculates using the estimated cost of the close operation is 0. The same holds for closing B_1 and opening C_1 and C_2. ■

Lastly, we show that by using standard scaling techniques [8, 3], we can marginally improve the approximation guarantee of our algorithm.

Theorem 6.4 *The scaled version of our algorithm has an approximation guarantee of* $8.532 + \epsilon$.

Sketch of Proof. Given an input problem, we first scale all service costs c_{ij} by a constant λ, whose value we will specify later. We then run our algorithm on this modified version of the problem, yielding a solution S. Using a similar argument to that of Lemma 4.1, we can show that $\lambda c_s(S) \leq c_f(S^*) + \lambda c_s(S^*)$. We can also follow the reasoning of Lemma 4.2 to show that the exchange graph has a flow with cost at most $\lambda(c_s(S) + c_s(S^*))$. Carrying these results through Theorem 6.1 yields $c(S) \leq (8 + \frac{1}{\lambda})c_f(S^*) + (4\lambda + 1)c_s(S^*)$. Selecting $\lambda = 1.883$ gives us $c(S) \leq 8.532c(S^*)$. ■

Note that if we run the scaled version of our algorithm on the example in Figure 4 we obtain the optimal solution. However, if we consider the same problem instance with all distances multiplied by $\frac{1}{1.883}$, then after scaling the algorithm is left with exactly the example in Figure 4, which we have already shown to be locally optimal. Furthermore, such a solution has a cost of more than 4.6 times that of the optimal solution. Hence, although we can prove a better worst-case bound on the performance of the scaled algorithm, we also observe that it has poorer performance in some cases than the unscaled version.

References

[1] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Mungala, and V. Pandit. Local Search Heuristic for k-median and Facility Location Problems. *Proceedings of the 33st Annual ACM Symposium on the Theory of Computing*, 2001, pp. 21-29.

[2] F. Barahona and D. Jensen. Plant location with minimum inventory. *Mathematical Programming* 83:101-111, 1998.

[3] M. Charikar and S. Guha. Improved Combinatorial Algorithms for Facility Location and K-Median Problems. *Proceedings of the IEEE Foundations of Computer Science*, 1999.

[4] M. Charikar, S. Guha, E. Tardos and D. Shmoys. A constant-factor approximation algorithm for the k-median problem. *Proceedings of the 31st Annual ACM Symposium on the Theory of Computing*, 1999, pp. 1-10.

[5] F. A. Chudak and D.B. Shmoys. Improved approximation algorithms for the capacitated facility location problem. *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1999.

[6] F. A. Chudak and D. P. Williamson. Improved Approximation Algorithms for Capacitated Facility Location Problems. *Proceedings of the 7th International IPCO Conference*, June 1999.

[7] R. D. Carr, L. Fleischer, V. J. Leung, and C. A. Phillips. Strengthening Integrality Gaps for Capacitated Network Design and Covering Problems. *Proceedings of the 11th ACM/SIAM Symposium on Discrete Algorithms*, January 2000.

[8] S. Guha and S. Khuller. Greedy strikes back: Improved Facility Location Algorithms. *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, 1998.

[9] K. Jain and V. Vazirani. Primal-Dual Approximation Algorithms for Metric Facility Location and k-Median Problems. *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, October 1999.

[10] M. Korupolu, C. Plaxton and R. Rajaraman. Analysis of a Local Search Heuristic for Facility Location Problems. *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, January 1998.

[11] M. Korupolu, C. Plaxton and R. Rajaraman. Analysis of a Local Search Heuristic for Facility Location Problems. *Technical Report 98-30, DIMACS*, June 1998.

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

[13] J. K. Lenstra, D. B. Shmoys and É. Tardos. Approximation Algorithms for Scheduling Unrelated Parallel Machines. *Mathematical Programming* 46 (1990), pp. 259-271.

[14] D. B. Shmoys and É. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming* 62 (1993), pp. 461-474.

[15] D. B. Shmoys, É. Tardos and K. Aardal. Approximation algorithms for the facility location problem. *Proceedings of the 29th Annual ACM Symposium on the Theory of Computing*, 1997, pp. 265-274.