Building Steiner Trees with Incomplete Global Knowledge

David R. Karger[*] Maria Minkoff[†]
MIT Laboratory for Computer Science
Cambridge, MA 02139, USA
{karger, mariam}@theory.lcs.mit.edu

Abstract

A networking problem of present day interest is that of distributing a single data item to multiple clients while minimizing network usage. Steiner tree algorithms are a natural solution method, but only when the set of clients requesting the data is known. We study what can be done without this global knowledge, when a given vertex knows only the probability *that any other client will wish to be connected, and must simply specify a fixed path to the data to be used in case it is requested. Our problem is an example of a class of network design problems with* concave *cost functions (which arise when the design problem exhibits economies of scale).*

In order to solve our problem, we introduce a new version of the facility location problem: one in which every open facility is required to have some minimum amount of demand assigned to it. We present a simple bicriterion approximation for this problem, one which is loose in both assignment cost and minimum demand, but within a constant factor of the optimum for both. This suffices for our application. We leave open the question of finding an algorithm that produces a truly feasible approximate solution.

1. Introduction

A networking problem of present day interest is that of distributing a single data item to multiple requesters while minimizing network usage. In the presence of caches, it is never necessary for an item to traverse a network link more than once. Thus, this problem can be modeled as a Steiner tree problem: connect all requesters to the root using a minimum cost set of edges. However, such modeling neglects the cost, in time and algorithmic complexity, of determining the full set of requesters who want an item. Because such global coordination can be difficult, many present day Internet algorithms are designed to make use only of local information, accepting that some degradation in performance may result as a consequence of this simplicity.

Such a philosophy has led us to study the following *maybecast* problem. We are given a network (possibly with edge costs), a *root* node, and a collection of N *clients* at nodes in the network. Each client i will choose to contact the root independently from others with some probability p_i. In advance of the requests we must choose, for each client in the network, a path from the client's node to the root. If a client chooses to contact the root, the edges on this path become *active*. Our goal is to minimize, over the random choices of the clients, the expected number (or cost) of active network edges. We can think of this as a probabilistic version of the rooted Steiner tree problem, in which one desires to connect a set of terminals with the root while minimizing total edge cost.

This problem models the data distribution problem mentioned above. If caches are placed at nodes in the network then as soon as a requester's path to the root encounters a cached copy of the item, the item can be returned without traversing the rest of the path. Thus every edge on the set of paths is traversed at most once by the item, so the total number (resp. total cost) of edges on the paths connecting root to requesters reflects the total network bandwidth (resp. total transmission cost) allocated to the distribution of the item. Alternatively, we can note that traversing a network link takes some finite amount of time. Assuming that requests occur at sufficiently spread-out times, we can expect that at most one request will be forced to wait for the item to traverse any given link—all future requests will be able to use the cached item immediately. Thus, the total number of links reflects the average time spent waiting by clients for the item.

Of course, in the absence of *any* information about the world, it is impossible to make sensible local decisions. We therefore aim to separate relatively stable global information (the structure of the network) that can be used for preprocessing from unpredictable information (the set of

[*]Research supported by NSF grant CCR-9624239 and a Packard Foundation Fellowship
[†]Research supported by AT&T Labs Foundation Fellowship.

clients) that becomes available too late to be useful. We also aim for the solution to have a simple local specification—a path for each client. (Alternatively, we might require a "routing tree" solution, in which each node simply specifies a parent to which it forwards any request it receives, and the path from any client is determined by following parent pointers. We will show that this is in fact equivalent.) In the case of large items, one might devote more time to "planning" the transmission of the item when it is requested; however, in the (not uncommon) case of small items, even limited planning may be too expensive.

Our problem is of course $\mathcal{NP}$-complete, since the Steiner tree problem is a special case. In fact, our problem remains $\mathcal{NP}$-complete even in the uniform case of one client per network node and unit costs on edges. Thus, the focus of this paper is on approximation algorithms. We give a constant-factor approximation algorithm. Our solution relies on some structural analysis of the optimum solution which leads to application of facility location and Steiner tree algorithms to solve the problem.

1.1. Overview of Results

We begin with a study of the optimum solution. We show that the optimum solution is invariably a tree. However, the obvious first choice of a shortest path tree can cost a factor as large as $\Theta(n^{1/2})$ times the optimum in an n-node graph. To find a better tree, we note that the optimum tree consists of a central "hub" area within which all edges are basically certain to be used, together with a fringe of "spokes" in which multiple clients can be considered to be contributing *independently* (and linearly) to the cost of the solution. We use a facility location algorithm to identify a good set of "hubs" to which we route clients at independent (linear) costs, and then use a Steiner tree algorithm to connect the hubs to the root.

To identify a good set of hubs, we introduce a new version of the facility location problem: one in which every open facility is required to have some minimum amount of demand assigned to it. This problem can also be phrased as a clustering problem where we wish to minimize the average radius of clusters without letting any cluster be too small. We present a simple bicriterion approximation for this problem, one which is loose in both assignment cost and minimum demand. This suffices for our application. We leave open the question of finding an algorithm that produces a truly feasible approximate solution.

1.2. Related Work

The Steiner tree problem has a constant factor approximation algorithm, but the set of terminals must be known in advance. There has been work on an *online* algorithm for the Steiner tree problem, in which terminals are revealed one at a time and must immediately be connected to the previously constructed Steiner tree [3]. This approach is stronger than ours on that it constructs a solution within $O(\log^2 N)$ of the optimal solution for the specific set of terminals that is chosen, while we are only competitive against the expected solution cost over all sets of terminals. However, it is also weaker since, like the standard Steiner tree algorithm, the online algorithm requires at each step full knowledge of the previously existing terminal set. Ours makes its routing decisions oblivious to what set of terminals will eventually be chosen.

Alternatively, the maybecast problem can be represented as a kind of min-cost flow problem with infinite capacities and a *concave* cost function. We can think of a client i as having "demand" for a flow of capacity p_i to the root. The cost of routing along an edge exhibits an *economy of scale*: the more paths use an edge, the cheaper it is per path. By approximating our cost function by a piece-wise linear function, we can establish a connection with the "buy-at-bulk" network design problem, another problem which exhibits an economy of scale. As an example, consider the problem of wiring a telephone network in some metric space such that every pair of clients is connected by a path of capacity one. Our approximate cost function would correspond to the situation in which two types of wires are available: low cost wires of infinitesimal capacity, and "infinite" capacity wires of large cost. Awerbuch and Azar [2] provide a randomized $O(\log^2 n)$ approximation algorithm for this problem, where n is the number of nodes in the network. Their approach relies on the tree metric embedding of Bartal; subsequent improvements in tree embeddings yield a slightly better approximation ratio of $O(\log n \log\log n)$. Andrews and Zhang [1] give an $O(K^2)$-approximation algorithm, where K is the number of different types of wires, for a special case of the problem. However, the restrictions which they place on the cost function preclude application to our problem. Thus, our work is the first that can achieve a constant factor approximation ratio for our problem.

Simultaneously and independently, Guha, Meyerson and Munagala [6] studied the same facility location problem variant that we use in our solution, and developed essentially the same solution. They generalized the approach to a hierarchical setting that gave results for the buy-at-bulk network design problem.

2. Preliminaries

In this section we formally define the maybecast problem and provide some preliminary results regarding the structure of the optimum solution. We show that the paths of the optimum solution must define a tree, but that the obvious shortest paths tree can be a terrible approximation.

2.1. Problem Statement

Input: We consider an undirected graph $G = (V, E)$ with a non-negative edge weight function $l : E \to \Re_+$ and a root vertex $r \in V$. A set of N clients is assigned to a subset of vertices $S \subset V$. Client i becomes active independently with probability $p_i > 0$, in which case it communicates with the root r along some to-be-specified path. Every edge on the path from an active client to the root becomes *active*.

Output: Construct a set of paths connecting each client to the root; this is the path that will be used if the client becomes active. The goal is to minimize the *expected* total weight of *active* edges.

This problem can be thought of as a probabilistic version of the rooted Steiner tree problem: we are given some subset of nodes each of which needs to be connected with the root with some probability. In order to study this problem, we think of l_e as the length of edge e and define a per-unit-length edge cost function c_e reflecting the probability that an edge will be used. Given a solution, let us denote by U_e the set of clients using edge e to communicate with the root. Then $c_e = \Pr[e \text{ is active}] = 1 - \prod_{i \in U_e}(1 - p_i)$. Using linearity of expectation, we can express the objective function as the sum over all edges of probabilities that an edge is active weighted by its length

$$E[\text{weight of active edges}] = \sum_{e \in E} \left(1 - \prod_{i \in U_e} (1 - p_i)\right) l_e$$

2.2. Solution Structure

In this section, we show that the optimum solution to any maybecast problem is a tree.

Given a set of paths connecting each client with a root, let us think of the solution as a flow along those paths. A client i contributes $d_i = -\ln(1 - p_i)$ units of flow to every edge on its path to the root. Thus, a given edge e carries $f_e = \sum_{i \in U_e} - \ln(1-p_i)$ units of flow. The cost of the flow on that edge is given by

$$\begin{aligned} \left(1 - \prod_{i \in U_e} (1 - p_i)\right) l_e &= \left(1 - \prod_{i \in U_e} e^{-d_i}\right) l_e \\ &= (1 - e^{-f_e}) l_e \end{aligned}$$

This cost function is concave in f_e, the flow on that edge.

If we have two distinct paths between the same pair of nodes with non-zero flows f_1 and f_2 on each, by properties of concave functions, $cost(f_1 + f_2) \leq cost(f_1) + cost(f_2)$. This suggests that an optimal solution never sends flow from a given node to the root along 2 distinct paths. We deduce that it has to be a tree. We can actually prove this in a general setting.

Theorem 2.1. *Consider any single-sink min-cost flow problem where capacities are infinite and every edge e has a nondecreasing concave cost function c_e of the total flow actually carried by that edge. Then there is an optimal (min-cost) solution to the flow problem that is a tree (that is, every vertex has at most one outgoing edge carrying flow).*

Proof. For simplicity we provide a proof only for the case where all cost functions are increasing and *strictly* convex $(f(\lambda x + (1 - \lambda)y) > \lambda f(x) + (1 - \lambda) f(y))$.

Consider an optimum solution. Let us decompose the the flow into paths carrying flow to the root and cycles carrying flow in circles. Note that if there is a flow-carrying cycle, we can decrease flow on every edge of the cycle, decreasing the solution cost, a contradiction. So the optimum flow is decomposed into paths to the root.

We will now show that this solution must be a tree. For the sake of contradiction, assume the solution is not a tree. This can only happen if there exist two clients i and j with paths to the root that intersect at a non-root vertex and then separate, i.e. use 2 different edges out of that vertex (as a special case we might have $i = j$). Let v be a vertex where the separation happens and let w be the next intersection of the two paths. Let P denote the segment of the first path between v and w and Q the corresponding segment of the other path. Let u_P denote the amount of flow on path P, and u_Q the amount of flow on path Q. Notice that we can substitute segment P for Q to obtain a different valid path to the root for client i, and vice versa for client j. We argue that at least one of these substitutions yields an improved solution, violating our assumption of optimality.

To see this, we evaluate the cost of our flow in 3 pieces: (i) the u_P units of flow on P, the u_Q units of flow on Q, and everything else. If we leave out the flow on P and Q, the remaining flow (which is not feasible, since it violates conservation at v and w) has some total amount of flow u_e on each edge e for some total fixed cost C.

To make this flow feasible we must add $u_{PQ} = u_P + u_Q$ units of flow from P to Q, but we can distribute this flow any way we like between P and Q. Let us consider the incremental cost over C of placing u'_P units of flow on P and u'_Q units on Q such that $u'_P + u'_Q = u_{PQ}$. The incremental cost on edge $e \in P$ of sending an additional u'_P units is $c_e(u_e + u'_P)$, which (since c_e is strictly concave) is a concave function of u'_P (offsets of concave functions are strictly concave). The overall incremental cost $f_P(u'_P)$ of sending u'_P units on P is thus a sum of strictly concave functions and thus strictly concave. Similarly, the overall incremental cost $f_Q(u'_Q)$ of sending u'_Q units of flow on Q is also strictly concave. Paths P and Q are disjoint, so our overall solution cost is simply $C + f_P(u'_P) + f_Q(u'_Q)$. We wish to minimize this subject to $u'_P + u'_Q = u_{PQ}$, and both positive which means minimizing $f_P(\lambda u_{PQ}) + f_Q((1 - \lambda)u'_P)$ over $0 \leq \lambda \leq 1$. But since f_P and f_Q are strictly concave functions, this sum is a strictly concave function of λ, so is optimized at one of the "endpoints" $\lambda = 0$ or 1, i.e. $u'_P = 0$ or $u'_P = u_{PQ}$.

This shows that we can find a solution better than the (presumed optimal) one which sent flow on both P and Q, a contradiction. $\square$

2.3. The shortest path tree heuristic

Now that we know that an optimal solution is a tree, a shortest path tree is a logical candidate for an approximate solution. Unfortunately, the shortest path tree can have polynomial approximation gap, as the following example illustrates.

Consider an unweighted $m \times m$ grid graph with a root vertex attached to the top m vertices by single edges as shown in Figure 1a. All edge lengths are 1.

Consider an instance in which only the m vertices on the bottom have clients, each having the same activation probability p. Let $n = m^2 + 1$ be the total number of vertices. The number of clients in this instance is then given by $N = m = \sqrt{n-1}$.

In a shortest path tree solution, every client will use a path that goes up along vertical edges, until it merges into the root (see Figure 1b). Let us compute the cost of such a solution. Consider a vertical path of length m. Each edge of this path is used with probability p, so the cost of this path is $p \cdot m$. Summing over all m vertical paths, the total cost of the shortest path solution is pm^2.

Now consider an alternative solution, in which all clients in the bottom row go to the middle vertex of the row, and then use 1 central vertical path to get to the root (illustrated in Figure 1c). The central vertical path costs at most m since the maximum cost of an edge is 1 and there are m edges in it. Similarly, the cost of the paths converging in the center of the bottom row is at most $m - 1$. Thus, the total cost of this solution is at most $2m$. The ratio of costs of the shortest path solution and the central path solution is at least $pm^2/2m = \Omega(p\sqrt{n-1}) = \Omega(n^{1/2})$ when p is constant. Thus, a shortest path tree can give a polynomially-bad approximate solution.

In a straightforward variation, using a $n^{1/3} \times n^{2/3}$ grid, one can show that even in the case of uniform probabilities $p_i = n^{-2/3}$, the shortest path tree solution can be $\Omega(n^{1/3})$ worse than optimum.

3. Tools for the Solution

We now know that we are seeking a tree solution to our problem. In this section, we introduce an approximate cost function that is easier to work with. We also outline a "hub and spoke" principle which will guide us in the search for a solution.

3.1. A price function

To develop our approximation algorithm, we convert our concave cost function into one that is piecewise linear but closely approximates our original function.

Given a solution, let us define the *unit cost* of an edge e in the solution as the probability that e becomes active, $c_e = 1 - \prod_{i \in U_e}(1 - p_i)$, where U_e is the set of clients whose paths to the root contain e. We can assume that $|U_e| > 0$, since we can always throw out edges that are not used by any of the clients. We can upper bound c_e by the sum of activation probabilities of the clients using e, $c_e \le \sum_{i \in U_e} p_i$. Notice that when this sum is small, c_e is very close to it, whereas for when the sum becomes large (greater than 1), c_e rapidly approaches 1. Based on this observation, let us introduce a *unit price* function

$$\hat{c}_e = \min \left\{ \sum_{i \in U_e} p_i, 1 \right\}$$

We have already argued that our unit price function is an upper bound on the actual unit cost. We show that it is a tight upper bound: that $\hat{c}_e$ is at most some constant times c_e, independent of probabilities p_i.

Lemma 3.1.

$$c_e \le \hat{c}_e \le \frac{1}{1 - 1/e} c_e.$$

Proof. We have already argued the first inequality. For the second inequality, let us fix $s \equiv \sum_{i \in U_e} p_i$. Observe first that since $1 - x \le e^{-x}$,

$$\prod_{i \in U_e}(1 - p_i) \le \prod_{i \in U_e} e^{-p_i} = e^{-s}.$$

So we must have that $c_e \ge 1 - e^{-s}$. Now we show that

$$\hat{c}_e \le \frac{1}{1 - 1/e}(1 - e^{-s})$$

which, combined with the previous observation, proves the lemma.

Case 1: $s \ge 1$, so $\hat{c}_e = 1$. The desired inequality holds trivially

$$\frac{\hat{c}_e}{1 - e^{-s}} \le \frac{1}{1 - 1/e}$$

Case 2: $s < 1$, so $\hat{c}_e = s$. Consider the quantity $\frac{s}{1 - e^{-s}}$. This is an increasing function of s on the interval $(0, 1]$, so the maximum is attained at $s = 1$. So once again,

$$\frac{\hat{c}_e}{1 - e^{-s}} \le \frac{1}{1 - 1/e}$$

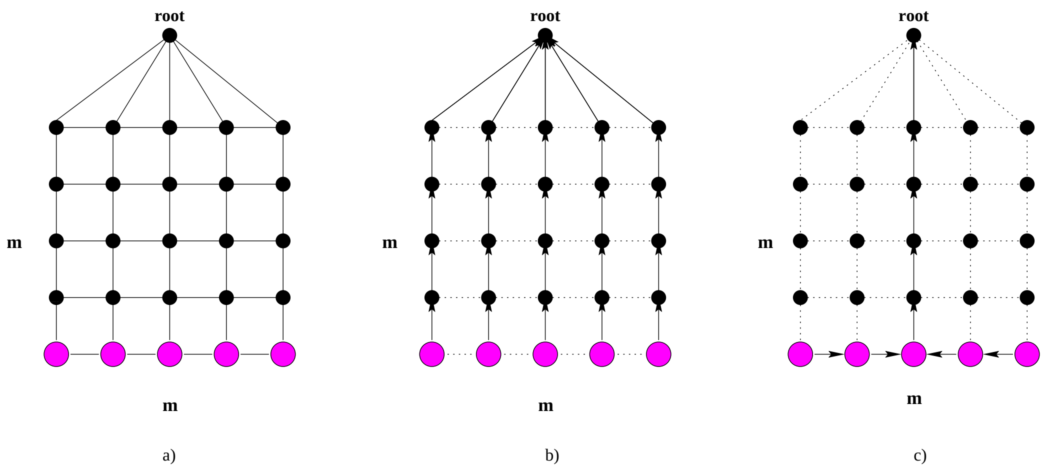

Figure 1. The shortest path example. a) $m \times m$ grid graph with a root. b) Shortest path tree solution. c) Alternative stem path solution.

$\square$

In a problem with edge lengths, the actual contribution of an edge to the objective function is its length multiplied by its unit cost, $l_e \cdot c_e$. Let us refer to this quantity as the weighted cost of an edge, or simply its cost. We can similarly define the (weighted) price of an edge to be $l_e \cdot \hat{c}_e$. Because of the linear relationship to unit price and cost, the weighted price of an edge is an upper bound on its actual weighted cost and their ratio is bounded by the same constant $1 - 1/e$.

Since costs and prices are related within a constant, we see that the optimum *cost* maybecast tree has cost within a constant factor of the optimum *price* maybecast tree. It follows immediately that any algorithm that approximates the minimum price tree to within a constant will also approximate the minimum cost tree to within a the same constant time $e/(e-1)$.

3.2. A hub and spoke model

What the grid graph example demonstrated was that sometimes instead of using a shortest path to the root, it pays for clients to cluster together at some node and then share one path to the root. To get a better understanding of why this is the case, let us go back to our flow analogy. Suppose each client i has to send p_i units of flow to the root via some path. Once the total amount of flow from clients using the same path reaches a certain value, adding more flow (from one more client) onto the path doesn't increase the real unit cost (the probability of the edges on the path being active) by much. This is captured by our unit price function which uniformly charges 1 for every edge once $\sum_{i \in U_e} p_i$, the amount of flow on that edge, reaches 1.

Consider any tree solution T to the maybecast problem. Every client has a unique path to the root. We will say that a client i *passes through* node v if node v lies on i's path to the root. As paths of different clients converge towards the root, they might start sharing the same edges and passing through the same nodes. Define a *hub* to be a node that has at least 1 unit of flow from clients passing through it. The unit price of every edge on the path from a hub to the root is 1. Thus, once a client gets to a hub, the rest of its path to the root is already paid for. Let $hub(i)$ be the first hub on the path of a client i to the root. We will say that client i is *assigned* to $hub(i)$. In building a solution, if we were given a set of hubs and an assignment of the clients to the hubs, to obtain the rest of the solution we would need to connect all of the hubs up to the root at the minimum edge cost. But edges on the paths from hubs to the root have price one, so the optimum "hub tree" is just a Steiner tree on the hubs (which we can approximate via a Steiner tree approximation algorithm).

Let us now analyze the cost of assigning a client to a hub. Any edge e on the path of i to $hub(i)$ carries no more than 1 unit of flow; otherwise we would have a hub sitting earlier on that path. This means that unit price of edge e is $\hat{c}_e = \sum_{i \in U_e} p_i$, where U_e is the set of clients using that edge. Thus, client i contributes p_i towards the unit price of every such edge. Let $h(i)$ denote the length of the path (distance) from client i to $hub(i)$. Then over all the edges on its path to a hub, client i contributes $p_i \cdot h(i)$ to the overall solution.

We can therefore decompose our problem into two parts: first, clustering clients at hubs, and second, connecting hubs up at minimum cost. Of course, we do not know where the hubs are, rather we have to create them. In this respect, the problem is similar to the uncapacitated facility location problem in which, given costs for opening facilities and assigning locations to them, one wants to open a set of facilities and connect each location to an open facility while minimizing net cost [7]. There is no direct cost associated with opening a hub, however the Steiner tree parallel applies only if there is a large number of clients assigned to every hub. So how can we find a set of hubs, so that there is a hub not too far from each client but every hub receives a substantial amount of demand?

4. A gathering problem

In the uncapacitated facility location problem [4, 5, 7, 9], we are given a set of facilities F, a set of locations C, a cost f_i for opening each facility $i \in F$, and for each pair $i \in F, j \in C$, the cost c_{ij} of connecting location j to (opened) facility i. The goal is to select a subset of facilities to open, and assign each location to an opened facility so as to minimize the total sum of the assignment costs and the costs of opening facilities. There are a number of constant-factor approximation algorithms for solving the uncapacitated facility location problem when the assignment costs satisfy the triangle inequality [4, 5, 7, 9]. The currently best approximation guarantee, due to Charikar and Guha [4], is 1.728. In a demand-weighted version of the problem, we are also given a non-negative demand d_j for each location j, so that the cost of assigning j to a facility i is $d_j c_{ij}$.

Intuitively, facility costs are supposed to encourage us to use facility resources frugally and assign many clients to the same facility. However, it is still possible to have an optimal solution in which a facility serves only very few clients. If we wanted to prevent opening a facility that serves too few clients, instead of having a fixed cost associated with each facility, we could require that a facility i can be opened only if at least a certain number of clients get assigned to i, or in the demand-weighted case, facility i gets to serve at least some minimum required demand.

Let us define an r-**gathering** problem to be a special type of demand-weighted facility location problem. The goal is to open a subset of facilities and assign demands to them, such that each opened facility has at least r units of demand assigned to it, and the total assignment cost is minimized. In our particular application, the cost of opening a facility is zero if that facility is feasible. However, the problem and our solution generalize to include facility-opening costs, and also to allow different lower bounds at different facilities.

We can formulate our problem of finding a set of hubs close to all the clients as an instance of the r-gathering problem. Take the set of all nodes of the graph to be the set of potential facilities, and the set of clients to be the set of locations. Set assignment costs to be the shortest path distances between nodes. Let the demand of a client j be p_j. If we solve the r-gathering problem on this instance with $r = 1$, the opened facilities can be thought of as hubs. Next we will show that there exists a 1-gathering solution of cost comparable to the cost of an optimum solution to our original problem.

The r-gathering problem also makes sense in the context of clustering problems: an r-gathering solution attemps to divide the clients into an arbitrary number of clusters (one per open facility) so as to minimize the average radius (distance from the cluster center) while forbidding any cluster from have few items.

4.1. Solving the r-gathering problem

We now describe an approximation algorithm for the r-gathering problem. For simplicity, we first consider the case without facility-opening costs, and assume that all lower bounds are the same value r. This is the problem we need to solve in our maybecast application. This simplified version can be described as follows: given a set of potential facilities (indexed by i), a set of of clients (indexed by j), each with a demand d_j, and a cost matrix c_{ij} satisfying the triangle inequality, we wish to open a set of facilities, and assign each client to an open facility, such that

1. At least r units of demand are assigned to each open facility

2. The total distance traveled by all the demand to its assigned facility is minimized.

More formally, we wish to set assignment variables x_{ij}, where $x_{ij} = 1$ if j is assigned to i and 0 otherwise, such that

1. For every i, $\sum_j x_{ij} \geq r$ (facility i is open) *or* $\sum_j x_{ij} = 0$ (facility i is not open).

2. We minimize $C = \sum_{i,j} x_{ij} c_{ij} d_j$.

We present a *bicriterion approximation* to this problem.

Theorem 4.1. *Let C^* be the optimum cost of a feasible solution to an r-gathering problem. In polynomial time, we can find a solution to the problem, of cost C, such that*

1. *For every i, $\sum_j x_{ij} \geq r/2$ (facility i is open)* or *$\sum_j x_{ij} = 0$ (facility i is not open).*

2. *$C = O(C^*)$.*

That is, we can find a solution that gathers at least half the required demand at every open facility, and does so at cost within a constant factor of the optimum gathering cost. There is a typical tradeoff: we can in fact come within $(1 -$

ϵ) of the required demand (nearer feasibility) at every open facility if we are willing to worsen the approximation ratio.

To prove this theorem, given our gathering problem, we define a related facility location problem and show that its solution meets our requirements. To define the facility location problem, we assign a *cost* f_i to each facility i. The cost of i is defined as twice the minimum cost of moving r units of demand from the clients to facility i.

More formally, let $j_1, j_2, \dots, j_n$ be the clients, ordered in increasing distance from facility i (that is, $c_{ij_1} \leq c_{ij_2} \leq \cdots$). Let k be minimum such that $d_{j_1} + d_{j_2} + \cdots d_{j_k} \geq r$. For simplicity let us assume this sum is exactly equal to r—if not, just split client k into two smaller demands. Then the cost

$$f_i = 2(c_{ij_1} d_{j_1} + \cdots + c_{ij_k} d_{j_k}).$$

Note that for any assignment that assigns at least r units of demand to i, we must have $\sum_j c_{ij} x_{ij} d_j \geq f_i/2$.

Let $\mathcal{F}$ denote the facility location problem with costs c_{ij} inherited from the gathering problem $\mathcal{G}$ and with facility costs f_i.

Lemma 4.2. *The cost of an optimum solution to $\mathcal{F}$ is at most thrice that of $\mathcal{G}$.*

Proof. Consider any solution to $\mathcal{G}$. It opens certain facilities and makes assignments x_{ij}. The same solution is clearly feasible for $\mathcal{F}$; let us analyze its cost under $\mathcal{F}$. The assignment cost $\sum x_{ij} c_{ij} d_j$ is the same for both problems; we need only measure the added cost of opening the facilities in $\mathcal{F}$.

Consider some facility i that was opened in $\mathcal{G}$. By feasibility of the solution for $\mathcal{G}$, there are at least r units of demand incident on i. It follows that the total cost of shipping this demand to i, $\sum_j x_{ij} c_{ij} d_j \geq f_i/2$. Thus, summing over all open i, we have

$$\sum_{i \text{ open}} f_i \leq 2 \sum_{i \text{ open}} \sum_j x_{ij} c_{ij} d_j,$$

that is, the total cost of opening facilities is at most twice the total assignment cost. But that total assignment cost is the entire cost of $\mathcal{G}$, so the cost of $\mathcal{F}$ is at most 3 times the cost of $\mathcal{G}$. □

Lemma 4.3. *Any solution to $\mathcal{F}$ can be converted into a solution of no greater cost that assigns at least $r/2$ units of demand to every open facility.*

Proof. Suppose we have a solution to $\mathcal{F}$ that does not satisfy the gathering requirement. We give a procedure that modifies the solution to a cheaper one by closing a facility. We repeat this procedure until the gathering requirement is satisfied. Clearly this will happen within n repetitions.

As a first step, convert the solution to one that is *locally optimal:* assign every client to the nearest open facility. Clearly this can only improve the cost.

Now suppose there is an open facility i with less than $r/2$ incident demand. Consider the nearest clients $j_1, \dots, j_k$ that are used to define f_i. By definition, these clients in total have $d_{j_1} + d_{j_2} + \cdots + d_{j_k} = r$ units of demand, and send at most $r/2$ units to i, so at least $r/2$ of this demand must be assigned to other facilities. Let

$$c = \frac{1}{r}(d_{j_1} c_{ij_1} + d_{j_2} c_{ij_2} + \cdots + d_{j_k} c_{ij_k}) = f_i/2r$$

be the *average distance* of these units of nearby demand. By Markov's inequality, less than half of this nearby demand is at distance exceeding $2c$ from i. But by assumption less than $r/2$ of these nearby demand units are assigned to i. Thus, some client j' *not* assigned to i is at distance less than $2c$ from i. But by local optimality, this j' must be assigned to a facility i' at distance less than $2c$ from j'. By the triangle inequality, this facility is at distance less than $4c$ from i.

So suppose that we close facility i, take all the demand assigned to i, and assign it to i' instead. By the triangle inequality, this adds at most $4c$ to the assignment distance of those units of demand; thus the total increase in assignment cost is the amount of demand at i (less than $r/2$) times the added distance (at most $4c$) for a total of less than $2rc$. But now note that $c = f_i/2r$, meaning that the change in assignment cost is less than f_i.

Thus, by closing a facility, we have saved f_i in facility cost, and paid less than f_i in assignment cost. We can repeat this process until no facility remains with less than $r/2$ units of incident demand. □

The conversion outlined in this lemma is clearly algorithmic. This leads to the following result:

Corollary 4.4. *Given a ρ-approximation algorithm for the facility location problem, and given an r-gathering problem, we can find a solution of cost at most 3ρ times the optimum gathering cost that gathers at least $r/2$ units of demand at every open facility.*

Proof. Given the gathering problem $\mathcal{G}$ with cost G, define the related facility location problem as above, which has cost at most $3\mathcal{G}$. Run the ρ-approximation algorithm to find a solution of cost at most $3\rho G$. To that solution apply the conversion that ensures that every facility has at least $r/2$ incident demand at no greater cost. □

Applying current approximation bounds for facility location yields our initial theorem:

Corollary 4.5. *There is a bicriterion approximation algorithm for the r-gathering problem that gives an $r/2$-gather costing at most 5.184 times the optimum r-gathering.*

Proof. Use the 1.728-approximation algorithm for facility location [4]. □

4.2. Generalizations

Our approximation algorithm straightforwardly handles two generalizations (which are not needed for our application):

- We can tradeoff between the gathering cost and the nearness to feasibility (gathering lower bounds).
- We can include an actual facility opening cost f'_i.
- We can allow the lower bound of demand needed to open each facility to be a distinct value r_i.

There is a tradeoff between the approximation factor and the factor by which the lower bound on demand gets relaxed. By scaling our special facility cost f_i appropriately we can demonstrate the following result.

Corollary 4.6. *There is a bicriterion approximation algorithm for the r-gathering problem that gives an $r\epsilon$-gather costing at most $\frac{1+\epsilon}{1-\epsilon} \cdot 1.728$ times the optimum r-gathering.*

To handle nonzero facility costs, we simply add our special facility cost f_i (minumum cost to ship r units of demand to i) to the original facility opening cost f'_i to get the facility opening cost $f'_i + f_i$ in our derived facility location problem $\mathcal{F}$. Our analysis still shows that $\mathcal{F}$ has cost within a constant factor of $\mathcal{G}$. Furthermore, as in the simpler problem, the added cost f_i is enough to pay for closing any facility with less than $r/2$ demand and rerouting to a different open facility. Note that in fact, it suffices to set the derived problem's facility cost to be $\max(f_i, f'_i)$; this might improve performance in practice.

To handle distinct lower bounds, we note that our analysis is essentially local to every vertex. If we define f_i to be the minimum cost of shipping r_i units of demand to facility i, the entire analysis goes through unchanged. We end up with a constant factor approximation that places at least $r_i/2$ units of demand on facility i if it is open.

5. Gathering for maybecast

In this section, we use our gathering algorithm as a black box for solving the maybecast problem. We apply the following algorithm:

Our algorithm clusters clients into large enough groups and then connects up all the groups with the root. For clustering purposes, we will think of a client i as having p_i units of demand and take assignment costs to be equal to shortest path distances. We solve an r-gathering problem on this instance using our bicriterion approximation algorithm and then build a Steiner tree on all of the facilities opened by the r-gathering algorithm, thus connecting each client with the root via its gathering point. In the last step we simplify the paths of all clients, clearly without increasing the cost of solution.

MAYBECAST ALGORITHM

1. Let $F = V$, $C = S_c$. For any pair $i \in F, j \in C$, let c_{ij} = shortest path distance between i and j. For any client $j \in S_c$, let $d_j = p_j$. Set $r = 1$. Run our bicriterion approximation algorithm on this instance of the r-gathering problem to obtain a feasible solution to an $r/2$-gathering problem.
2. Let $H \subset F$ be the set of facilities opened in Step 1. Build an approximately minimum cost Steiner tree T_S connecting nodes in H to the root r.
3. For a client j, its path P_j to the root consists of its path to the facility i to which it was assigned in Step 1 plus the path from i to the root in the tree T_S.
4. If there are cycles in P_j, remove them.

Figure 2. The Maybecast Algorithm.

We analyze this algorithm's performance in two steps. First, we show that the derived gathering problem has an optimum cost close to that of the maybecast optimum. Then we show that, given a solution to the derived gathering problem, we can convert it to a maybecast solution of similar cost. Combining these two arguments with our approximation algorithm for gathering shows that the maybecast algorithm finds a good solution.

5.1. Cost of the derived gathering problem

In this section we analyze the cost of the derived gathering problem.

Theorem 5.1. *Given an instance $\mathcal{I}$ of the maybecast problem with an optimal solution of cost OPT, there exists a solution to the derived r-gathering problem ($r = 1$) of cost $O(OPT)$.*

To prove this theorem, we will take an optimal maybecast tree for instance $\mathcal{I}$ and construct an r-gathering solution using the structure of the tree. In the optimal maybecast solution, the path from each client eventually reaches a hub. By the definition of a hub, an ancestor of a hub is also a hub, which means that the hubs form a subtree T_H of the maybecast tree (containing the root) which we will call the *hub tree*. Demand from any one client moves on non-hub edges until it reaches a hub, and then moves on the hub tree to the root.

Consider the price $\hat{C}^* = O(OPT)$ of the optimum cost maybecast solution. Let us decompose the price paid for the overall solution into the price of moving demand from each client to its first hub in the hub tree (along non-hub edges) and the price of moving demand from the hubs to the root. Along non-hub edges, the price function is linear: client i sending demand p_i along non-hub-edge e contributes $l_e p_i$ to the price function $\hat{c}_e$. So if we define $h(i)$ to be the distance (under l_e) from client i to the first hub on its path to the root, then client i contributes $p_i h(i)$ in total to the price on non-hub edges. On the other hand, by definition edges in the hub tree are carrying demand exceeding 1. Thus the price of each edge is just the length of that edge. So $\hat{C}^*_H$, the price of edges in T_H, is just the total length of hub-tree edges. We have thus argued that

$$\hat{C}^* = \sum_{\text{clients } i} p_i h(i) + \hat{C}^*_H.$$

We use this two-part decomposition to construct a solution to the gathering problem. In the first step of the solution, we move the demand from each client to its first hub, exactly as in the optimum solution. This costs us exactly the first term, $\sum p_i h(i)$. We then gather all this shifted demand from the hubs into facilities, each holding at least r demand in total, at cost $\hat{C}^*_H$. This provides a solution to the gathering problem whose cost is the sum of the two parts, namely $\sum p_i h(i) + \hat{C}^*_H$. This is precisely the price of the maybecast solution, which in turn is within a constant factor of $e/(e-1)$ times the true cost of the maybecast solution.

That the first part costs $\sum p_i h(i)$ is immediate from the definition of the (linear) gathering problem objective function. It remains to prove that we can gather the demand from where it arrives at the hubs at cost $\hat{C}^*_H$.

Lemma 5.2. *Given any tree with edge costs $c_e \geq 0$, demands $d_i \geq 0$ on nodes, and total demand at least r, there exists a solution to the demand-weighted r-gathering problem of cost no more than $r \cdot C$, where C is the total edge cost of T.*

Proof. By induction on the size of the tree. Suppose first that there is a leaf (degree-one node) with less than r units of demand. Move this demand from the leaf to its parent along edge e. Since we are moving less than r units this costs at most rc_e. Then we can delete edge e and the leaf. This leaves a tree with total edge cost $C - c_e$ and demand which, by induction, can be gathered at cost $r(C - c_e)$. Thus, the total gathering cost is at most $rc_e + r(C - c_e) \leq rC$ as desired.

Now suppose every leaf has r units of demand. Take one leaf and open a facility there. Assign all demand on the leaf to the facility. This has no cost. Then delete the leaf and its demand. Since all (unrooted) trees have at least two leaves, the remaining tree still has demand greater than r so, again by induction, it can be gathered at cost rC. □

Theorem 5.1 implies that the cost of an optimal r-gathering solution is $O(OPT)$. The $r/2$-gathering solution obtained with our bicriterion approximation algorithm must then cost $O(OPT)$ as well.

Corollary 5.3. *The $r\epsilon$-gathering solution ($0 < \epsilon < 1$) produced by our bicriterion approximation algorithm costs at most*

$$\frac{1.728}{1 - 1/e} \cdot \frac{1+\epsilon}{1-\epsilon} \cdot OPT$$

5.2. Cost of the Steiner Tree

The previous section showed that the cost of our gathering solution is $O(OPT)$. In this section, we show that the cost of the Steiner tree we build on the gathering points is also $O(OPT)$.

Theorem 5.4. *The cost of the minimum Steiner tree on the gathering points is $O(OPT)$.*

Proof. Consider the following new maybecast instance, based on our prior solution to the $r/2$-gathering problem with $r = 1$. Move each client to the gathering point to which it was assigned in the $1/2$-gathering solution. Since each gathering point has at least $1/2$ unit of demand, we will refer to it as a *half-hub*.

Let us construct a solution to this new maybecast instance. We will first send all the demand back to its original nodes (via shortest paths), and from there route them to the root using an optimal solution to the original maybecast problem at cost OPT. Notice that the price (which is an upper bound on the cost) of sending demand to its original nodes is equal to the cost of gathering it, which is precisely $cost(r/2\text{-gathering})$. Let us denote by OPT' the cost of an optimum solution to this modified instance. Thus,

$$OPT' \leq cost(r/2\text{-gathering}) + OPT = O(OPT)$$

As has been already demonstrated in Theorem 2.1, it is actually suboptimal to send clients from the same node on different paths to the root. In an optimal solution (to the modified instance) all of the clients from a given node will use the same path to the root. Because we gathered at least $1/2$ unit of demand at each gathering point, each path will have at least $1/2$ unit of demand, so every edge on any path will carry at least $1/2$ unit of demand. So the unit cost of each edge on the path from a half-hub to the root in this optimal solution is at least

$$1 - \prod_{i \in h}(1 - p_i) \geq 1 - e^{-\sum_{i \in h} p_i} \geq 1 - e^{-1/2}.$$

It follows that the cost of every edge we use is at least $1 - e^{-1/2}$ times its length, and that the cost of the optimum solution is at least $1 - e^{-1/2}$ times the sum of the lengths of edges used.

A Steiner Tree on the gathering points connects all the half-hubs to the root while minimizing the total edge cost (length). Hence, $(1 - e^{-1/2}) \cdot cost(\text{Steiner})$ is the lower bound on the OPT'. Thus we have

$$cost(\text{Steiner}) \leq OPT' / \left(1 - e^{-1/2}\right) = O(OPT)$$

□

We can use a known 1.55-approximation algorithm [8] to build a Steiner tree on the set of facilities opened by our bicriterion gathering algorithm. By the above theorem it is guaranteed to cost $O(OPT)$.

Corollary 5.5. *The Steiner tree on the $r\epsilon$-gathering points obtained with our bicriterion approximation algorithm costs at most*

$$\frac{cost(r\epsilon\text{-}gathering) + OPT}{1 - e^{-\epsilon}} \cdot 1.55$$

Notice that we can decrease the cost of the Steiner tree by picking an appropriate demand relaxation factor ϵ.

5.3. Performance of the approximation algorithm

In this section, we show that our algorithm finds a constant factor approximation to the optimum maybecast solution.

First let us verify that we don't increase the cost of the solution by performing Step 4. Consider some edge e and the set of clients U_e using it to get to the root before Step 4. Notice than when we simplify the clients' paths in Step 4, we don't add any clients to e, but we might remove some. This means that the cost of e can only decrease after Step 4. Thus, the cost of solution T obtained before we simplified the paths is an upper bound on the cost of the final solution.

To evaluate the performance of the algorithm, we measure the price of the solution obtained. The price paid can be decomposed into two components: (i) the price of moving demand from each client to its gathering point, and (ii) the price of moving demand from all the gathering points to the root. The first price component (i) is precisely the cost of the gathering problem, which we already proved $O(OPT)$. The second one is equal to the total price of flow on Steiner tree edges, which is upper bounded by the total length of the Steiner tree edges, i.e. the cost of the Steiner tree. We already showed this cost is $O(OPT)$. Thus, our total solution cost is $O(OPT) + O(OPT) = O(OPT)$ as claimed.

We have thus shown that the price of the solution T is at most $O(OPT)$. Since the price function is an upper bound on the true cost, we have the following result.

Theorem 5.6. *The* MAYBECAST ALGORITHM *yields an $O(1)$-approximation solution to the maybecast problem.*

We can rebalance the tradeoff between the cost of $r \cdot \epsilon$-gathering solution and the cost of the Steiner tree built on the corresponding gathering points. By setting the demand relaxation factor $\epsilon = 0.357$, we can obtain a solution to the maybecast problem which is within a factor of 40.7 of optimal.

Corollary 5.7. *There is an approximation algorithm for the maybecast problem that produces a solution of cost at most 41 times the optimum cost.*

5.4. A local optimization

In our algorithm above, we set the path of every client to go via the hub to which it was gathered in the derived gathering problem. This can result in a solution which is not a tree (paths can cross). We can fix this problem, without making the solution worse, if after finding the Steiner tree on gather points, we simply assign route each client, via a shortest path, to the closest point in the steiner tree. Equivalently, we can imagine contracting the Steiner tree into the root, and building a shortest path tree on what remains. This will clearly result in a tree solution to the maybecast problem.

To see that the cost of our modified solution is no worse than before, note that our analysis above assumes that every edge in the Steiner tree is saturated, with unit price (and cost) 1. Thus, no matter how we reroute the client paths, we will never pay more than we thought on the Steiner tree edges. As for the remaining edges, we bounded their unit cost by the total demand ($\sum p_i$) of paths through the edge, thus bounding the total cost on those edges by the sum of path lengths from each client to the Steiner tree. It follows that minimizing the sum of path lengths to the Steiner tree can only improve our solution, and that is precisely what taking shortest paths to the Steiner tree does.

6. Conclusion

We have studied a particular routing problem in which certain global information is hard to gather, and developed a local-decision approximation algorithm that achieves results within constant of optimum. Our solution is studying a "concave cost" network design problem. Perhaps the techniques used could be applied as well to more general cost functions or the design of more complex networks. In particular, it might lead to more powerful results for the buy at bulk network design problem, or for general concave-cost flow problems with capacities.

We might also ask whether our solution can get by with less global information. At present we require global knowledge of the set of clients and activation probability p. Is it possible to define a path-based scheme which works *regardless* of p? Under such a model, all we are requiring is that every equally sized subset of clients be equally likely to activate—a plausible assumption for non-geographically-dependent requests. Even better would be to define a scheme in which our path solution is always competitive against the best possible solution *for the set of active terminals*. But we suspect there are strong lower bounds for this variant.

More generally, we might consider other problems in which only part of the solution can be preplanned, while the remainder must be generated quickly and locally.

7. Acknowledgements

This problem arose from discussions between John Guttag, Ulana Lezegda, and Tom Leighton.

References

[1] M. Andrews and L. Zhang. The access network design problem. 1999.

[2] B. Awerbuch and Y. Azar. Buy-at-bulk network design. In *Proc. 38th Symposium on Foundations of Computer Science*, pages 542–547, 1997.

[3] B. Awerbuch, Y. Azar, and Y. Bartal. On-line generalized steiner problem. In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 68–74, New York/Philadelphia, Jan. 28–30 1996. ACM/SIAM.

[4] M. Charikar and S. Guha. A constant-factor approximation algorithm for the k-median problem. In *Proc. 40th Symposium on Foundations of Computer Science*, pages 378–388, 1999.

[5] F. Chudak and D. Shmoys. Improved approximation algorithms for the uncapacitated facility location problem. 1998.

[6] S. Guha, A. Meyerson, and K. Munagala. Hierarchical placement and network design problems. In *Proc. 41st Annual Symposium on Foundations of Computer Science*, 2000.

[7] K. Jain and V. V. Vazirani. Primal-dual approximation algorithms for metric facility location and k-median problems. In *Proc. 40th Symposium on Foundations of Computer Science*, pages 2–13, 1999.

[8] G. Robins and A. Zelikovsky. Improved steiner tree approximation in graphs. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 770–779, 2000.

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