Approximation Via Cost-Sharing: A Simple Approximation Algorithm for the Multicommodity Rent-or-Buy Problem

Anupam Gupta* Amit Kumar† Martin Pál ‡ Tim Roughgarden‡

April 7, 2003

Abstract

We study the *multicommodity rent-or-buy problem*, a type of network design problem with economies of scale. In this problem, capacity on an edge 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. The first constant-factor approximation algorithm for this problem was recently given by Kumar et al. (FOCS '02); however, this algorithm and its analysis are both quite complicated, and its performance guarantee is extremely large.

In this paper, we give a 12-approximation algorithm for this problem. The algorithm is conceptually very simple: it picks a random subset of the source-sink pairs, buys a set of edges spanning these chosen pairs, and greedily rents paths for the other source-sink pairs. Moreover, our approximation ratio is by far the best known for any network design problem with economies of scale, other than special cases of the rent-or-buy problem.

Our analysis of this algorithm makes crucial use of *cost sharing*, the task of allocating the cost of an object to many users of the object in a "fair" manner. While techniques from approximation algorithms have recently yielded new progress on cost sharing problems, our work is the first to show the converse—that ideas from cost sharing can be fruitfully applied in the design and analysis of approximation algorithms.

*Department of Computer Science, Carnegie Mellon University, Pittsburgh PA 15213. Email: `anupamg@cs.cmu.edu`

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

‡Department of Computer Science, Cornell University, Ithaca NY 14853. Supported by ONR grant N00014-98-1-0589. Email: `{mpal,timr}@cs.cornell.edu`.

1 Introduction

We study the *multicommodity rent-or-buy* (MRoB) problem. In this problem, we are given an undirected graph $G = (V, E)$ with non-negative weights c_e on the edges e and a set $\mathcal{D} = \{(s_1, t_1), \ldots, (s_k, t_k)\}$ of vertex pairs called *demand pairs*. 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_i to the corresponding sink t_i. The cost of installing capacity on an edge is given by a simple concave function: 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. Precisely, there are 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). By scaling, there is no loss of generality in assuming that $\mu = 1$.

The MRoB problem is a simple model of network design with *economies of scale*, and is a central special case of the more general *buy-at-bulk network design* problem, where the cost of installing capacity can be described by an arbitrary concave function. In addition, the MRoB problem naturally arises as a subroutine in multicommodity versions of the *connected facility location* problem and the *maybecast* problem of Karger and Minkoff [21]; see [24] for further details on these applications.

The MRoB problem is easily seen to be NP- and MAX SNP-hard (for example, it contains the Steiner tree problem as a special case [9]), and researchers have therefore sought approximation algorithms for the problem.[1] For many years, the best known performance guarantee for MRoB was the $O(\log n \log \log n)$-approximation algorithm due to Awerbuch and Azar [3] and Bartal [6] ($n = |V|$ denotes the number of nodes in the network). The first constant-factor approximation algorithm for the problem was recently given by Kumar et al. [24]. However, both the analysis and the primal-dual algorithm of [24] are quite complicated, and the performance guarantee shown for the algorithm is, while constant, extremely large.[2] The problem of obtaining an algorithm with constant performance guarantee for MRoB while using only a transparent algorithm and/or obtaining a reasonable constant has since remained open.

In a separate recent development, Gupta et al. [19] showed that extremely simple randomized combinatorial algorithms suffice to achieve best-known performance guarantees for several network design problems, including the *single-sink* special case of MRoB in which all sink vertices t_i are identical. However, the analysis of [19] required that the underlying "buy-only" problem (such as the Steiner tree problem for the single-sink special case of MRoB) admit a good greedy approximation algorithm (e.g., the MST heuristic for Steiner tree, with Prim's MST algorithm [29]). Since the "buy-only" version of the MRoB problem is the Steiner forest problem (see e.g. [1]), for which no greedy algorithm is known, it was not clear if the techniques of [19] could yield good algorithms for MRoB.

Our Results: We show how a nontrivial extension of the randomized framework of [19] gives a 12-approximation algorithm for MRoB. The algorithm is conceptually very simple: it picks a random subset of the source-sink pairs, buys a set of edges spanning these chosen pairs, and greedily rents paths for the other source-sink pairs.

Our analysis of the algorithm is based on *cost sharing*, the task of allocating the cost of an object to many users of the object in a "fair" manner. Cost sharing has been extensively studied in the economics literature (see e.g. [37]); more recently, techniques from approximation algorithms have yielded new progress in this field, see e.g. [20]. We believe the present work to be the first showing the converse, that ideas from cost sharing can lead to better approximation

[1] Recall a *c-approximation algorithm* for a minimization problem runs in polynomial time and returns a solution no more than c times as costly as an optimal solution. The value c is the *approximation ratio* or *performance guarantee* of the algorithm.

[2] While this constant was neither optimized nor estimated in [24], the approach of that paper only seems suitable to proving a performance guarantee of at least several hundred.

algorithms. A second key ingredient for our result is a simple but novel extension of the primal-dual algorithms of Agrawal et al. [1] and Goemans and Williamson [15] for the Steiner forest problem.

Our performance guarantee of 12 is almost certainly not the best achievable for the MRoB problem, but it is far better than any other approximation ratio known for a network design problem exhibiting economies of scale, with the exception of the single-sink special case of MRoB (for which a 3.55-approximation is known [19]). Single-sink buy-at-bulk network design, where all commodities share a common sink but the cost of installing a given amount of capacity can essentially be an arbitrary concave function, is only known to be approximable to within a factor of 73 [19]. Keeping the single-sink assumption and placing further restrictions on the function describing the cost of capacity yields the *Access Network Design* problem of Andrews and Zhang [2], for which the best known approximation ratio is 68 [28]. (There are no known constant-factor approximation algorithms for the multicommodity versions of these two problems.) The previous-best performance guarantee for MRoB was yet an order of magnitude larger [24]. The present work is thus the first to suggest that restricting the capacity cost function could yield a more tractable special case of multicommodity buy-at-bulk network design than the popular assumption that all commodities share a common sink.

Related Work: As stated above, the only previous constant-factor approximation algorithm for the MRoB problem studied in this paper is due to Kumar et al. [24]. Additional papers that considered multicommodity network design with economies of scale are Awerbuch and Azar [3] and Bartal [6], who gave an $O(\log n \log\log n)$-approximation to the more general multicommodity buy-at-bulk problem. The special case of MRoB where all commodities share the same sink, and the closely related *connected facility location* problem, have been extensively studied [18, 19, 21, 22, 23, 25, 26, 30, 34]. The randomized 3.55-approximation algorithm of Gupta et al. [19] is the best known approximation algorithm for the problem. Swamy and Kumar [34] achieve a performance guarantee of 4.55 with a deterministic algorithm. Several more general problems that retain the single-sink assumption have also been intensively studied in recent years, including the Access Network Design problem [2, 16, 17, 28], the single-sink buy-at-bulk network design problem [13, 17, 19, 32, 35], and the still more general problems where the capacity cost function can be edge-dependent [11, 27] or unknown to the algorithm [14]. The best known approximation ratios for these four problems are 68 [28], 73 [19], $O(\log n)$ [11, 27], and $O(\log n)$ [14], respectively.

Finally, our high-level algorithm of randomly reducing the MRoB problem to the Steiner forest problem, followed by computing shortest paths, is similar to and partially inspired by previous work that gave online algorithms with polylogarithmic competitive ratios for many rent-or-buy-type problems [4, 5, 7, 8].

2 Approximation via Cost Sharing

In this section we show how an appropriate cost-sharing scheme gives a good approximation algorithm for MRoB. We will give such a cost-sharing scheme in the technical heart of the paper, Sections 3 and 4.

In Subsection 2.1, we define our desiderata for cost shares. We give the main algorithm in Subsection 2.2, the analysis in Subsection 2.3, and an illustration in the single-sink special case of MRoB in Subsection 2.4.

2.1 Some Definitions

We now describe precisely what we mean by a cost-sharing method, as well as the additional properties required by our application. Cost-sharing methods can be defined quite generally (see e.g. [37]); here, we take a narrower approach. In preparation for our first definition, recall that an instance of MRoB is defined by a weighted undirected graph G (we leave the weight vector c implicit) and a set $\mathcal{D}$ of demand pairs. By a *Steiner forest* for $(G, \mathcal{D})$, we mean a subgraph F of G so that, for each demand pair $(s,t) \in \mathcal{D}$, there is an s-t path in F. For such a subgraph F,

$c(F) = \sum_{e\in F} c_e$ denotes its overall cost. Since we are only interested in solutions with small cost and edge costs are non-negative, we can always assume that F is a forest.

The next definition states that a cost-sharing method is a way of allocating cost to the demand pairs of an instance $(G, \mathcal{D})$, with the total cost bounded above by the cost of an optimal Steiner forest for $(G, \mathcal{D})$. (While this latter quantity is NP-hard to compute, the definition is still useful in what follows.)

Definition 2.1 A *cost-sharing method* is a non-negative real-valued function χ defined on a triple $(G, \mathcal{D}, (s,t))$ of a weighted undirected graph G, a set $\mathcal{D}$ of demand pairs, and a single demand pair $(s,t) \in \mathcal{D}$. Moreover, for every instance $(G, \mathcal{D})$ admitting a min-cost Steiner forest $F^*_{\mathcal{D}}$,

$$\sum_{(s,t)\in\mathcal{D}} \chi(G, \mathcal{D}, (s,t)) \leq c(F^*_{\mathcal{D}}).$$

Definition 2.1 permits some rather uninteresting cost-sharing methods, such as the function that always assigns all demand pairs zero cost. The key additional property that we require of a cost-sharing method is that, intuitively, it allocates costs to each demand pair commensurate with its distance from the edges needed to connect all of the other demand pairs. Put differently, no demand pair can be a "free rider", imposing a large burden in building a Steiner forest, but only receiving a small cost share. We call cost sharing methods with this property *strict*.

To make this idea precise, we require further notion. Let $d_G(\cdot,\cdot)$ denote the shortest-path distance in G (w.r.t. edge costs in G). Given a subset $E' \subseteq E$ of edges, G/E' is the graph obtained from G by contracting the edges in E'. Note that the cheapest way of connecting vertices s and t by renting edges, given that edges E' have already been bought, is precisely $d_{G/E'}(s,t)$. Our main definition is then the following.

Definition 2.2 Let $\mathcal{A}$ be a deterministic algorithm that, given instance $(G, \mathcal{D})$, computes a Steiner forest. A cost-sharing method is *β-strict for $\mathcal{A}$* if for all $(G, \mathcal{D})$ and $(s,t) \in \mathcal{D}$, the cost $\chi(G, \mathcal{D}, (s,t))$ assigned to (s,t) by χ is at least a $1/\beta$ fraction of $d_{G/F}(s,t)$, where F is the Steiner forest returned for $(G, \mathcal{D} \setminus \{(s,t)\})$ by algorithm $\mathcal{A}$.[3]

It is not clear *a priori* that strict cost-sharing methods with small β exist: Definition 2.1 states that the aggregate costs charged to demand pairs must be reasonable, while Definition 2.2 insists that the cost allocated to each demand pair is sufficiently large.

We note that strict cost-sharing methods are reminiscent of some central concepts in cooperative game theory, such as the *core* and the *nucleolus* (see e.g. [33]). However, we are not aware of any precise equivalence between strict cost-sharing methods and existing solution concepts in the game theory literature.

2.2 The Algorithm SimpleMROB

We now state the main algorithm. In employs as a subroutine a Steiner forest algorithm $\mathcal{A}$, which in our implementation will be a constant-factor approximation algorithm. No cost-sharing method is needed for the description or implementation of the algorithm; cost shares arise only in the algorithm's analysis. We assume for simplicity that each source-sink pair was to transport a single unit of flow; this assumption is easy to remove, see Appendix A.

A1. *Mark* each pair (s_i, t_i) with probability $1/M$, and let $\mathcal{D}_{mark}$ be the set of marked demands.

A2. Construct a Steiner forest F on $\mathcal{D}_{mark}$ using algorithm $\mathcal{A}$, and *buy* all edges in F.

A3. For each (s_i, t_i) pair outside $\mathcal{D}_{mark}$, *rent* edges to connect s_i and t_i in a minimum-cost way (at cost $d_{G/F}(s,t)$).

[3]A cost-sharing method (which is defined independent of any Steiner forest algorithm) can be β-strict for one algorithm and not for another, as strictness depends on the distance $d_{G/F}(s,t)$, and the edge set F is algorithm-dependent.

2.3 Proof of Performance Guarantee

We now state the main theorem of this section: a constant-factor approximation algorithm for Steiner forest with an accompanying $O(1)$-strict cost-sharing method yields a constant-factor approximation algorithm for MRoB. The proof is an extension of the techniques of [19], where the connection to cost sharing was not explicit and therefore only simpler network design problems were considered.

Theorem 2.3 *Suppose $\mathcal{A}$ is an α-approximation algorithm for the Steiner forest problem that admits a β-strict cost sharing method. Then algorithm* SimpleMROB*, employing algorithm $\mathcal{A}$ as a subroutine in Step (A2), is an $(\alpha + \beta)$-approximation algorithm for the multicommodity rent-or-buy problem.*

Proof: Fix an arbitrary instance of MRoB with an optimal solution OPT, and let Z^* denote the cost of OPT. Let B^* denote the cost of the bought edges E_b in OPT, and R^* the cost of the rented edges E_r. (Note that $B^* = Mc(E_b)$, and $R^* = \sum_{e \in E_r} c_e x_e^*$, where x_e^* is the amount of capacity that OPT reserves on edge e, or equivalently the number of demand pairs that use e to route their flow). It suffices to show that algorithm SimpleMROB incurs an expected cost of at most αZ^* in Step (A2) and at most βZ^* in Step (A3). We prove each of these bounds in turn.

For the first bound, it suffices to show that:

$$\mathbb{E}[\text{ cost of buying a min-cost Steiner forest on the (random) set of demand pairs } \mathcal{D}_{mark}\,] \le Z^*. \tag{2.1}$$

To prove this, it suffices to exhibit a (random) subgraph of G that spans the vertices of $\mathcal{D}_{mark}$ that has expected cost at most Z^*/M. To construct this subgraph, include all edges of E_b, and every edge e of E_r for which some demand pair using e in OPT is marked. The cost of the edges in E_b is deterministically B^*/M. For each edge in E_r, the expected cost of including it is $c_e \times 1/M \times x_e^* = c_e x_e^*/M$ (each of the x_e^* demand pairs using e in OPT contributes $1/M$ to the chance of e being included). Summing over all edges of E_r, the expected cost of including edges of E_r is R^*/M; since $Z^* = B^* + R^*$, inequality (2.1) is proved.

We now bound the expected cost incurred in Step (A3). We will say that demand pair (s,t) incurs *buying cost* $B_i = \chi(G, \mathcal{D}_{mark}, (s,t))$ and *renting cost* $R_i = 0$ if $(s,t) \in \mathcal{D}_{mark}$, and buying cost $B_i = 0$ and renting cost $R_i = d_{G/F}(s,t)$ otherwise.

For a demand pair (s_i, t_i), let $X_i = R_i - \beta B_i$ denote the random variable equal to the renting cost of this pair minus β times its buying cost. We next condition on the outcome of all of the coin flips in Step (A1) except that for (s_i, t_i). Let $\mathcal{D}$ denote the demand pairs that have been marked so far, together with (s_i, t_i). Since χ is β-strict for $\mathcal{A}$, this conditional expectation is at most 0. Since this inequality holds for any outcome of the coin flips for demand pairs other than (s_i, t_i), it also holds unconditionally: $E[X_i] \le 0$. Summing over all i and applying linearity of expectations, we find that

$$\mathbb{E}[\textstyle\sum_i R_i] \le \beta\, \mathbb{E}[\textstyle\sum_i B_i]. \tag{2.2}$$

The left-hand side of (2.2) is precisely the expected cost incurred in Step (A3) of the algorithm. By Definition 2.1, the sum $\sum_i B_i$ is at most the cost of the min-cost Steiner forest on the set $\mathcal{D}_{mark}$ of demands; by (2.1), it follows that the right-hand side of (2.2) is at most βZ^*, and the proof is complete. ■

In Sections 3 and 4, we will give a 6-approximation algorithm for Steiner forest that admits a 6-strict cost sharing method. The main theorem of the paper is then a direct consequence of Theorem 2.3.

Theorem 2.4 *There is a 12-approximation algorithm for* MRoB.

2.4 The Single-Sink Case

As a warm-up to the main result of the paper, we now briefly illustrate how to apply Theorem 2.3 to the special case of the single-sink rent-or-buy problem, thereby rederiving a result from [19]. In this setting, the subroutine $\mathcal{A}$ in Step (A2) of algorithm SimpleMROB must construct a Steiner tree on the sink t and the set of marked sources (unless no demand pairs are marked, in which case $\mathcal{A}$ returns the empty set). We instantiate $\mathcal{A}$ as Prim's minimum spanning tree algorithm [29], beginning with the common sink t as the root vertex, running in the graph of shortest-path distances between marked vertices. This is well known to be a 2-approximation for the Steiner tree problem (see e.g. [36]).

We now define a 2-strict cost-sharing method for $\mathcal{A}$ as follows: commodity (s,t) receives a cost share χ equal to half the length of the first edge incident to s, in the graph of shortest-path distances on the marked vertices, chosen by Prim's algorithm. Using the above fact that the minimum spanning tree 2-approximates the min-cost Steiner tree, it is easy to check that χ is a cost-sharing method in the sense of Definition 2.1. To prove that the method is 2-strict, consider the output of algorithm $\mathcal{A}$ on the nonempty set $\mathcal{D}$ of demand pairs and on $\mathcal{D} \setminus \{(s,t)\}$ for some $(s,t) \in \mathcal{D}$. The execution of $\mathcal{A}$ is identical in the two scenarios until s is the next marked vertex to be included in the tree; at this point, $\mathcal{A}$ has constructed a tree T that is closer to s than to any other source in $\mathcal{D}$ not yet spanned by T. The renting cost to connect s to T is twice the cost share that (s,t) receives in the former execution of $\mathcal{A}$, and this cost can only decrease as $\mathcal{A}$ extends T to a spanning tree on $\mathcal{D} \setminus \{(s,t)\}$. Since our choices of $\mathcal{D}$ and $(s,t) \in \mathcal{D}$ were arbitrary, χ is 2-strict. Applying Theorem 2.3 with $\alpha = \beta = 2$, we have proved the following.

Theorem 2.5 ([19]) *There is a 4-approximation for the single-sink rent-or-buy problem.*

Using the 1.55-approximation algorithm for Steiner tree by Robins and Zelikovsky [31], the performance guarantee of the algorithm can be improved to 3.55. See [19] for details.

3 Our Steiner Forest Algorithm and Cost Shares

3.1 Motivation

Linear programming duality is well known to have an economic interpretation, and moreover to be useful in cost sharing applications (see e.g. [20] for a recent example). It is therefore natural to expect strict cost-sharing methods to fall out of existing primal-dual approximation algorithms for the Steiner forest problem, such as those by Agrawal et al. [1] and Goemans and Williamson [15]. In particular, the reader might hope that taking the subroutine $\mathcal{A}$ of algorithm SimpleMROB to be such a primal-dual algorithm, and defining the cost shares χ according to the corresponding dual solution, would be enough to obtain a constant-factor approximation for MRoB.

Unfortunately, we have discovered examples that demonstrate that straightforward implementations of this idea cannot give β-strict cost-sharing methods for any constant β. On the other hand, in these families of examples, an $O(1)$-strict cost-sharing method can be defined if a few extra edges are bought (extra edges make Definition 2.2 easier to satisfy, since the shortest-path distance $d_{G/F}$ relative to the bought edges F decreases). Our main technical result is that buying a few extra edges beyond what is advocated by the algorithms of [1, 15] *always suffices* for defining an $O(1)$-strict cost-sharing method, enabling the application of Theorem 2.3.

3.2 The Algorithm PD and the Cost Shares χ

In this subsection we show how to extend the algorithms of [1, 15] to "build a few extra edges" while remaining constant-factor approximation algorithms for the Steiner forest problem. We also describe our cost-sharing method.

Recall that we are given a graph $G = (V, E)$ and a set $\mathcal{D}$ of source-sink pairs $\{(s_k, t_k)\}$. Let D be the set of *demands*— the vertices that are either sources or sinks in $\mathcal{D}$ (without loss of generality, all demands are distinct). It will be convenient to associate a cost share $\chi(\mathcal{D}, j)$ with each demand $j \in D$; the cost share $\chi(G, \mathcal{D}, (s, t))$ is then just $\chi(\mathcal{D}, s) + \chi(\mathcal{D}, t)$. Note that we have also dropped the reference to G; in the sequel, the cost shares are always w.r.t. G.

Before defining our algorithm, we review the LP relaxation and the corresponding LP dual of the Steiner forest problem that was used in [15]:

$$\begin{aligned} &\min \textstyle\sum_e c_e x_e && \text{(LP)} \\ &x(\delta(S)) \geq 1 \quad \forall \text{ valid } S && (3.3) \\ &x_e \geq 0 \end{aligned}$$

$$\begin{aligned} &\max \textstyle\sum_S y_S && \text{(DP)} \\ &\textstyle\sum_{S \subseteq V : e \in \delta(S)} y_S \leq c_e && (3.4) \\ &y_S \geq 0, \end{aligned}$$

where a set S is *valid* if for some i, it contains precisely one of s_i, t_i.

We now describe a general way to define primal-dual algorithms for the Steiner forest problem. As is standard for the primal-dual approach, the algorithm with maintain a feasible (fractional) dual, initially the all-zero dual, and a primal integral solution (a set of edges), initially the empty set. The algorithm will terminate with a feasible Steiner forest, which will be proved approximately optimal with the dual solution (which is a lower bound on the optimal cost by weak LP duality). The algorithms of [1, 15] arise as a particular instantiation of the following algorithm. Our presentation is closer to [1], where the "reverse delete step" of Goemans and Williamson [15] is implicit; this version of the algorithm is more suitable for our analysis.

Our algorithm has a notion of *time*, initially 0 and increasing at a uniform rate. At any point in time, some demands will be *active* and others *inactive*. All demands are initially active, and eventually become inactive. The vertex set is also partitioned into *clusters*, which can again be either active or inactive. In our algorithm, a cluster will be one or more connected components (w.r.t. the currently built edges). Initially, each vertex is a cluster, and the demands are the active clusters. We will consider different rules by which demands and clusters become active or inactive. To maintain dual feasibility, whenever the constraint (3.4) for some edge e between two clusters S and S' becomes tight (i.e., first holds with equality), the clusters are *merged* and replaced by the cluster $S \cup S'$. We raise dual variables of active clusters until there are no more such clusters.

We have not yet specified how an edge can get built. Toward this end, we define a (time-varying) equivalence relation $\mathcal{R}$ on the demand set. Initially, all demands lie in their own equivalence class; these classes will only grow with time. When two active clusters are merged, we merge the equivalence classes of all active demands in the two clusters. Since inactive demands cannot become active, this rule ensures that all active demands in a cluster are in the same equivalence class.

We build edges to maintain the following invariant: the demands in the same equivalence class are connected by built edges. This clearly holds at the beginning, since the equivalence classes are all singletons. When two active clusters meet, the invariant ensures that, in each cluster, all active demands lie in a common connected component. To maintain the invariant, we join these two components by adding a path between them. Building such paths without incurring a large cost is simple but somewhat subtle; Agrawal et al. [1] (and implicitly, Goemans and Williamson [15]) show how to accomplish it. We will not repeat their work here, and instead refer the reader to [1].

Remark 3.1 For the reader more familiar with the exposition of Goemans and Williamson [15], let us give an (informal) alternate description of the network output by the algorithm given above. Specifically, we grow active clusters uniformly, and when *any* two clusters merge, we build an edge between them. At the end, we perform a reverse-delete step—when looking at an edge e, if e lies on the path between some x and y with (x, y) in the final relation $\mathcal{R}$, then we keep the edge, else we delete it. The reader will notice that the network finally output by this new description is *exactly* the same as the one previously given; however, they may not be the same at intermediate points of the execution, and hence we will *not* use this restatement in the rest of the paper. ■

Specifying the rule by which clusters are deemed active or inactive now gives us two different algorithms:

1. **Algorithm** $\mathsf{GW}(G, \mathcal{D})$: A demand s_i is active if the current cluster containing it does not contain t_i. A cluster is active as long as it contains at least one active demand. This implementation of the algorithm is equivalent to the algorithms of Agrawal et al. [1] and Goemans and Williamson [15].
2. **Algorithm** $\mathsf{Timed}(G, D, T)$: This algorithm takes as an additional input a function $T : V \to \mathbb{R}_{\geq 0}$ which assigns a *stopping time* to each vertex. (We can also view T as a vector with coordinates indexed by V.) A vertex j is active at time τ if $j \in D$ *and* $\tau \leq T(j)$. (T is defined for vertices not in D for future convenience, but such values are irrelevant.) As before, a cluster is said to be active if at least one demand in it is active.

To get a feeling for $\mathsf{Timed}(G, D, T)$, consider the following procedure: run the algorithm $\mathsf{GW}(G, \mathcal{D})$ and set $T_{\mathcal{D}}(j)$ to be the time at which vertex j becomes inactive during this execution. (If $j \notin D$, then $T_{\mathcal{D}}(j)$ is set to zero.) Since the period for which a vertex stays active in the two algorithms $\mathsf{GW}(G, \mathcal{D})$ and $\mathsf{Timed}(G, D, T_{\mathcal{D}})$ is the same, they clearly have identical outputs.

The Timed algorithm gives us a principled way to essentially force the GW algorithm to build additional edges: run the Timed algorithm with a vector of demand activity times that is larger than what is naturally induced by the GW algorithm.

The Algorithm PD: The central algorithm, **Algorithm** $\mathsf{PD}(G, \mathcal{D})$ is obtained thus: run $\mathsf{GW}(G, \mathcal{D})$ to get the time vector $T_{\mathcal{D}}$; then run $\mathsf{Timed}(G, D, \gamma T_{\mathcal{D}})$—the timed algorithm with the GW-induced time vector scaled up by a parameter $\gamma \geq 1$—to get a forest $F_{\mathcal{D}}$. (We will fix the value of γ later in the analysis.)[4]

The fact that $F_{\mathcal{D}}$ is a feasible Steiner network for $\mathcal{D}$ is easily verified, using the two facts that each demand pair (s_i, t_i) was connected at time $T_{\mathcal{D}}(s_i) = T_{\mathcal{D}}(t_i)$, and that $\gamma \geq 1$. We now define the cost shares χ.

The Cost Shares χ: For a demand $j \in D$, the cost share $\chi(\mathcal{D}, j)$ is the length of time during the run $\mathsf{GW}(G, \mathcal{D})$ in which j was the only active vertex in its cluster. Formally, let $a(j, \tau)$ be the indicator variable for the event that j is the only active vertex in its cluster at time τ; then

$$\chi(\mathcal{D}, j) = \int a(j, \tau)\, d\tau, \tag{3.5}$$

where the integral is over the entire execution of the algorithm.

In the sequel, we will prove our main technical result:

Theorem 3.2 PD *is a* $\alpha = 2\gamma$*-approximation for the Steiner network problem, and* χ *is a* $\beta = 6\gamma/(2\gamma - 3)$*-strict cost-sharing method for* PD.

Setting $\gamma = 3$ then gives us a 6-approximation algorithm that admits a 6-strict cost-sharing method, as claimed in Theorem 2.4.

[4] A technical point is that when $\gamma > 1$, algorithm PD may raise the dual variables of vertex sets that are not valid, and hence may produce a dual solution that is not feasible. As we will see in Lemma 4.2, however, this will not hinder our analysis.

4 Outline of Proof of Theorem 3.2

4.1 Two Easy Facts

We first show that algorithm PD is a 2γ-approximation algorithm for Steiner forest. We begin with an intuitive monotonicity lemma, which implies that the set of edges built by algorithm PD is monotone in the parameter γ. We omit its somewhat tedious proof.

Lemma 4.1 *Let T and T' be two time vectors with $T(j) \le T'(j)$ for all demands $j \in D$. Then at any time τ, the set of tight edges in* $\mathsf{Timed}(G, D, T)$ *is a subset of those in* $\mathsf{Timed}(G, D, T')$.

We can now prove the claimed approximation ratio.

Lemma 4.2 *The cost of the Steiner forest $F_{\mathcal{D}}$ constructed by algorithm* PD *for instance $(G, \mathcal{D})$ is at most $2\gamma \sum_S y_S \le 2\gamma\, c(F^*_{\mathcal{D}})$, where $F^*_{\mathcal{D}}$ is an optimal Steiner forest for $(G, \mathcal{D})$.*

Proof: Let $\{y_S\}$ be the dual variables constructed by the run of $\mathsf{GW}(G, \mathcal{D})$. If $a(\tau)$ is the number of active clusters at time τ during this run, then

$$\textstyle\sum_S y_S = \int a(\tau) d\tau. \tag{4.6}$$

Similarly, let $a'(\tau)$ be the number of active clusters at time τ during the execution of $\mathsf{Timed}(G, D, \gamma T_{\mathcal{D}})$, and $\{y'_S\}$ be the dual solution constructed by it. Recall that $\{y'_S\}$ need not be dual feasible, since in general we have $y'_S > 0$ for invalid S. As in (4.6), $\sum_S y'_S = \int a'(\tau) d\tau$.

First, we claim that the cost of $F_{\mathcal{D}}$ is at most $2 \sum_S y'_S$. This follows from the arguments in Agrawal et al. [1] (since our algorithm builds paths between merging clusters as in [1]).

We next relate $\sum_S y'_S$ to the cost of an optimal Steiner forest, via the feasible dual solution $\{y_S\}$. Toward this end, we claim that $a'(\gamma\tau) \le a(\tau)$. Indeed, let $C_1, \dots, C_k$ be the active clusters in $\mathsf{Timed}(G, D, \gamma T_{\mathcal{D}})$ at time $\gamma\tau$. Each active cluster must have an active demand – let these be $j_1, \dots, j_k$. By the definition of $T_{\mathcal{D}}$, these demands must have been active at time τ in $\mathsf{GW}(G, \mathcal{D})$, and by Lemma 4.1, no two of them were in the same cluster at this time. With this claim in hand, we can derive

$$\int a'(\tau) d\tau = \gamma \int a'(\gamma\tau) d\tau \le \gamma \int a(\tau) d\tau \le \gamma \sum_S y_S.$$

Since $\sum_S y_S$ is a lower bound on the optimal cost $c(F^*_{\mathcal{D}})$ (by LP duality), the lemma is proved. ∎

Is is also easy to show that χ is a cost-sharing method in the sense of Definition 2.1.

Lemma 4.3 *The function χ satisfies*

$$\sum_{(s,t) \in \mathcal{D}} \chi(G, \mathcal{D}, (s,t)) = \sum_{j \in D} \chi(\mathcal{D}, j) \le \sum_S y_S \le c(F^*_{\mathcal{D}}).$$

Proof: At any point τ in time, $\sum_j a(j, \tau) \le a(\tau)$, since each active cluster can have at most one demand j with non-zero $a(j, \tau)$. Thus $\int \sum_j a(j, \tau) d\tau \le \int a(\tau) d\tau = \sum_S y_S$. ∎

4.2 Outline of Proof of Strictness

We first recall some notation we will use often. The algorithm $\mathsf{PD}(G, \mathcal{D})$ first runs $\mathsf{GW}(G, \mathcal{D})$ to find a time vector $T_{\mathcal{D}}$, and then runs $\mathsf{Timed}(G, D, \gamma T_{\mathcal{D}})$ to build a forest $F_{\mathcal{D}}$. Let (s, t) be a new source-sink pair. Define $\mathcal{D}' = \mathcal{D} \cup \{(s, t)\}$, $D' = D \cup \{s, t\}$ and let $T_{\mathcal{D}'}$ be the time vector obtained by running $\mathsf{GW}(G, \mathcal{D}')$. Our sole remaining hurdle is the following theorem, asserting the strictness of our cost-sharing method χ for the algorithm PD.

Theorem 4.4 (Strictness) *Let (s, t) be a source-sink pair $\notin \mathcal{D}$, and let $\mathcal{D}' = \mathcal{D} + (s, t)$ denote the demand set obtained by adding this new pair to $\mathcal{D}$. Then the length of the shortest path $d_{G/F_{\mathcal{D}}}(s, t)$ between s and t in $G/F_{\mathcal{D}}$ is at most $\beta(\chi(\mathcal{D}', s) + \chi(\mathcal{D}', t))$, where $\beta = 6\gamma/(2\gamma - 3)$.*

4.2.1 Simplifying our goals

The main difficulty in proving Theorem 4.4 is that the addition of the new pair (s, t) may change the behavior of primal-dual algorithms for Steiner forest in fairly unpredictable ways (cf., the simpler single-sink case discussed in Subsection 2.4). In particular, it is difficult to argue about the relationship between the two algorithms we care about: (1) the algorithm $\mathsf{Timed}(G, D, \gamma T_{\mathcal{D}})$ that gives us the forest $F_{\mathcal{D}}$, and (2) the algorithm $\mathsf{GW}(G, \mathcal{D}') = \mathsf{Timed}(G, D', T_{\mathcal{D}'})$ that gives us the cost-shares. The difficulty of understanding the sensitivity of primal-dual algorithms to small perturbations of the input is well known, and has been studied in detail in other contexts by Garg [12] and Charikar and Guha [10].

In this section, we apply some transformations to the input data to partially avoid the detailed analyses of [10, 12]. In particular, we will obtain a new graph H from G (with analogous demand sets D_H and D'_H), as well as a new time vector T_H so that it now suffices to relate (1') the algorithm $\mathsf{Timed}(H, D_H, \gamma T_H)$ and (2') the algorithm $\mathsf{Timed}(H, D'_H, T_H)$. In the rest of this section, we will define the new graph and time vector; Section 4.2.2 will show that this transition is kosher, and then Sections 4.2.3 and C.1 will complete the argument.

A simpler time vector T: To begin, let us note that the time vectors $T_{\mathcal{D}}$ and $T_{\mathcal{D}'}$ may be very different, even though the two are obtained from instances that differ only in the presence of the pair (s, t). However, a monotonicity property *does* hold till time $T_{\mathcal{D}'}(s) = T_{\mathcal{D}'}(t)$, as the following lemma shows:

Lemma 4.5 *The set of tight edges at time $\tau \leq T_{\mathcal{D}'}(s)$ during the run of the algorithm $\mathsf{GW}(G, \mathcal{D})$ is a subset of the set of tight edges at the same time in $\mathsf{GW}(G, \mathcal{D}')$.*

Corollary 4.6 *Let T be the vector obtained by truncating $T_{\mathcal{D}'}$ at time $T_{\mathcal{D}'}(s)$, i.e., $T(j) = \min(T_{\mathcal{D}'}(j), T_{\mathcal{D}'}(s))$. Then for all demands $j \in D$, $T(j) \leq T_{\mathcal{D}}(j)$.*

The vector T is now a time vector for which we can say something interesting for both runs. Suppose we were to now run the algorithm $\mathsf{Timed}(G, D, \gamma T)$ as the second step of $\mathsf{PD}(G, \mathcal{D})$ (instead of $\mathsf{Timed}(G, D, \gamma T_{\mathcal{D}})$ prescribed by the algorithm). By the monotonicity results in Lemma 4.1 and Corollary 4.6, the edges that are made tight in $\mathsf{Timed}(G, D, \gamma T)$ are a subset of those in $\mathsf{Timed}(G, D, \gamma T_{\mathcal{D}})$. Hence it suffices to show that the distance between s and t in the forest resulting from $\mathsf{Timed}(G, D, \gamma T)$ is small; this is made precise by the following construction.

A simpler graph H: Let us look at the equivalence relation defined by the run of $\mathsf{Timed}(G, D, \gamma T)$ over the demands, which we shall denote by $\mathcal{R}$. (Recall that for $j_1, j_2 \in D$, $(j_1, j_2) \in \mathcal{R}$ if at some time τ during the run, they are both active and the clusters containing them meet. Equivalently, at some time τ, both j_1 and j_2 are active and some path between them becomes tight.) Similarly, let the equivalence relation $\mathcal{R}_D$ be obtained by running $\mathsf{Timed}(G, D, \gamma T_{\mathcal{D}})$. Lemma 4.1 and Corollary 4.6 now imply that the former is a refinement of the latter, i.e.,

$\mathcal{R} \subseteq \mathcal{R}_D$. (Note that this also means that the equivalence classes of $\mathcal{R}_D$ can be obtained by taking unions of the equivalence classes of $\mathcal{R}$.)

Since $F_\mathcal{D}$ connects all the demands that lie in the same equivalence class of $\mathcal{R}_D$, the fact that $\mathcal{R} \subseteq \mathcal{R}_D$ implies that it connects up all demands in the same equivalence class in $\mathcal{R}$ as well. Hence, to show Theorem 4.4 that there is a short s-t path in G/F_D it suffices to show the following result.

Theorem 4.7 (Strictness restated) *Let H be the graph obtained from G by identifying all the vertices that lie in the same equivalence class of $\mathcal{R}$. Then the distance between s and t in H is at most $\beta(\chi(\mathcal{D}', s) + \chi(\mathcal{D}', t))$.*

4.2.2 Relating the runs on G and H

We will need some new (but fairly obvious) notation: note that each vertex in H either corresponds to a single vertex in G, or to a subset of the demands D that formed an equivalence class of $\mathcal{R}$. The vertices of the latter type are naturally called the demands in H, and denoted by D_H. The set D'_H is just $D_H \cup \{s,t\}$. Each demand $j \in D'_H$ has a set $C_j \subseteq D'$ of demands that were identified to form j. A new time-vector T_H is defined by setting $T_H(s) = T_H(t) = T(s) = T(t)$; furthermore, for $j \in D_H$, we set $T_H(j) = \max_{x \in C_j} T(x)$.

Going from $\mathsf{Timed}(G, D, \gamma T_\mathcal{D})$ to $\mathsf{Timed}(H, D_H, \gamma T_H)$: Note that H was obtained from G by identifying some demands; the edge sets of G and H are exactly the same. We now show they *behave* identically as well. Let us denote the execution of $\mathsf{Timed}(H, D_H, \gamma T_H)$ by $\mathcal{E}$.

Lemma 4.8 *The set of tight edges in the two runs $\mathsf{Timed}(G, D, \gamma T)$ and $\mathcal{E}$ at any time τ are the same.*

Corollary 4.9 *For all $\gamma \geq 1$, an active cluster at time τ during the run of $\mathcal{E}$ contains only one active demand. In particular, two active clusters never merge during the entire run of the algorithm.*

Note that this implies that the run $\mathcal{E}$ is very simple: each demand $j \in D_H$ grows a cluster around itself; though this cluster may merge with inactive clusters, it never merges with another active one.

Going from $\mathsf{Timed}(G, D', T_{\mathcal{D}'})$ to $\mathsf{Timed}(H, D'_H, T_H)$: Recall that the cost-shares $\chi(\mathcal{D}', s)$ and $\chi(\mathcal{D}', t)$ are defined by running $\mathsf{GW}(G, \mathcal{D}') = \mathsf{Timed}(G, D', T'_\mathcal{D})$ and using the formula (3.5). The following lemma shows that it suffices to look instead at $\mathsf{Timed}(H, D'_H, T_H)$ in order to define the cost-share. (Let $\mathcal{E}'$ be short-hand for the execution $\mathsf{Timed}(H, D'_H, T_H)$.)

Lemma 4.10 *In $\mathcal{E}'$, if s (respectively, t) is the only active vertex in its cluster at time τ, then $a(s, \tau) = 1$ (resp., $a(t, \tau) = 1$) in the run $\mathsf{Timed}(G, D', T_{\mathcal{D}'})$.*

Corollary 4.11 *Let $\mathsf{alone}(s)$ be the total time in the run $\mathcal{E}'$ during which s is the only active vertex in its cluster, and define $\mathsf{alone}(t)$ analogously. Then $\chi(\mathcal{D}', s) + \chi(\mathcal{D}', t) \geq \mathsf{alone}(s) + \mathsf{alone}(t)$.*

A property similar to Corollary 4.9 can also be proved about the run $\mathcal{E}'$.

Lemma 4.12 *Let τ_{st} be the time at which s and t become part of the same cluster in $\mathcal{E}'$. Any active cluster at time $\tau \leq \tau_{st}$ in $\mathcal{E}'$ contains at most one active demand from D_H, and at most one active demand from the set $\{s, t\}$.*

4.2.3 The path between s and t

Lemma 4.13 *The vertices s and t lie in the same tree in the Steiner forest output by $\mathcal{E}'$.*

This simple lemma (whose proof is in the Appendix) gives us a path $\mathbb{P}$ between s and t in H whose length we will argue about. Note that this path is already formed at time τ_{st}, and hence the rest of the argument can (and will) be done truncating the time vector at time τ_{st} instead of at $T_H(s)$.

A useful fact to remember is that all edges in $\mathbb{P}$ must be tight in the run of $\mathcal{E}'$. The proof of the following theorem, along with Corollary 4.11, will complete the proof of Theorem 4.7.

Theorem 4.14 (Strictness restated again) *The length of the path $\mathbb{P}$ is bounded by $\beta(\mathsf{alone}(s) + \mathsf{alone}(t))$.*

Due to space constraints, we defer the proof of Theorem 4.14 (with $\beta = 6\gamma/(2\gamma - 3)$) to Appendix C.

References

[1] Ajit Agrawal, Philip Klein, and R. Ravi. When trees collide: an approximation algorithm for the generalized Steiner problem on networks. *SIAM J. Comput.*, 24(3):440–456, 1995. (Preliminary version in *23rd STOC*, 1991).

[2] Matthew Andrews and Lisa Zhang. Approximation algorithms for access network design. *Algorithmica*, 34(2):197–215, 2002. (Preliminary version in 39th FOCS, 1998.).

[3] Baruch Awerbuch and Yossi Azar. Buy-at-bulk network design. In *Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science*, pages 542–547, 1997.

[4] Baruch Awerbuch, Yossi Azar, and Yair Bartal. On-line generalized Steiner problem. In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (Atlanta, GA, 1996)*, pages 68–74, New York, 1996. ACM.

[5] Yair Bartal. *Competitive Analysis of Distributed On-line Problems — Distributed Paging*. PhD thesis, Tel-Aviv University, Israel, 1994.

[6] Yair Bartal. On approximating arbitrary metrics by tree metrics. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 161–168, 1998.

[7] Yair Bartal, Moses Charikar, and Piotr Indyk. On page migration and other relaxed task systems. In *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 43–52, 1997.

[8] Yair Bartal, Amos Fiat, and Yuval Rabani. Competitive algorithms for distributed data management. *J. Comput. System Sci.*, 51(3):341–358, 1995. (Preliminary version in 24th STOC, 1992).

[9] M. Bern and P. Plassman. The Steiner problem with edge lengths 1 and 2. *Information Processing Letters*, 32:171–176, 1989.

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

[11] Chandra Chekuri, Sanjeev Khanna, and Joseph Naor. A deterministic algorithm for the cost-distance problem. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 232–233, 2001.

[12] N. Garg. A 3-approximation for the minimum tree spanning k vertices. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, pages 302–309, 1996.

[13] Naveen Garg, Rohit Khandekar, Goran Konjevod, R. Ravi, F. Sibel Salman, and Amitabh Sinha. On the integrality gap of a natural formulation of the single-sink buy-at-bulk network design formulation. In *Proceedings of the 8th Integer Programming and Combinatorial Optimization Conference*, volume 2081 of *Lecture Notes in Computer Science*, pages 170–184, 2001.

[14] Ashish Goel and Deborah Estrin. Simultaneous optimization for concave costs: Single sink aggregation or single source buy-at-bulk. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 499–505, 2003.

[15] Michel X. Goemans and David P. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Comput.*, 24(2):296–317, 1995. (Preliminary version in *5th SODA*, 1994).

[16] Sudipto Guha, Adam Meyerson, and Kamesh Munagala. Hierarchical placement and network design problems. In *Proceedings of the 41th Annual IEEE Symposium on Foundations of Computer Science*, pages 603–612, 2000.

[17] Sudipto Guha, Adam Meyerson, and Kamesh Mungala. A constant factor approximation for the single sink edge installation problems. In *Proceedings of the 33rd Annual ACM Symposium on the Theory of Computing (STOC)*, pages 383–388, 2001.

[18] Anupam Gupta, Amit Kumar, Jon Kleinberg, Rajeev Rastogi, and Bülent Yener. Provisioning a Virtual Private Network: A network design problem for multicommodity flow. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, pages 389–398, 2001.

[19] Anupam Gupta, Amit Kumar, and Tim Roughgarden. Simpler and better approximation algorithms for network design. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, 2003. To appear.

[20] Kamal Jain and Vijay Vazirani. Applications of approximation algorithms to cooperative games. In *Proceedings of the 33rd Annual ACM Symposium on the Theory of Computing (STOC)*, pages 364–372, 2001.

[21] David R. Karger and Maria Minkoff. Building Steiner trees with incomplete global knowledge. In *Proceedings of the 41th Annual IEEE Symposium on Foundations of Computer Science*, pages 613–623, 2000.

[22] Samir Khuller and An Zhu. The general Steiner tree-star problem. *Information Processing Letters*, 84:215–220, 2002.

[23] Tae Ung Kim, Timothy J. Lowe, Arie Tamir, and James E. Ward. On the location of a tree-shaped facility. *Networks*, 28(3):167–175, 1996.

[24] Amit Kumar, Anupam Gupta, and Tim Roughgarden. A constant-factor approximation algorithm for the multicommodity rent-or-buy problem. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science*, pages 333–342, 2002.

[25] Martine Labbé, Gilbert Laporte, Inmaculada Rodrígues Martín, and Juan José Salazar González. The median cycle problem. Technical Report 2001/12, Department of Operations Research and Multicriteria Decision Aid at Université Libre de Bruxelles, 2001.

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

[27] Adam Meyerson, Kamesh Munagala, and Serge Plotkin. Cost-distance: Two metric network design. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science*, pages 624–630, 2000.

[28] Adam Meyerson, Kamesh Munagala, and Serge Plotkin. Designing networks incrementally. In *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science*, pages 406–415, 2001.

[29] Robert C. Prim. Shortest interconnection networks and some generalizations. *Bell System Technical Journal*, 36:1389–1401, 1957.

[30] R. Ravi and F. S. Salman. Approximation algorithms for the traveling purchaser problem and its variants in network design. In *Algorithms—ESA '99 (Prague)*, volume 1643 of *Lecture Notes in Comput. Sci.*, pages 29–40. Springer, Berlin, 1999.

[31] Gabriel Robins and Alexander Zelikovsky. Improved Steiner tree approximation in graphs. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 770–779, 2000.

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

[33] P. D. Straffin. *Game Theory and Strategy*. Mathematical Association of America, 1993.

[34] Chaitanya Swamy and Amit Kumar. Primal-dual algorithms for the connected facility location problem. In *Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, volume 2462 of *Lecture Notes in Computer Science*, pages 256–269, 2002.

[35] Kunal Talwar. Single-sink buy-at-bulk LP has constant integrality gap. In *Proceedings of the 9th Integer Programming and Combinatorial Optimization Conference*, volume 2337 of *Lecture Notes in Computer Science*, pages 475–486, 2002.

[36] Vijay V. Vazirani. *Approximation algorithms*. Springer-Verlag, Berlin, 2001.

[37] H. P. Young. Cost allocation. In R. J. Aumann and S. Hart, editors, *Handbook of Game Theory*, volume 2, chapter 34, pages 1193–1235. North-Holland, 1994.

A Non-Uniform Demands

We assume that the amount d_j of flow that must be sent from s_j to t_j is rational from every j; by scaling, we can then assume all d_j's to be integral. We then modify Step (A1) of algorithm SimpleMROB so that d_j coins are flipped for the pair (s_j, t_j); the demand pair is marked if at least one coin reads heads. Conceptually, we are replacing (s_j, t_j) by d_j co-located demand pairs, each of whom must send one unit of flow. Since this is equivalent to flipping a coin for (s_j, t_j) that comes up heads with probability $1 - (1 - 1/M)^{d_j}$, this process can be implemented efficiently even when demand weights are not polynomially bounded.

B Proofs for Section 4

Proof of Lemma 4.5: Since $\tau \le T_{\mathcal{D}'}(s)$, both s and t are active at time τ in $\mathsf{GW}(G, \mathcal{D}')$. Any cluster that has not merged yet with clusters containing s or t has the same behavior in both runs. A cluster that merges with a cluster containing s or t will continue to grow. So compared with $\mathsf{GW}(G, \mathcal{D})$, only more edges can get tight in $\mathsf{GW}(G, \mathcal{D}')$. ■

Proof of Corollary 4.6: If $T_{\mathcal{D}'}(s) \le T_{\mathcal{D}}(j)$, the claim clearly holds. If $T_{\mathcal{D}}(j) < T_{\mathcal{D}'}(s)$, then there is a tight path from j to its partner at time $T_{\mathcal{D}}(j)$. But the monotonicity Lemma 4.5 implies that these edges must be tight in $\mathsf{GW}(G, \mathcal{D}')$ at time $T_{\mathcal{D}}(j)$ as well, and hence $T(j) = T_{\mathcal{D}'}(j) \le T_{\mathcal{D}}(j)$. ■

The following monotonicity lemma will be used in the following proofs, and we omit the simple proof here:

Lemma B.1 *For all $\gamma \geq 1$, the set of tight edges in* $\mathsf{Timed}(H, D_H, \gamma T_H)$ *contains the tight edges in* $\mathsf{Timed}(G, D, \gamma T)$. *Similarly, the tight edges of* $\mathsf{Timed}(H, D'_H, \gamma T_H)$ *contain those in* $\mathsf{Timed}(G, D', \gamma T)$.

Proof of Lemma 4.8: By Lemma B.1, we know that the latter edges contain the former; we just have to prove the converse. For a demand $j \in D_H$, we claim that each demand $x \in C_j$ is in some active cluster of $\mathsf{Timed}(G, D, \gamma T)$ till time $\gamma T_H(j)$. Suppose not: let $x \in C_j$ be in an inactive cluster at time $\tau < \gamma T_H(j)$. By the definition of the equivalence relation $\mathcal{R}$, no more demands are added to the equivalence class of x (which is C_j). But then $\max_{y \in C_j} \gamma T(y) \leq \tau < \gamma T_H(j)$, a contradiction. Hence all demands in C_j are in active clusters till time $\gamma T_H(j)$. Thus, in the run $\mathsf{GW}(G, D, \gamma T)$, we had grown regions around these demands till time τ, giving us the same tight edges as in $\mathcal{E}$. ∎

Proof of Corollary 4.9: Suppose $j \neq j' \in D_H$ are two active demands in the same (active) cluster at time τ. Let C_j and $C_{j'}$ be the corresponding sets of demands in D respectively. By Lemma 4.8, the set of edges that become tight at time τ is the same as that in $\mathsf{Timed}(G, D, \gamma T)$, and hence there must be demands $x \in C_j, x' \in C_{j'}$ which lie in the same active cluster of $\mathsf{Timed}(G, D, \gamma T)$ at time τ. By an argument identical to that in the proof of the previous lemma, there must be active demands $y \in C_j$ and $y' \in C_{j'}$ in the same cluster as well. Hence $C_j = C'_j$ by the definition of the equivalence class, contradicting that $j \neq j'$. ∎

Proof of Lemma 4.10: Suppose $a(s, \tau) = 0$ in $\mathsf{Timed}(G, D', T_{\mathcal{D}'})$, and some active demand $j \neq s$ lies in s's cluster at time τ. By the definition of T, j and s are both active in the same cluster in $\mathsf{Timed}(G, D', T)$ as well. Now by Lemma B.1, j and s must lie in the same cluster in $\mathcal{E}'$ too; furthermore, they must both be active (by the definition of T_H). This contradicts the assumption of the theorem. ∎

Proof of Lemma 4.12: Let C be an active cluster in $\mathcal{E}'$ at time τ with two active demands $j, j' \in D_H$. If C contains neither s nor t, then C is also a cluster (with two active demands) in $\mathcal{E}$ at the same time τ, which contradicts the implication of Corollary 4.9.

Hence C must contain one of s or t (in addition to j, j'); it cannot contain both since $\tau \leq \tau_{st}$. Suppose it has s; we claim that we can prove that j and j' must lie in the same cluster in $\mathcal{E}$ at time $2\,\tau$. (For this to make sense, we have to assume that $\gamma \geq 2$.) To prove this, consider a tight path between s and j in $\mathcal{E}'$ at time τ—since they both lie in the same cluster, such a path must exist. Since s has been active for time τ, the portion of this path which is tight due to a s's cluster has length at most τ. Now consider the same path in $\mathcal{E}$ at time τ — all but a portion of length at most τ of this path must already be tight. (Note that though we have dropped both s and t, we lose only τ, since no part of the path could have been made tight by t's cluster. If this were to happen, then $\tau_{st} < \tau$, which is assumed not to happen.) Hence the cluster containing j will contain s after another τ units of time. The same argument can be made for j'. But this would imply that j and j' would lie in the same cluster in $\mathcal{E}$ at time 2τ, contradicting Corollary 4.9. ∎

Proof of Lemma 4.13: By the definition of T, s and t lie in the same cluster at time $T(s) = T(t)$. Since $T_H(s) = T_H(t) = T(s)$, applying Lemma B.1 implies that s and t lie in the same cluster in $\mathcal{E}'$ as well. Since they are both active at the time their clusters merge, they lie in the same connected component of the Steiner forest. ∎

C Proof of Theorem 4.14

C.1 The length of the path $\mathbb{P}$ is small

Before we proceed, here is some more syntactic sugar:

Given an execution of an algorithm, a *layer* is a tuple $(C, I = [\tau_1, \tau_2))$, where $C \subseteq V$ is an active cluster between times τ_1 and τ_2. If $I = [\tau_1, \tau_2)$ is a time interval, its *thickness* is $\tau_2 - \tau_1$. A *layering* $\mathcal{L}$ of an execution is a set of layers such that, for every time τ and every active cluster C, there is exactly one layer $(C, I) \in \mathcal{L}$ such that $\tau \in I$.

Given any layering $\mathcal{L}$, another layering $\mathcal{L}'$ can be obtained by *splitting* some layer $(C, I) \in \mathcal{L}$—i.e., replacing it with two layers $(C, [\tau_1, \hat{\tau})), (C, [\hat{\tau}, \tau_2))$ for some $\hat{\tau} \in (\tau_1, \tau_2)$. Conversely, two layers with the same set C and consecutive time intervals can be merged. It is easy to see that given some layering of an execution, all other layerings of the same execution can be obtained by splittings and mergings.

We fix layerings of the two executions $\boldsymbol{\mathcal{E}}$ and $\boldsymbol{\mathcal{E}}'$, which we denote by $\mathcal{L}$ and $\mathcal{L}'$ respectively. The only property we desire of these layerings is this: if $(C, I) \in \mathcal{L}$ and $(C', I') \in \mathcal{L}'$ with $1/\gamma I \cap I' \neq \emptyset$, then $I = \gamma I'$. (I.e., $I = [\gamma\tau_1, \gamma\tau_2)$ and $I' = [\tau_1, \tau_2)$.) It is easy to see that such a condition can be satisfied by making some splittings in $\mathcal{L}$ and $\mathcal{L}'$.

Lemma 4.12 implies that each active layer $(C, I) \in \mathcal{L}'$ in categorized thus:

- **lonely:** The only active demand in C is either s and t. *Assign* the layer to that demand.
- **shared:** C contains one active demand from D_H **and** one of s and t. The layer is *assigned* to the active demand from D_H.
- **unshared:** The only active demand in C is from D_H. Again, the layer is *assigned* to that demand.

Note that the total thickness of the lonely layers is a lower bound on $\mathsf{alone}(s) + \mathsf{alone}(t)$. Furthermore, since $\mathbb{P}$ consists only of tight edges in $\boldsymbol{\mathcal{E}}'$, the length of $\mathbb{P}$ is exactly the total thickness of the layers of $\mathcal{L}'$ that $\mathbb{P}$ crosses. If L, S, and U denote the total thickness of the alone, shared and unshared layers that $\mathbb{P}$ crosses, then

$$\mathsf{len}(\mathbb{P}) = L + S + U \tag{C.7}$$

Of course, any layer $(C, I) \in \mathcal{L}'$ that crosses $\mathbb{P}$ must have $I \subseteq [0, \tau_{st}]$, since the path is tight at time τ_{st}. Hence their corresponding layers have time intervals that lie in $[0, \gamma\tau_{st}]$.

Note that the total thickness of the layers of $\mathcal{L}$ that $\mathbb{P}$ crosses is a lower bound on its length. This suggests the following plan for the rest of the proof: for each shared or unshared layer in $\mathcal{L}'$, there is a corresponding distinct layer in $\mathcal{L}$ that is γ times thicker. In an ideal world, each crossing of a layer in $\mathcal{L}'$ would also correspond to a crossing of the corresponding layer in $\mathcal{L}$; in this case, $\gamma(S + U) \leq \mathsf{len}(\mathbb{P})$, and hence $L \geq \frac{\gamma-1}{\gamma}\mathsf{len}(\mathbb{P})$. Sadly, we do not live in an ideal world and the argument is a bit more involved than this, though not by much.

Mapping shared and unshared layers of $\mathcal{L}'$: Each such layer $\ell' = (C', I)$ is assigned to a demand $j \in D_H \cap C'$. Since j is active during the interval γI in $\boldsymbol{\mathcal{E}}$, there must be a layer $\ell = (C, \gamma I) \in \mathcal{L}$ containing j—this is defined to be the layer corresponding to ℓ'. (The properties of the layerings ensure that the two time intervals are just rescalings of each other by a factor γ.) Furthermore, since each layer in $\mathcal{L}$ contains only one active vertex, the mapping is one-one. It remains to show that crossings of $\mathbb{P}$ by layers is preserved (approximately) by this correspondence.

Lemma C.1 *Each* unshared layer $\ell' = (C', I = [\tau_1, \tau_2))$ *is crossed by* $\mathbb{P}$ *either zero or two times. Furthermore, if its corresponding layer is* $\ell = (C, \gamma I)$*, then* $C' \subseteq C$.

Proof: Since $\mathbb{P}$ begins and ends outside ℓ', it must cross ℓ' an even number of times. Furthermore, $\mathbb{P}$ cannot cross ℓ' more than twice—if it does so, there must exist vertices j_1, j_2, and j_3 visited by $\mathbb{P}$ (in order), such that $j_1, j_3 \in C'$, but $j_2 \notin C'$. However, our algorithms ensure that if j_1 and j_3 lie in the same cluster C', then the edges joining them lie within C' as well. This implies that there must be two disjoint paths between j_1 and j_2, contradicting that the algorithms construct a forest.

For the second part, note that if C' is assigned to j and does not contain s or t at time τ_2, then the cluster containing j at time $\gamma\tau_2$, i.e., C must contain C'. ∎

Lemma C.2 *If $\ell' = (C', I = [\tau_1, \tau_2))$ is a* shared layer *containing s (resp., t) which assigned to j, and $\gamma \geq 2$, then its corresponding layer $\ell = (C, \gamma I)$ contains s (resp., t).*

Proof of Lemma C.2: The proof of the first claim is similar to that of Lemma 4.12; we just sketch the idea again. Consider a tight path between j and s at time τ_1; at most a τ_1 portion of it can be tight due to s. Hence by time $\gamma\tau_1 \geq 2\tau_1$, the path must be completely tight. ■

C.2 Finally, the book-keeping

Note that both s and t both contribute separately to S and A till time τ_{st}, and hence

$$S + L = 2\,\tau_{st}. \tag{C.8}$$

Let $\ell' = (C', I) \in \mathcal{L}'$ be an unshared layer that $\mathbb{P}$ crosses, and let its corresponding layer be $\ell = (C, \gamma I) \in \mathcal{L}$. If $v \in \mathbb{P} \cap C'$ is a vertex on the path, then $v \in \mathbb{P} \cap C$ by Lemma C.1. If both s and t are outside C, then P crosses C twice as well, and we get a contribution of 2γ times the thickness of ℓ'. Suppose not, and $s \in C$ or $t \in C$: then we lose γ times the thickness of ℓ for each such infringement. For shared layers ℓ' (which map to $\ell \in \mathcal{L}$), Lemma C.2 implies that we lose only when s and t both lie inside ℓ. Hence, if W denotes the wastage, the

$$\mathsf{len}(\mathbb{P}) \geq \gamma(S + U) - W. \tag{C.9}$$

We bound the wastage in two ways: firstly, the wastage only occurs when s or t lie in layers in $\mathcal{L}$ they are "not supposed to". However, s and t can only lie in one layer at a time, and hence they can only waste $\gamma\,\tau_{st}$ amount each, which by (C.8) gives

$$W \leq \gamma(S + L). \tag{C.10}$$

For another bound, let $\gamma\,\tau_{meet}$ be the earliest time that s and t lie in the same layer in $\mathcal{L}$ (not to be confused with τ_{st}, which is the earliest time they lie in the same layer in $\mathcal{L}'$). We can be very pessimistic and claim that all the contribution due to the unshared layers (i.e., $\gamma\,U$) is lost. A shared layer $\ell' = (C', I)$ (containing s, say) mapped to $\ell = (C, \gamma I)$ can incur loss only when t also lies in C—and hence $I \subseteq [\tau_{meet}, \tau_{st}]$. But now we claim that t must have been lonely. Indeed, suppose not, and that t was sharing layer $\ell' \in \mathcal{L}'$ assigned to j, and ℓ' was assigned to j'. But then both these layers would be mapped to ℓ, contradicting the fact that the mapping was one-one.

Hence we can charge the loss for each shared layer to a lonely layer, and bound the wastage by:

$$W \leq \gamma(U + L). \tag{C.11}$$

Averaging (C.10) and (C.11), and plugging into (C.9), we get

$$\mathsf{len}(\mathbb{P}) \leq \frac{\gamma}{2}(S + U - 2L). \tag{C.12}$$

Subtracting this from $\frac{\gamma}{2}$ times (C.7), we get

$$\mathsf{alone}(s) + \mathsf{alone}(t) \geq L \geq \frac{3\gamma}{\gamma - 2}\mathsf{len}(\mathbb{P}), \tag{C.13}$$

proving Theorem 4.14 with $\beta = \frac{3\gamma}{\gamma-2}$. The calculations of the next subsection show how to improve this to $\beta = \frac{6\gamma}{2\gamma-3}$.

C.3 Tightening the constants

One can get a better bound than given in (C.13). Suppose we were split each of the various quantities L, S, U, W in two (L_1, L_2 etc.) to account for layers of $\mathcal{L}'$ that correspond to layers before and after time τ_{meet}. Then, analogous to (C.10), we have

$$W_1 \leq \gamma(S_1 + L_1) \tag{C.14}$$
$$W_2 \leq \gamma(S_2 + L_2) \tag{C.15}$$

The reasoning of (C.11) implies $W_2 \leq \gamma\,(U_2 + L_2)$. However, before time τ_{meet}, we can do the following argument: each unshared layer gets mapped to a layer that can contain only one of s or t, and hence loses at most one crossing with $\mathbb{P}$ (while each shared layer loses no crossings at all), giving us that

$$W_1 \leq \gamma\,U_1/2 \tag{C.16}$$
$$W_2 \leq \gamma\,(U_2 + L_2) \tag{C.17}$$

Multiplying (C.14-C.17) by $\frac{1}{3}, \frac{2}{3}, \frac{2}{3}, \frac{1}{3}$ respectively, and adding, we get

$$W = W_1 + W_2 \leq \frac{\gamma}{3}[S_1 + 2S_2 + L_1 + 2L_2 + U_1 + U_2].$$

Now using the fact that $S_2 \leq L_2$, which follows from the basic argument behind (C.11), and algebra, we get

$$\mathsf{alone}(s) + \mathsf{alone}(t) \geq L \geq \frac{6\gamma}{2\gamma - 3}\mathsf{len}(\mathbb{P}) \tag{C.18}$$