A Constant-Factor Approximation Algorithm for the Multicommodity Rent-or-Buy Problem

Amit Kumar [*] Anupam Gupta[†] Tim Roughgarden[‡]

Abstract

We present the first constant-factor approximation algorithm for network design with multiple commodities and economies of scale. We consider the rent-or-buy *problem, a type of multicommodity buy-at-bulk network design in which there are two ways to install capacity on any given edge. Capacity can be* rented, *with cost incurred on a per-unit of capacity basis, or* bought, *which allows unlimited use after payment of a large fixed cost. Given a graph and a set of source-sink pairs, we seek a minimum-cost way of installing sufficient capacity on edges so that a prescribed amount of flow can be sent simultaneously from each source to the corresponding sink.*

Recent work on buy-at-bulk network design has concentrated on the special case where all sinks are identical; existing constant-factor approximation algorithms for this special case make crucial use of the assumption that all commodities ship flow to the same sink vertex and do not obviously extend to the multicommodity rent-or-buy problem. Prior to our work, the best heuristics for the multicommodity rent-or-buy problem achieved only logarithmic performance guarantees and relied on the machinery of relaxed metrical task systems or of metric embeddings. By contrast, we solve the network design problem directly via a novel primal-dual algorithm.

1 Introduction

We consider the problem of network design with multiple commodities and economies of scale. More precisely, given an undirected graph $G = (V, E)$ and a set $\mathcal{D} = \{(s_1, t_1), \ldots, (s_p, t_p)\}$ of vertex pairs called *demands*, we seek a minimum-cost way of installing sufficient capacity on the edges E so that a prescribed amount of flow can be sent simultaneously from each source s_k to the corresponding sink t_k. We are interested in the scenario where the cost of installing capacity exhibits economies of scale, in the sense that buying a large amount of capacity on a single edge results in a high capacity-to-cost ratio (i.e., good "bang for your buck"). Put differently, the cost of capacity is a concave function of the capacity bought.

The general problem described above goes by the name *buy-at-bulk network design*, and was introduced by Salman et al. [23]. The problem is NP-hard [23], and researchers have therefore sought out good approximation algorithms for the problem. The best algorithm currently known for the general problem is due to Awerbuch and Azar [3], who give an $O(\log n \log \log n)$-approximation based on Bartal's method for probabilistically embedding general metrics into tree metrics [5], where n is the number of nodes in the network. Improvements on the algorithm of [3] have been elusive, leading researchers to consider special cases of the problem. The *single-sink* version of buy-at-bulk network design, where all sinks t_k are identical, has recently received much attention. Andrews and Zhang [2] designed an $O(K^2)$-approximation algorithm for the single-sink problem when the cost of installing capacity is a restricted type of concave piecewise linear function with K breakpoints. This problem is called *access network design* in [2]. A constant-factor approximation algorithm for the access network design problem was later given by Guha et al. [12]. Subsequently and independently, Garg et al. [7] gave an $O(K)$-approximation algorithm and Guha et al. [13] designed a constant-factor approximation algorithm for the general single-sink buy-at-bulk network design problem (with an arbitrary concave, piecewise linear function describing the cost of installing a given amount of capacity). The constant of [13] has recently been improved upon by Talwar [25].

Despite these recent successes for the single-sink problem, there have been few improvements over the algorithm of Awerbuch and Azar [3] for any nontrivial version of *mul-*

[*]Lucent Bell Labs, 600 Mountain Avenue, Murray Hill NJ 07974. This work was done while the author was at Cornell University, Ithaca NY 14853, and supported by NSF ITR/IM Grant IIS-0081334 and the ONR Young Investigator Award of Jon Kleinberg. Email: amitk@research.bell-labs.com.

[†]Lucent Bell Labs, 600 Mountain Avenue, Murray Hill NJ 07974. Email: anupamg@research.bell-labs.com.

[‡]Department of Computer Science, Cornell University, Ithaca NY 14853. Supported by an NSF Graduate Fellowship, a Cornell University Fellowship, and ONR grant N00014-98-1-0589. Email: timr@cs.cornell.edu.

ticommodity buy-at-bulk network design. In this paper, we present the first constant-factor approximation algorithm for such a problem. As all of the techniques employed in recent papers [2, 7, 12, 13, 17, 25] make crucial use of the assumption that all commodities ship flow to the same sink vertex and do not obviously extend to the multicommodity setting, our algorithm and analysis require several new ideas. We also avoid reliance on metric embedding techniques (unlike [3]), instead attacking the network design problem directly via a novel primal-dual algorithm.

The Rent-or-Buy Problem. In this paper, we consider the *rent-or-buy* problem, a type of multicommodity buy-at-bulk network design with a simple function describing the cost of installing capacity. In the rent-or-buy problem, there are two ways to install capacity on any given edge. Capacity can be *rented*, with cost incurred on a per-unit of capacity basis, or *bought*, which allows unlimited use after payment of a large fixed cost. We model this scenario with positive parameters μ and M, with the cost of renting capacity equal to μ times the capacity required (per unit length), and the cost of buying capacity equal to M (per unit length). There is no loss of generality in assuming that $\mu = 1$. The multicommodity rent-or-buy problem was previously studied in an online setting by Awerbuch et al. [4] (where it was called the *network connectivity leasing problem*) and Bartal et al. [6], who used the framework of relaxed metrical task systems to give $O(\log^2 n)$- and $O(\log n)$-competitive algorithms for the problem, respectively.

Buy-at-bulk network design was originally defined in terms of installing *cables* on edges, with different cable types offering different amounts of capacity and carrying different costs [3, 23]. Andrews and Zhang [2] showed that this problem can be rephrased (with a loss of a small constant factor in the approximation ratio) with each cable type carrying a *fixed cost* (which must be paid irrespective of the capacity needed) and an *incremental cost* (which is paid for each unit of capacity required). The rent-or-buy problem therefore corresponds to the special case of one cable type with an incremental cost but no fixed cost, and one cable type with a fixed cost but no incremental cost.

We believe the rent-or-buy problem captures much of the essence of buy-at-bulk network design. Most of the difficulty of network design problems in which capacities obey economies of scale stems from the following tension: on the one hand, we would like to route flow between a source and sink on an (approximately) shortest path; on the other, we would like to gather flow from many different commodities together in order to purchase large quantities of capacity and take advantage of economies of scale. This issue of "route vs. gather" is clearly present in the rent-or-buy problem, and we believe that overcoming the difficulties caused by multiple commodities in this simple setting will lead to further progress on the general multicommodity buy-at-bulk network design problem.

Application to Maybecast. In addition to being a nontrivial special case of buy-at-bulk network design, the rent-or-buy problem arises in important applications. For example, Karger and Minkoff [17] introduced the so-called *maybecast* problem, defined as follows. There is an underlying undirected network G, with a source vertex s from which a multicast transmission will emanate, and a set D of demand vertices that wish to receive the transmission. The problem of building the min-cost network that connects the source to all of the demands is the classical min-cost Steiner tree problem. Karger and Minkoff [17] proposed a probabilistic version of this problem: each demand vertex i contacts the source s independently with probability p_i. Relative to a fixed Steiner tree on $\{s\} \cup D$, when a demand contacts the source s, all edges on the path joining it to s are said to become *active*. The goal is then to build the Steiner tree that minimizes the expected cost of the active edges.

Our solution to the rent-or-buy problem provides a constant factor approximation for the following *multicommodity* version of the maybecast problem. Instead of a single source s, we are given a set of sources S. Each demand wants to receive data from one source in S, and it contacts that source with some probability. As in the previous problem, we seek paths between the demands and the sources they wish to contact so that the expected number of active edges is minimized. This problem reduces, modulo a small constant factor in the approximation ratio, to rent-or-buy network design (see [17]).

Application to Connected Facility Location. Our results also give a constant-factor approximation algorithm for a multicommodity version of *connected facility location*, a problem that has recently received attention in both the operations research literature [19, 20, 21] and the computer science community [14, 17, 18]. In the previously studied version of the connected facility location problem, the input is a set F of *facilities*, a set D of *demands*, a graph $G = (V, E)$ with $V = F \cup D$ and costs c_e on edges e, and a parameter $M > 1$. A solution consists of an assignment of demands to facilities and a subgraph of G spanning the open facilities (a Steiner tree). If demand j is assigned to facility $i(j)$ and the length of the shortest path between them in G (w.r.t. c) is $d(j, i(j))$, then the cost of a solution is $\sum_{j \in D} d(j, i(j)) + M \sum_{e \in T} c_e$ (where T is the Steiner tree spanning the open facilities). The first constant-factor approximation algorithm for this problem was given by Karger and Minkoff [17], and Gupta et al. [14] subsequently gave an algorithm with improved performance guarantee. Very recently, Swamy and Kumar [24] obtained a 5-approximation algorithm for this problem.

In the multicommodity version of connected facility location, we are in addition given several *commodities*. Each demand belongs to one of these commodities. We again open facilities and assign demands to them, but now require only a subgraph T such that, for any commodity k, the set of facilities serving demands of commodity k are connected. In solving the rent-or-buy problem, we develop techniques that also give a constant-factor performance guarantee for the multicommodity connected facility location problem.

New Techniques for Primal-Dual Approximation Algorithms. Our algorithm is based on the primal-dual method. The high-level idea of this method is to consider an integer programming formulation of our network design problem and the dual of its linear programming relaxation, and to iteratively construct both an integral primal solution (i.e., a feasible network) and a feasible dual solution proving that the network has near-optimal cost.

The first systematic application of the primal-dual method was to a large class of network design problems; see [11, 26] for a survey of this and earlier work. More recently, Jain and Vazirani [16] gave primal-dual approximation algorithms for several facility location problems that could not be solved using earlier techniques. Our algorithm is at times reminiscent to the facility location algorithms of [16] (reflecting our need to cluster demands together to leverage economies of scale) and to the network design algorithms described in [11] (as clustered demands must then be connected cheaply, as in canonical network design problems). However, these two implementations of the primal-dual method are not easily combined, and we require further ideas to obtain a good approximation algorithm for the rent-or-buy problem. In particular, we contribute two new techniques to existing primal-dual technology that we believe may find other applications.

First, we introduce *geometric scaling* in a primal-dual context. We use scaling to break up the execution of our algorithm into successive stages in a way that ensures that the "mistakes" made in any given stage have little significance for future stages. While other primal-dual algorithms have been used as a black-box within a scaling procedure [1, 8, 9, 27], we use scaling *inside* our primal-dual algorithm to control the rate of increase of dual variables.

Second, unlike most previous primal-dual approximation algorithms, we do not explicitly maintain feasibility of our dual solution. Rather, we maintain feasibility with respect to a strict subset of the dual constraints, and prove that we are always *approximately* feasible for the full LP. This idea is similar in spirit to recent "dual fitting" approaches to facility location problems [15, 22]. Freed from the need to maintain dual feasibility, we can make use of an unusually aggressive dual increase step; this in turn allows us to more easily argue that the cost of our solution is close to the objective function value of our (approximately feasible) dual solution.

2 Preliminaries

An instance of multicommodity rent-or-buy network design (MROB) is specified by an undirected graph $G = (V, E)$, a nonnegative cost c_e for each edge e, a set $\mathcal{D} = \{(s_1, t_1), \ldots, (s_p, t_p)\}$ of pairs of demands, and a parameter $M > 1$. We will abuse notation and write $j \in \mathcal{D}$ for a generic demand j of the form s_k or t_k. We assume for simplicity that the source s_k wishes to send one unit of flow to the sink t_k, but our algorithm and analysis extend without difficulty to non-uniform flow requirements (details omitted from this abstract). By $d(u, v)$ we mean the length of the shortest path[1] in G between vertices u and v, with respect to edge lengths c.

A solution to an MROB instance is specified by an assignment of each demand pair $(s_k, t_k) \in \mathcal{D}$ to an s_k-t_k path of G. If a_e paths use edge e, then the cost of this solution is defined by $\sum_{e \in E} c_e \min\{a_e, M\}$. The term $c_e a_e$ corresponds to renting capacity on edge e, and the term $c_e M$ corresponds to buying capacity on e. We seek a solution of minimum cost.

2.1 Reformulation as Connected Facility Location

We begin with a reduction from MROB to multicommodity connected facility location (MCFL). We will see shortly that the latter problem admits a relatively simple integer programming formulation, thereby allowing us to use the primal-dual method.

The precise problem that we reduce to is the following. The input is an undirected graph $G = (V, E)$ with edge e possessing cost c_e, a set $\mathcal{D} = \{(s_1, t_1), \ldots, (s_p, t_p)\}$ of vertex pairs, and a parameter $M > 1$. A solution consists of a set $F \subseteq V$ of facilities to open, an assignment of sources and sinks to open facilities, and a subgraph (V, H) of G with the following property: if for some k, s_k is assigned to facility i_1 and t_k to i_2, then there is a path in (V, H) between i_1 and i_2. The cost of a solution is $\sum_{j \in \mathcal{D}} d(j, i(j)) + M \sum_{e \in H} c_e$, where $i(j)$ is the facility to which the demand j is assigned and d is again shortest-path distance in G (with respect to c). (The seemingly more general statement of MCFL in Section 1 can also be reduced to the one above.) We then have the following reduction.

Lemma 2.1 *A β-approximation algorithm for MCFL gives a 2β-approximation algorithm for MROB.*

Proof: An instance of MROB naturally defines an instance of MCFL with the same parameters (G, c, $\mathcal{D}$, and M). We will map every solution of the latter problem to one of the former with equal cost, and an optimal solution to the former problem to one of the latter with at most twice the cost.

[1]Throughout this paper, we assume some arbitrary but fixed tie-breaking mechanism that ensures uniqueness of shortest paths.

Given a solution to the induced MCFL instance, define an MROB solution as follows. The s_k-t_k path is defined to be the shortest path from s_k to $i(s_k)$ and from $i(t_k)$ to t_k, connected by a path from $i(s_k)$ to $i(t_k)$ in H, the subgraph of edges chosen in the facility location solution (which exists by feasibility for MCFL). This solution to the MROB instance has cost bounded above by the MCFL solution.

Consider an optimal solution $P_1^*, \ldots, P_p^*$ to an MROB instance. Let H^* denote the edges used by M or more paths. We cannot simply reverse the mapping of the previous paragraph, since there is no guarantee that the subpaths of P_k^* from s_k to H^* and from t_k to H^* terminate in a common component of H^*. Instead, initialize H (the connecting edges in our facility location solution) to be H^* and F (the open facilities) to be the vertices spanned by H^*; we will supplement these sets with further edges and vertices shortly. Define $\{H_i\}$ to be the components of (V, H) with isolated vertices discarded; this set is initially just the non-trivial components of (V, H^*), but will change as we add further edges to H.

Call a demand pair (s_k, t_k) *good* if path P_k^* is edge-disjoint from all but at most one H_i, and *bad* otherwise. If P_k^* is edge-disjoint from all H_i's, then add vertex s_k to F and assign both s_k and t_k to it. If P_k^* intersects only H_i, then assign each of s_k and t_k to their nearest neighbors in H_i. As long as there is a bad pair, we execute the following procedure.

Let H_i be the component of minimum index that intersects some bad demand demand pair, say (s_k, t_k). Call H_i the *current component*. Let P_k^1 denote the edges of $P_k^* \setminus H$, and P_k^2 the edges of $P_k^* \cap H$ that lie outside of H_i (in components with larger index). Our analysis breaks into two cases. Let $c(P)$ denote $\sum_{e \in P} c_e$ for a subgraph P.

Case 1: Suppose $c(P_k^1) \geq c(P_k^2)$. In this case we assign each of s_k and t_k their nearest neighbors in H_i, and redefine the demand pair (s_k, t_k) to be good.

Case 2: Suppose $c(P_k^1) < c(P_k^2)$. In this case we add all edges of P_k^1 to H, and add all endpoints of these edges to F. Since (s_k, t_k) is bad, this addition causes two or more components (H_i and components of higher index) to merge into a single component; the new component retains the index i. Any demand pair (s_q, t_q) whose path is now edge-disjoint from all non-trivial components of (V, H) except H_i (such as (s_k, t_k)) is redefined to be good, and s_q and t_q are assigned to their nearest neighbors in H_i.

Each iteration of the above procedure strictly increases the number of good demand pairs and maintains the invariant that all good demand pairs have been assigned to open facilities in a common component of (V, H). The procedure therefore terminates with a feasible solution to the MCFL instance; it remains to show that this solution has small cost.

We first claim that assignment costs of our solution are at most $2\sum_{e \notin H^*} a_e c_e$, where $a_e < M$ paths of the network design solution use edge e. It suffices to show that, for each demand pair (s_k, t_k), our assignment costs for s_k and t_k are upper bounded by twice the cost of the edges in $P_k^* \setminus H^*$. This is clear for a demand pair (s_k, t_k) whose path P_k^* is edge-disjoint from H^*, since its assignment cost is $d(s_k, t_k) \leq c(P_k^*) = c(P_k^* \setminus H^*)$. Suppose now that at some point in the procedure, the demand pair (s_k, t_k) got assigned because its path P_k^* intersected the current graph (V, H) in exactly one component, say H_i. Since s_k and t_k are assigned to their nearest neighbors in H_i, it is then easy to see that $d(s_k, i(s_k)) + d(t_k, i(t_k)) \leq c(P_k^* \setminus H_i) \leq c(P_k^* \setminus H^*)$. Finally, suppose demand pair (s_k, t_k) is assigned in case 1 of some iteration of the procedure, with H_i the current component. Since s_k and t_k are assigned to nearest neighbors in H_i, we have $d(s_k, i(s_k)) + d(t_k, i(t_k)) \leq c(P_k^* \setminus H_i) = c(P_k^1) + c(P_k^2) \leq 2c(P_k^1) \leq 2c(P_k^* \setminus H^*)$.

To conclude we prove that $\sum_{e \in H} c_e \leq 2\sum_{e \in H^*} c_e$. Edges are only added to H during case 2 of the above procedure. Suppose this occurs with current component H_{i_1}, and with path P_k^* intersecting components $H_{i_1}, \ldots, H_{i_q}$ with $i_1 < \cdots < i_q$. By eligibility for case 2, the edges added to H at this point have cost at most $\sum_{s=2}^{q} \sum_{e \in H_{i_s}} c_e$. The key observations are these: only components with index larger than that of the current component appear in this expression ($i_s > i_1$ for $s > 1$); once a component appears in this expression, its edges are absorbed into the current component (which retains its index); edges of any such component lie in H^*; and the index of the current component can only increase. Because of these four facts, every edge of H^* participates in the expression $\sum_{s=2}^{q} \sum_{e \in H_{i_s}} c_e$ at most once. Summing over all additions of edges to H, we get $\sum_{e \in H} c_e = \sum_{e \in H^*} c_e + \sum_{e \in H \setminus H^*} c_e \leq 2\sum_{e \in H^*} c_e$. ■

2.2 An LP formulation

We now give an integer programming formulation for MCFL. The decision variables are of the form x_{ij} (1 if demand j is assigned to facility i and 0 otherwise) and z_e (1 if e is selected as a connecting edge and 0 otherwise). The integer program is as follows:

$$\min \quad \sum_{j \in \mathcal{D}} \sum_{i \in V} x_{ij} d(i,j) + M \sum_{e \in E} c_e z_e \quad \text{s.t.} \qquad \text{(IP)}$$

$$\sum_{i \in V} x_{ij} = 1 \qquad \forall j \in \mathcal{D}$$

$$\sum_{e \in \delta(S)} z_e \geq \sum_{i \in S} x_{is_k} - \sum_{i \in S} x_{it_k} \qquad \forall S \subseteq V, s_k \in \mathcal{D}$$

$$\sum_{e \in \delta(S)} z_e \geq \sum_{i \in S} x_{it_k} - \sum_{i \in S} x_{is_k} \qquad \forall S \subseteq V, t_k \in \mathcal{D}$$

$$x_{ij}, z_e \in \{0, 1\},$$

where $\delta(S)$ is the set of edges having precisely one endpoint in S. We replace the integrality constraint by $x_{ij}, z_e \geq 0$

for all e, i, j to obtain a linear program. The dual to this relaxation is

$$\max \quad \sum_{j \in \mathcal{D}} \alpha_j \quad \text{s.t.} \tag{DP}$$

$$\alpha_{s_k} - \sum_{S: i \in S} y_{S,s_k} + \sum_{S: i \in S} y_{S,t_k} \leq d(i, s_k) \tag{1}$$

$$\alpha_{t_k} - \sum_{S: i \in S} y_{S,t_k} + \sum_{S: i \in S} y_{S,s_k} \leq d(i, t_k) \tag{2}$$

$$\sum_{j \in \mathcal{D}} \sum_{S: e \in \delta(S)} y_{S,j} \leq M c_e \ \ \forall e \in E \tag{3}$$

$$y_{S,s_k}, y_{S,t_k} \geq 0 \tag{4}$$

where constraints (1) and (2) range over all $(s_k, t_k) \in \mathcal{D}$ and $i \in V$. By weak duality, any feasible solution to this dual LP is a lower bound on the cost of an optimal solution to the connected facility location problem.

The dual LP should be interpreted as follows. The value α_j is the amount that demand $j \in \mathcal{D}$ is "willing to pay" towards a solution. If demand j is assigned to facility i, a portion of α_j pays for the distance $d(i, j)$; the rest contributes to the connecting edges. At the highest level, the goal of our algorithm (and of any primal-dual algorithm) is to raise the dual variables α_j as much as possible ("generating revenue") while maintaining dual feasibility, thereby ensuring that $\sum_{j \in \mathcal{D}} \alpha_j$ is a valid lower bound on the optimum.

3 The Algorithm

3.1 Difficulties

Before presenting our algorithm, we try to indicate some of the main difficulties that arise in solving MCFL. We first propose a simple primal-dual algorithm for the problem. Call a demand j *tight with facility* i if the constraint (1) for j is satisfied with equality (with respect to the current dual solution), and edge e *tight* if the constraint (3) for e is satisfied with equality. Call a facility i *reachable from* j if there is a facility k with the following property: j is tight with k and there is a path of tight edges between k and i. The algorithm is as follows, and is similar to that of Jain and Vazirani [16] for classical facility location. We begin with all dual variables set to zero, and begin raising the α_j's at a uniform rate. We also raise the dual variable $y_{S_j,j}$ in conjunction with j, where S_j is set of facilities reachable from j. This procedure ensures dual feasibility with the constraints (1) and (2) replaced by

$$\alpha_{s_k} - \textstyle\sum_{S: i \in S} y_{S,s_k} \leq d(i, s_k) \quad \forall\, s_k \in \mathcal{D}, i \in V \tag{5}$$

$$\alpha_{t_k} - \textstyle\sum_{S: i \in S} y_{S,t_k} \leq d(i, t_k) \quad \forall\, t_k \in \mathcal{D}, i \in V. \tag{6}$$

We ignore further issues of dual feasibility for the moment (though our algorithm must handle this difficulty).

When M unassigned demands become tight with a common facility, we open the facility and call this group of demands a *cluster*. Assume we succeed in clustering all of the demands into groups of size M. Intuitively, these are groups large enough to justify building edges to connect the open facilities (since the cost of building an edge is M times the cost of assigning demands across an edge). These clusters induce an instance of the well-solved generalized Steiner problem [1, 10] with clusters as terminals and connectivity requirements induced in a natural way. This suggests running a primal-dual algorithm for the generalized Steiner problem (as in [1, 10]).[2] Unfortunately, a problem arises. The algorithms of [1, 10] build edges one-by-one, until all connectivity requirements are satisfied. When an edge is built, two components of edges merge into one; in our application, this may connect many of the demand pairs in the original connected facility location instance, dropping the number of unsatisfied demands in the new component to a nonzero number much less than M. We may thus encounter a partial solution that fails to satisfy all connectivity requirements and also fails to cluster unsatisfied demands into large enough groups to justify building further edges.

To handle this problem, we are forced to interleave clustering and building phases. This in turn causes several technical problems that must be dealt with. For example, in any given phase, the dual variables of previous phases will contribute to the constraints of type (3), thereby creating many tight edges and forcing the reachable sets S_j to grow large quickly. We deal with this problem in two ways.

First, we break our algorithm into stages, with the dual increase of each variable in one stage being a constant factor larger than the increase in the previous stage; this ensures that dual increases in one stage cannot affect future stages too much. Second, we introduce a method for bounding the proliferation of tight edges via a *distance-preserving property*. Roughly speaking, this property asserts that we can pay for "most" of the tight edges with the current dual solution, in the following sense: if T is the set of tight edges and $B \subseteq T$ are the edges that we can pay for with the current dual solution, then the distances between any pair of vertices in the graphs G_B and G_T obtained by contracting the edges of B and T, respectively, differ only by a small factor. We then show that all demand pairs with source and sink "not too far apart" can be assigned to facilities in the graph G_T with a cost that can be accounted for with our current dual solution; the smallest distance qualifying as "far apart" will increase exponentially with the number of stages. By the distance-preserving property, it follows that assignments in G_B of such demands can be (approximately) paid for. The cost of assignments in G_B approximately reflect the cost of assignments in G (since the contracted edges in G_B

[2]Indeed, this approach leads to a constant-factor approximation for the special case when all sinks are identical.

are the edges of B, which are already paid for and can therefore be used freely), so such demands can be assigned to open facilities without incurring too much cost. At the end of the algorithm, all demands are "not too far apart", and we obtain a feasible solution with small cost.

3.2 Some Preliminaries

Auxiliary Graphs. Our algorithm maintains two graphs, G' and G_B. Both of these graphs will change throughout the execution of our algorithm. Let $d_{G'}$ and d_{G_B} denote shortest-path distance in these two graphs. Let B denote the set of edges already built by our algorithm. As in the previous section, the graph G_B is obtained from G by contracting all edges in B. The distance $d_{G_B}(s_k, t_k)$ should be interpreted as the cost of assigning s_k and t_k to open facilities that are already connected to each other (this is not quite true, but motivates why G_B is a useful network to consider). Since the edges in B are in some sense "already paid for", G_B can be thought of as a "residual network". Also, each connected subgraph H of G_B corresponds to a connected subgraph of G in a natural way; we denote this subgraph by $G[H]$.

The graph G' intuitively corresponds to the graph G_T of the previous section, but has a more complicated definition. We will call G' the *auxiliary graph*. At every point in our algorithm, the graph G' is obtained from G_B by a sequence of the following two operations: (1) contract an edge of G_B; (2) decrease the length, $\ell(e)$, of an edge e from c_e to zero. (While setting an edge length to zero is intuitively the same as contracting it, it will be technically convenient to distinguish between these two operations.) The distance $d_{G'}$ in G' will be with respect to the length $\ell(e)$ of edges in G', which for each edge e will be either c_e or zero.

A vertex $v' \in G'$ corresponds to a connected subgraph in both G_B and in G—we denote these subgraphs $G_B[v']$ and $G[v']$, respectively (see Figure 1). We define $G_B[H']$ and $G[H']$ for a connected subgraph H' of G' in a similar manner. We intuitively think of G' as a "coarser version" of G_B, with each vertex v' in G' representing a small "region" in G_B. We associate with v' a vertex of $G_B[v']$ that we call a *core* (denoted $\mathrm{core}(v')$). As a vertex in G_B, $\mathrm{core}(v')$ is a connected component of built edges that we think of as being "nearby" all vertices of $G_B[v']$; thus if demands in v' need to be assigned to an open facility, $\mathrm{core}(v')$ represents some that are close by. Similarly, building edges between u' and v' in G' should translate in a distance-preserving manner to building edges between $\mathrm{core}(u)$ and $\mathrm{core}(v)$ in G_B.

Each vertex v of G or G_B is contained in some vertex of G'; we will denote this vertex by $v(G')$. When we speak of demand j in G_B (or G'), we mean the vertex of G_B (or G') that contains j.

Some Assumptions. We next make two easily-imposed assumptions about the problem input, which will simplify

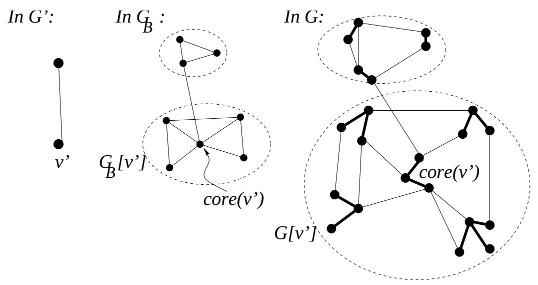

Figure 1. Mappings between vertices of various graphs. Bold edges have been built.

the description of our algorithm: (1) we assume that every edge with non-zero cost has cost precisely ϵ, where ϵ is a sufficiently small constant; (2) we assume that the distance between any two demands that are not co-located is at least $20C^2$, where C is a sufficiently large constant.

Assumption (2) can be enforced by scaling up all distances of the input graph. Assumption (1) is enforced by rounding edge costs to a multiple of ϵ and then subdividing edges until all edges have length precisely ϵ. This permits the dual increases in our algorithm to occur in discrete steps, and affects the approximation ratio by a negligible factor. These subdivisions allow for facilities to be located at these new subdividing points, but simple postprocessing relocates facilities at the original vertices at the expense of a small constant factor increase in the solution cost. There is also a concern that these subdivisions may result in a pseudopolynomial time algorithm, but in fact the algorithm we give below for subdivided networks is easily converted into a strongly polynomial time algorithm. Details are given in the full version.

Defining Tight. We now give our revised definition of what it means for a demand j to be *tight* with a facility i. The definition will be similar but not identical to the notion of "reachable" in Subsection 3.1, and will make use of the auxiliary graph G'.

Initially, a demand j is tight with all facilities that are colocated with it (including j itself). A demand j can become tight with additional facilities when its dual variable α_j is increased or when the auxiliary graph G' is modified. First, if α_j is raised by ϵ units (by assumption (1) above, all dual increases are of this form), then by definition j becomes tight with any facility i satisfying the following: there is a facility k of G and vertices i' and k' of G' containing i and k such that j is tight with k and $d_{G'}(k', i') = \epsilon$.

Finally, the facilities with which j is tight will satisfy the following closure property, by definition: If j is tight with a facility contained in vertex v' of G' and $d_{G'}(v', w') = 0$, then j is tight with all facilities of G contained in w'. This invariant implies that modifications of G' (contracting an edge or decreasing the length of an edge to zero) can implicitly increase the number of facilities with which a demand is tight. Note also that by taking $v' = w'$ in this invariant, it makes sense to say that a demand j is tight with a facility i' in G'; this simply means that j is tight with all facilities of G in $G[i']$.

3.3 Algorithm Description

Disclaimer: Our algorithm will not output a feasible solution to MCFL, but will instead output a partial solution that is easily transformed into a feasible solution with at most twice the cost. Our output consists of a set B of edges and for each demand pair (s_k, t_k), an s_k-t_k path P_k. Our algorithm will guarantee that $M \sum_{e \in B} c_e + \sum_{k=1}^{p} c(P_k \setminus B)$ is at most a constant factor larger than the cost of the optimal solution to the original connected facility location problem. We will say that edges in B are *bought* or *built* and that edges in $P_k \setminus B$ are *rented*. If we began with an instance of MROB, then it is straightforward to check that $P_1, \ldots, P_p$ can be reinterpreted as a solution for this instance with the same (or smaller) cost; it follows from the proof of Lemma 2.1 that this solution has cost within a constant factor of optimal for the rent-or-buy problem. If we desire a solution to the MCFL instance, then the reduction of Lemma 2.1 can be used to extract one from $P_1, \ldots, P_p, B$ that is within a constant factor of optimal.

First, we describe some initial conditions for our algorithm. We start with the empty primal solution (no edges built, no assignments made) and the all-zero dual solution. The auxiliary graph G' is initially G. Each vertex v' of G' will maintain a *budget* (intuitively, the amount of "revenue" it has raised to pay for building edges and making assignments); initially, all budgets are zero. Every demand will be in one of three states: (1) *alive and unfrozen*, which indicates an unassigned demand that is allowed to raise its dual variable; (2) *alive and frozen*, which indicates an unassigned demand that is not allowed to raise its dual variable; (3) *dead*, which indicates an assigned demand. Initially, every demand is alive and unfrozen; frozen demands may subsequently be unfrozen, but dead demands will never be resurrected. Every facility will be either *frozen* (if it participates in a cluster of M demands) or *unfrozen* (otherwise). Initially, every facility is unfrozen; frozen facilities may later become unfrozen.

Whenever we increase a dual variable α_j, we simultaneously increase the dual variable $y_{S_j,j}$, where S_j is the set of facilities with which j is tight (with the definition of tight given in the previous subsection); we will see that this ensures that the relaxed dual constraints (5) and (6) are always satisfied. We will henceforth only describe how to raise the α_j's, which we refer to as "the dual variables", with the understanding that the $y_{S_j,j}$'s are raised in this way.

We now describe stage r of our algorithm ($r \geq 0$). Let G^r be the auxiliary graph at the beginning of stage r; G' will always denote the auxiliary graph at the current point of time (so $G' = G^r$ at the beginning of stage r). The end of the previous stage defines a set F_r of frozen facilities of G^r (in stage 0, $F_0 = \emptyset$); all other facilities, as well as all alive demands, are unfrozen. The previous stage will ensure that facilities of F_r are far from each other in G^r, that each facility $i' \in F_r$ possesses a set $P(i')$ of M *primary demands* that are tight with i', and that no demand is a primary demand for two different frozen facilities. A stage of the algorithm consists of three phases.

Phase 1: The purpose of this phase is to form clusters. We implement this as follows. Dual variables corresponding to alive, unfrozen demands are raised uniformly, until one of the following events happen.

1. There is a frozen facility i' and an alive unfrozen demand j such that either $d_{G'}(i', j) \leq 8 \cdot C^{r+1}$ or j gets tight with i'. (Recall from the previous subsection that C is a sufficiently large constant.) Freeze the demand j and stop raising α_j.

2. There are M alive demand nodes tight with a facility i', and not all of them are frozen. Freeze the facility i', add it to F_r, and set the M demand nodes to be $P(i')$, the primary demands of i'.

3. There is a demand j such that α_j is at least C^{r+1}. Freeze the demand j and stop raising α_j.

Note that freezing of demands or facilities may occur before any dual variables have been raised. Phase 1 terminates when all alive demands are frozen. If more than one of these three events happen simultaneously, we give precedence to event (1).

Phase 2: This phase increases dual variables further (while still ignoring the issue of dual feasibility) to pay for assigning demands and building edges later in this stage. Precisely, we increase α_j of every demand j in $\cup_{i' \in F_r} P(i')$ by C^{r+1}. The budget of each node in F_r is updated to be C^{r+1}.

Phase 3: The final phase of our algorithm is the most complicated and breaks down into several procedures. We maintain a set of nodes X which is initially set to F_r. Let Z_r be the set of edges in G^r ($= G'$) such that the constraint (3) corresponding to these edges in the dual LP is violated, and set $\ell(e)$ to 0 for all these edges. (Of course, $\ell(e)$ may already be 0 in G^r for some edges). Let G' be this new auxiliary graph. Note that G^r and G' have the same set of vertices, but the distance functions in the two graphs are different.

Setting lengths of some edges to be 0 can contract distances in G' by a lot, compared to distances in G_B. In the procedure **CreateNodes**, we identify places where distances have contracted substantially. Since our aim is to maintain the fact that two points are nearly at the same distance in G_B as in G', we build edges in G_B at some of these places so that the corresponding distances go down in G_B as well. To this end, we add more vertices to the set X. To each vertex $v' \in X$, we associate a subgraph $\mathbf{B}(v')$ of G', which is the set of all nodes within distance at most $11 \cdot C^{r+1}$ of v' in G^r. Note the subtlety here that the distance is measured in G^r and not in the current auxiliary graph G'.

Procedure CreateNodes: Suppose there are two vertices $u', v' \in G'$ and P' is a shortest path between them in G' (according to ℓ, the length function on G'). Let $\gamma \ll C$ be a sufficiently large constant. Further suppose that u', v', P' satisfy the following properties: (1) none of the points in P' belong to any of the balls $\mathbf{B}(w')$ for any $w' \in X$; and (2) $d_{G_B}(\mathrm{core}(u'), \mathrm{core}(v'))$ is between γC^{r+1} and $2\gamma C^{r+1}$, whereas $d_{G'}(u', v') \leq \gamma C^{r+1}/4$.

We choose a set of γ points $u'_0, u'_1, \ldots, u'_\gamma$ from the path P' as follows: $u'_0 = u'$, u'_1 is the right-most point on P' such that $(1 - 1/4) \cdot C^{r+1} \leq d_{G_B}(\mathrm{core}(u'), \mathrm{core}(u'_1)) \leq C^{r+1}$, u'_2 is the right-most point in P' such that $d_{G_B}(\mathrm{core}(u'), \mathrm{core}(u'_2))$ is between $(2 - 1/4) \cdot C^{r+1}$ and $2 \cdot C^{r+1}$ and so on. We stop when we find γ such points. Existence of these points and the fact that they lie on P' in this order are proved in the full version. Let $\mathcal{D}(u', v')$ denote the set of these points, and add these $|\mathcal{D}(u', v')| = \gamma$ points to X. We shall say that this procedure creates the pair (u', v').

As before, we also construct the balls $\mathbf{B}(u'_l)$ around all $u'_l \in \mathcal{D}(u', v')$. Note that the union of these balls may not cover all of P'. Indeed, since shortest paths in G' do not map to shortest paths in G_B, there may be a point between u' and u'_1 whose distance from $\mathrm{core}(u')$ in G_B is much more than C^{r+1}.

We keep doing this operation above as long as it is possible. At the end, for each $v' \in X$, we want to contract the sets $\mathbf{B}(v')$ into single nodes. The first problem with this is more of a technical issue. For $w' \in \mathbf{B}(v')$, look at the shortest path in G_B joining $\mathrm{core}(w')$ and $\mathrm{core}(v')$; all the edges in this path may not lie in the set $\mathbf{B}(v')$. To handle this, we *complete* the set $\mathbf{B}(v')$ to $\mathbf{B}'(v')$ thus: initially $\mathbf{B}'(v')$ contains just $\mathbf{B}(v')$. Now if there is a vertex $w' \in \mathbf{B}(v')$ such that the shortest path between $\mathrm{core}(v')$ and $\mathrm{core}(w')$ in G_B uses a vertex x, where $x(G')$ is not in $\mathbf{B}(v')$, then we add $x(G')$ to $\mathbf{B}'(v')$.

We now want to contract $\mathbf{B}'(v')$ into a single node. Another problem presents itself: If $\mathbf{B}'(v')$ and $\mathbf{B}'(u')$ for $u' \neq v' \in X$ share some vertices, then both sets will get contracted to the same node. To decide what the core of this new node will be, we run the following procedure :

Procedure ContractTree(X): Let us construct a graph G_X on the vertex set X thus: $u', v' \in X$ are joined by an edge if $\mathbf{B}'(u') \cap \mathbf{B}'(v') \neq \emptyset$. Now let T_X be a spanning forest in G_X; i.e., T_X restricted to any connected component of G_X is a spanning tree. For each edge $e = (u', v') \in T_X$, let $w' \in \mathbf{B}'(u') \cap \mathbf{B}'(v')$. Find the shortest path between $\mathrm{core}(u')$ and $\mathrm{core}(w')$ in G_B that lies entirely inside $G_B[\mathbf{B}'(u')]$. Similarly, find a path from $\mathrm{core}(w')$ to $\mathrm{core}(v')$ in G_B. Build edges on these paths (hence adding these edges to the set B as well, and contracting all these edges in G_B).

Contract all the vertices in $\mathbf{B}'(v')$ to a single node for each $v' \in X$ in the auxiliary graph. If x' is such a node, then x' may have been obtained by contraction of several of the sets $\mathbf{B}'(v'_1), \ldots, \mathbf{B}'(v'_s)$, where $v'_1, \ldots, v'_s$ form a connected component of G_X in the procedure above. However, note that these contractions are accompanied by the building of edges, and hence $\mathrm{core}(v'_1), \ldots, \mathrm{core}(v'_s)$ contract to a single node in G_B as well. This is defined as $\mathrm{core}(x')$ in G_B, and we set the budget of x' to be C^{r+1}.

Procedure Contract: As the last round of building edges in a stage, we perform the following operation as long as possible. Let u', v' be two nodes with budget C^{r+1} such that the shortest path P' between them in G' has no internal vertex of budget C^{r+1}. Furthermore, suppose that these nodes are "somewhat close"; i.e., $d_{G'}(u', v') \leq 9 \cdot C^{r+2}$. Let $u = \mathrm{core}(u'), v = \mathrm{core}(v')$, find a shortest path P in G_B between u and v, and build edges on this path. Furthermore, if P contains a vertex w such that $w(G')$ is a vertex of budget C^{r+1} in G', then find a shortest path in $G_B[w']$ between w and $\mathrm{core}(w')$, and build edges on this path as well. Contract the edges we just built in G_B. Note that $\mathrm{core}(u')$ and $\mathrm{core}(v')$ will contract to a single vertex in G_B, call this x. P corresponds to a path P'' joining u' and v' in G', and contracting all the edges in P'' creates the new vertex x' in G'. We define $\mathrm{core}(x') = x$, and allot x' a budget of C^{r+1}.

Procedure Prune Demands: Our next step in this phase is to satisfy demands that are sufficiently close to each other in G'. Formally, let s_k, t_k be a pair of alive demands with $d_{G'}(s_k, t_k) \leq 5 \cdot C^{r+2}$; define path P_k connecting them to be the shortest s_k-t_k path in G_B, lifted to an s_k-t_k path of G in the obvious way. Edges of $P_k \setminus B$ are rented, and demands s_k and t_k are marked dead (we will never consider them again in the algorithm).

Procedure Regrow: The final procedure of Phase 3 raises the duals α_j of some demand nodes. For a vertex v' with a budget of C^{r+1}, define $D(v')$ to be the set of those demands j that are only tight with $G[v']$. In particular, such a demand j must lie in $G[v']$, because every demand is tight with itself.

If there is a vertex v' of budget C^{r+1} such that $|D(v')| < M$, then we start raising the α_j value of all demands in $D(v')$ simultaneously, stopping an α_j from rising further if it reaches C^{r+1}. Define F_{r+1} to be the set of all nodes v' that have a budget of C^{r+1}, and that also satisfy $|D(v')| \geq M$. This is the set of frozen facilities for the next stage. Furthermore, for $v' \in F_{r+1}$, define the primary demands $P(v')$ of v' to be any M of the demands in $D(v')$.

4 Overview of Analysis

We now give a high-level overview of the analysis; the precise arguments are given in the full version.

Approximate Dual Feasibility. Unlike a traditional primal-dual algorithm, our algorithm does not explicitly maintain feasibility of the dual solution. We explicitly obey the relaxed constraints (5) and (6) and do not obey the dual constraint (3) for edges at all. On the other hand, we prove that our algorithm always maintains a dual solution that is *approximately* optimal.

Theorem 4.1 *If (α, y) is the dual solution produced by the algorithm, then $(\frac{1}{5}\alpha, \frac{1}{5}y)$ is feasible for the LP (DP).*

Theorem 4.1 is proved in two steps. First, we use the fact that α_{s_k} and α_{t_k} are only raised when s_k and t_k are far apart (because of the **Prune Demands** procedure) in conjunction with feasibility for constraints (5) and (6) to show that constraints (1) and (2) are satisfied. Second, we show that no dual constraint of the form (3) is violated by more than a factor 5. Since we give edges with violated dual constraint length 0 at the beginning of Phase 3 (after which the left-hand side of the constraint will never increase again, by definition of tight), it suffices to prove that the contribution to the left-hand side of the dual constraint corresponding to edge e in a single stage is at most $4Mc_e$. It is easy to show that a single demand can contribute only c_e to the left-hand side, so the problem reduces to showing that only $O(M)$ demands contribute to the left-hand side. Our algorithm ensures this property by only allowing M different demands to become tight with a vertex (this limits the contribution in Phase 1), and by forcing any two frozen facilities to be far apart; primary demands are relatively close to their frozen facilities, and therefore primary demands belonging to different facilities cannot contribute to a single dual constraint (this limits the contribution in Phase 2).

The Distance-Preserving Property. We next argue that distances in G_B and G' are close to each other during the entire run of the algorithm. Let β and λ be constants such that $\beta \ll \gamma \ll \lambda \ll C$.

Theorem 4.2 *Let v' be a node with budget C^s, and let v be its core in G_B. Then $d_{G_B}(u, v) \leq \beta C^s$ for any $u \in G_B[v']$, with such a path lying inside the subgraph $G_B[v']$.*

Furthermore, let $u', v' \in G^s$, with cores $u, v \in G_B$ respectively. Then $d_{G_B}(u, v) \leq 5 \cdot d_{G^s}(u', v') + \lambda C^s$.

We prove Theorem 4.2 by induction on s. Assume the theorem holds for all stages before r. In stage r, we construct the balls $\mathbf{B}(v')$ for each $v' \in X$ of radius about C^{r+1} in G^r, which by induction correspond to radius $O(C^{r+1})$ balls in G_B, as well. Since we construct nodes of budget C^{r+1} by collapsing such balls, the first part of the inductive step essentially follows from the fact that these balls have radii $O(C^{r+1})$ in G_B.

Now we show how to prove the second part of the theorem above. Suppose we have finished the procedure **ContractTree** in Stage r. Let $u', v' \in G'$ have a shortest path P' between them in G' that does not contain any vertex of weight C^{r+1}, and $d_{G_B}(\text{core}(u'), \text{core}(v'))$ lies between γC^{r+1} and $2\gamma C^{r+1}$. We claim that $d_{G'}(u', v') \geq \gamma C^{r+1}/4$; indeed, otherwise we would have considered this path in the procedure **CreateNode** and collapsed it. If u', v', P' satisfy the above properties but $d_{G_B}(\text{core}(u'), \text{core}(v'))$ is much bigger than γC^{r+1}, then we can break the path P' into smaller segments of length about γC^{r+1} and argue independently on each segment. Thus the distance between u' and v' is nearly the same in G_B and G'; though this is only up to an additive factor of about γC^{r+1}.

But what if P' contains nodes of budget C^{r+1}? These nodes can be a problem: since they correspond to subgraphs of radii about C^{r+1} in G_B, we may be contracting distances substantially in G' by collapsing such subgraphs. Consider two consecutive nodes of budget C^{r+1} in P' — the procedure **Contract** ensures that the distance between them is so large that the contraction of these nodes will not have much effect on the distance between u' and v'. This allows us to prove the theorem.

The Performance Guarantee. The cost of the primal solution is the sum of the cost of renting some edges and the cost of building others. Let us first account for the rental costs: if we rent edges between s_k and t_k in the **Prune Demands** procedure of stage r, the distance between s_k and t_k is about C^{r+1} in G', and about the same in G_B as well (using the distance preserving property). So if α_{s_k} or α_{t_k} is at least C^{r+1} then these demands can pay for the cost of renting by their dual variables. If both α_{s_k} and α_{t_k} are small, we can show that both these demands are close enough to some frozen facility such that we can pay for renting of edges to this facility.

Accounting for the edges we build is more involved. Here we use the budgets of nodes, which is roughly the amount it can pay for building edges (scaled down by M). This explains why we assign a budget of C^{r+1} in Phase 2, since we can account for this by the raising of M of the α_j values. In **Contract**, since we build edges between two nodes of budget C^{r+1}, one of these high-budget nodes can pay for cost of building, which is $O(MC^{r+1})$. In procedure **ContractTree** also, each edge of the tree T_X basically corresponds to building edges on $O(C^{r+1})$ length paths, and so each node of T_X has to account for about C^{r+1} length edges. If a node of T_X comes from Phase 2, we know it has a budget of C^{r+1}, and so it can pay for building the edges. If not, then this node in T_X comes from the procedure **CreateNode**, and exists because of a pair (u', v') created by this procedure. But then the distance between u' and v' in G' was much less than that in G_B, and so many of the duals must have been raised for this shrinking of distances. We then show how to borrow C^{r+1} units of budget from these dual variables, which completes the proof of the following theorem.

Theorem 4.3 *The cost of the primal solution constructed by our algorithm is within a constant of* $\sum_j \alpha_j$.

Acknowledgments. We thank Jon Kleinberg and Éva Tardos for helpful discussions.

References

[1] A. Agrawal, P. Klein, and R. Ravi. When trees collide: An approximation algorithm for the generalized Steiner problem on networks. *SIAM Journal on Computing*, 24(3):440–456, 1995.

[2] M. Andrews and L. Zhang. Approximation algorithms for access network design. *Algorithmica*, 34(2):197–215, 2002.

[3] B. Awerbuch and Y. Azar. Buy-at-bulk network design. In *Proceedings of 38th FOCS*, pages 542–547, 1997.

[4] B. Awerbuch, Y. Azar, and Y. Bartal. On-line generalized Steiner problem. In *Proceedings of 7th SODA*, pages 68–74, 1996.

[5] Y. Bartal. On approximating arbitrary metrics by tree metrics. In *Proceedings of 30th STOC*, pages 161–168, 1998.

[6] Y. Bartal, M. Charikar, and P. Indyk. On page migration and other relaxed task systems. In *Proceedings of 8th SODA*, pages 43–52, 1997.

[7] N. Garg, R. Khandekar, G. Konjevod, R. Ravi, F. S. Salman, and A. Sinha. On the integrality gap of a natural formulation of the single-sink buy-at-bulk network design formulation. In *Proceedings of 8th IPCO*, pages 170–184, 2001.

[8] M. X. Goemans and D. Bertsimas. Survivable networks, linear programming relaxations, and the parsimonious property. *Mathematical Programming*, 60:145–166, 1993.

[9] M. X. Goemans, A. V. Goldberg, S. Plotkin, D. B. Shmoys, É. Tardos, and D. P. Williamson. Improved approximation algorithms for network design problems. In *Proceedings of 5th SODA*, pages 223–232, 1994.

[10] M. X. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24:296–317, 1995.

[11] M. X. Goemans and D. P. Williamson. The primal-dual method for approximation algorithms and its application to network design problems. In D. S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, chapter 4, pages 144–191. PWS Publishing Company, 1997.

[12] S. Guha, A. Meyerson, and K. Munagala. Hierarchical placement and network design problems. In *Proceedings of 41st FOCS*, pages 603–612, 2000.

[13] S. Guha, A. Meyerson, and K. Munagala. A constant factor approximation for the single sink edge installation problems. In *Proceedings of 33rd STOC*, pages 383–388, 2001.

[14] A. Gupta, J. Kleinberg, A. Kumar, R. Rastogi, and B. Yener. Provisioning a virtual private network: A network design problem for multicommodity flow. In *Proceedings of 33rd STOC*, pages 389–398, 2001.

[15] K. Jain, M. Madian, and A. Saberi. A new greedy approach for facility location problems. In *Proceedings of 34th STOC*, pages 731–740, 2002.

[16] K. Jain and V. V. Vazirani. Primal-dual approximation algorithms for metric facility location and k-median problems. *Journal of the ACM*, 48:274–296, 2001.

[17] D. R. Karger and M. Minkoff. Building Steiner trees with incomplete global knowledge. In *Proceedings of 41st FOCS*, pages 613–623, 2000.

[18] S. Khuller and A. Zhu. The general Steiner tree-star problem. *Information Processing Letters*, 2002. To appear.

[19] T. U. Kim, T. J. Lowe, A. Tamir, and J. E. Ward. On the location of a tree-shaped facility. *Networks*, 28(3):167–175, 1996.

[20] M. Labbé, G. Laporte, I. Rodrígues Martin, and J. J. Salazar González. The median cycle problem. Technical Report 2001/12, Department of Operations Research and Multicriteria Decision Aid at Université Libre de Bruxelles, 2001.

[21] Y. Lee, S. Y. Chiu, and J. Ryan. A branch and cut algorithm for a Steiner tree-star problem. *INFORMS Journal on Computing*, 8(3):194–201, 1996.

[22] M. Madian, E. Markakis, A. Saberi, and V. V. Vazirani. A greedy facility location algorithm analyzed using dual fitting. In *Proceedings of 4th APPROX*, pages 127–137, 2001.

[23] F. S. Salman, J. Cheriyan, R. Ravi, and S. Subramanian. Approximating the single-sink link-installation problem in network design. *SIAM Journal on Optimization*, 11(3):595–610, 2000.

[24] C. Swamy and A. Kumar. Primal-dual algorithms for the connected facility location problem. To appear in *APPROX 2002*.

[25] K. Talwar. Single-sink buy-at-bulk LP has constant integrality gap. In *Proceedings of 9th IPCO*, pages 475–486, 2002.

[26] D. P. Williamson. The primal-dual method for approximation algorithms. *Mathematical Programming, Series B*, 91(3):447–478, 2002.

[27] D. P. Williamson, M. X. Goemans, M. Mihail, and V. V. Vazirani. A primal-dual approximation algorithm for generalized Steiner network problems. *Combinatorica*, 15:435–454, 1995.