Approximation via Cost Sharing: Simpler and Better Approximation Algorithms for Network Design*

Anupam Gupta† Amit Kumar‡ Martin Pál § Tim Roughgarden¶

November 30, 2005

Abstract

We present constant-factor approximation algorithms for several widely-studied NP-hard optimization problems in network design, including the multicommodity rent-or-buy, virtual private network design, and single-sink buy-at-bulk problems. Our algorithms are simple and their approximation ratios improve over those previously known, in some cases by orders of magnitude.

We develop a general analysis framework to bound the approximation ratios of our algorithms. This framework is based on a novel connection between random sampling and game-theoretic cost sharing. 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.

*Preliminary versions of these results appeared in the Proceedings of the 35th Annual Symposium on Theory of Computing, June 2003; the Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science, October 2003; and the Proceedings of the 32nd Annual International Colloquium on Automata, Languages, and Programming, July 2005.

†Department of Computer Science, Carnegie Mellon University, Pittsburgh PA 15213. Part of this research was done when the author was at Lucent Bell Laboratories, Murray Hill, NJ. This research is also supported in part by NSF CAREER award CCF-0448095 and by an Alfred P. Sloan Fellowship. Email: anupamg@cs.cmu.edu.

‡Department of Computer Science and Engineering, IIT Delhi, India 110016. Part of this research was done while the author was at Cornell University and Lucent Bell Laboratories, Murray Hill, NJ. Email: amitk@cse.iitd.ernet.in.

§Google, Inc., 1440 Broadway, New York, NY. Part of this research was done while the author was at Cornell University and supported by ONR grant N00014-98-1-0589. Email: mpal@acm.org.

¶Department of Computer Science, Stanford University, 462 Gates Building, Stanford CA 94305. Supported in part by ONR grant N00014-04-1-0725, DARPA grant W911NF-04-9-0001, and an NSF CAREER Award. Part of this research was done while the author was at Cornell University. Email: tim@cs.stanford.edu.

1 Introduction

We present constant-factor approximation algorithms for several widely-studied NP-hard optimization problems in network design. Our algorithms are extremely simple and have the following flavor: randomly sample a simpler subproblem, solve the subproblem with an existing algorithm, and greedily extend the subproblem solution to a solution feasible for the original problem. The approximation ratios of our algorithms improve over all of those previously known, in some cases by orders of magnitude.

We develop a general analysis framework to bound the approximation ratios of our algorithms. This framework is based on a novel connection between random sampling and *cost sharing*, the task of allocating the cost of an object to many users of the object in a "fair" manner. Specifically, we define the notion of *strict* cost shares, and show that such cost shares provide a powerful tool for analyzing the performance of a class of random sampling algorithms. 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.

1.1 Four Network Design Problems

To describe our results more concretely, we define the three primary network design problems that we consider in this paper. We discuss the motivation for and prior work on these problems in Subsection 1.3 below.

Problem 1.1 (Multicommodity Rent-or-Buy) An instance of the *multicommodity rent-or-buy* (MRoB) problem is defined by an undirected graph $G = (V, E)$ and a set $\mathcal{D} = \{(s_i, t_i)\}_{i=1}^{k}$ of vertex pairs called *demand pairs*, where each edge $e \in E$ has a nonnegative cost c_e and each demand pair (s_i, t_i) has a nonnegative weight w_i. The goal is to compute a minimum-cost way of installing sufficient capacity on the edges E so that w_i units 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 infinite capacity equal to M (per unit length). By scaling, we can assume that $\mu = 1$ without loss of generality. We denote an MRoB instance by a tuple $(G, \mathcal{D}, w, M)$, leaving the cost vector c implicit.

We will also study the special case of *single-sink rent-or-buy* (SSRoB), where all demand pairs (s_i, t_i) share a common sink vertex t, and the more general *multicast rent-or-buy* problem (MuRoB), where there are arbitrary demand groups instead of demand pairs.

Problem 1.2 (Virtual Private Network Design) In an instance of *virtual private network design* (VPND), we are again given an undirected graph G with nonnegative edge costs c. There is also a set D of *demands*, each of which is located at a vertex of G. Each demand

$j \in D$ possesses two nonnegative *thresholds* $b_{in}(j)$ and $b_{out}(j)$. These thresholds specify the maximum amount of traffic that demand j will receive from and send to other demands, respectively. A $D \times D$ matrix describing the amount of (directed) traffic between each pair of demands is *valid* if it respects all thresholds. A feasible solution to an instance of VPND is specified by a path P_{ij} for each (ordered) demand pair (i, j) and by a capacity u_e for each edge e, such that there is sufficient capacity to route every valid traffic matrix via the paths $\{P_{ij}\}$. The objective is to find a feasible solution minimizing the cost $\sum_{e \in E} c_e u_e$. We denote an instance of VPND by the triple (G, D, b).

Problem 1.3 (Single-Sink Buy-at-Bulk) The *single-sink buy-at-bulk network design* (SS-BaB) problem is a generalization of the SSRoB problem. The input is the same as in the latter problem, except that instead of a single parameter M describing the cost of buying, there are K types of *cables*. A cable of type i has a given capacity u_i and a given cost (per unit length) σ_i. As in the SSRoB problem, the goal is to compute a minimum-cost way of installing sufficient capacity on the edges so that a prescribed amount of flow w_i can be sent simultaneously from each source s_i to the common sink t.

The following simpler network design problem arises frequently as a subroutine in our algorithms.

Problem 1.4 (Steiner Forest) An instance of the Steiner Forest problem is given by an undirected graph G with nonnegative edge costs c and a set $\mathcal{D} = \{(s_i, t_i)\}_{i=1}^{k}$ of demand pairs. The goal is to compute a minimum-cost subgraph of G that contains an s_i-t_i path for every $i \in \{1, 2, \ldots, k\}$. We denote such a Steiner Forest instance by $(G, \mathcal{D})$.

The Steiner Forest problem is equivalent to the special case of the MRoB problem where $M = 1$. If all demand pairs of a Steiner Forest instance have a common sink, then it is equivalent to an instance of the well-known Steiner Tree problem. All of the problems studied in this paper contain Steiner Tree as a special case.

Recall that an *α-approximation algorithm* for a minimization problem runs in polynomial time and returns a solution no more than α times as costly as an optimal solution. The value α is the *approximation ratio* or *performance guarantee* of the algorithm. Since even the Steiner Tree problem is MAX-SNP-hard [13], Problems 1.1–1.3 cannot be solved exactly or approximated to within an arbitrarily small constant factor in polynomial time, assuming $P \neq NP$ [4]. We are therefore justified in seeking constant-factor approximation algorithms for these problems, with the constant as small as possible.

1.2 Overview of Results

Our main results are the following.

- We develop an analysis framework that shows that random sampling, a Steiner Forest subroutine, and greedy augmentation leads to a constant-factor approximation algorithm for the MRoB problem, provided the subroutine admits what we call strict cost shares (defined in Section 2).

Problem Studied	Previously Best Approximation	This Paper
MRoB	over 1000 [50]	6.83
MuRoB	$O(\log n)$ [5, 23]	12.66
SSRoB	4.55 [60]	3.55
VPND	$O(\log n)$ [23, 33]	5.55
SSBaB	216 [61]	76.8

Table 1: Main results of this paper. "Previously best approximation" refers to the smallest approximation ratio known prior to the conference versions of our work [34, 35, 36]. The parameter n denotes the number of network vertices.

- We modify, in a simple but novel way, the well-known primal-dual Steiner Forest algorithm of Agrawal, Klein, and Ravi [1] and Goemans and Williamson [29] so that it admits strict cost shares. Combining this result with the one above, we obtain a randomized approximation algorithm for MRoB with an approximation ratio of $4+2\sqrt{2} \approx 6.83$.
- We extend this algorithm and analysis to obtain a 12.66-approximation algorithm for the MuRoB problem.
- For the SSRoB problem, we show that every Steiner Tree algorithm admits strict cost shares and obtain a randomized 3.55-approximation algorithm.
- For the VPND problem, we build on our SSRoB algorithm and analysis to obtain a randomized 5.55-approximation algorithm.
- We combine ideas from our SSRoB algorithm and analysis with an SSBaB algorithm of Guha, Meyerson, and Munagala [32] to obtain a randomized 76.8-approximation algorithm for the SSBaB problem.

Prior to our work, the best-known approximation ratios for the MRoB, MuRoB, SSRoB, VPND, and SSBaB problems were over 1000 [50]; $O(\log n)$, where n is the number of network vertices [5, 23]; 4.55 [60]; $O(\log n)$ [23, 33]; and 216 [61], respectively. See also Table 1. Our constant-factor approximation algorithm for the VPND problem answers the main open questions of Gupta et al. [33].

Finally, our 6.83-approximation algorithm for MRoB gives qualitatively new information about the relative tractability of different network design problems with economies of scale. Specifically, for many years even the simplest such problems with multiple commodities (like MRoB) seemed more difficult than relatively complex single-sink network design problems (such as SSBaB). Our MRoB algorithm shows that this state of affairs arose only because of a lack of a good algorithm for MRoB, not because of the problem's intrinsic difficulty.

1.3 Related Work

The literature on approximation algorithms for NP-hard network design problems is vast, and we will only discuss work that is directly related to the problems studied in this paper. In

this subsection, we only discuss research that occurred prior to or independent of the present work. Since the publication of preliminary versions of the results in this paper [34, 35, 36], there has been much research on further applications, generalizations, and improvements of our algorithms and analysis techniques. We survey this recent research in Section 6.

1.3.1 Rent-or-Buy Network Design

Rent-or-buy problems have long served as a simple model of network design with economies of scale—where the per-unit cost of installing capacity on an edge decreases as more capacity is installed. They also arise naturally in other applications, including stochastic optimization problems [45, 50] and facility location problems [50, 60].

For many years, the best algorithm known for the MRoB problem was an $O(\log n \log\log n)$-approximation algorithm, where n denotes the number of network vertices, due to Awerbuch and Azar [5] and Bartal [9]. (Recent work by Fakcharoenphol, Rao, and Talwar [23] can be used to improve the approximation ratio of this algorithm to $O(\log n)$.) The first constant-factor approximation algorithm for the problem is due to Kumar, Gupta, and Roughgarden [50]. However, both the analysis and the primal-dual algorithm of [50] are quite complicated, and the performance guarantee shown for the algorithm is, while constant, extremely large. This constant was neither optimized nor estimated in [50], but it is at least 1000. Our MRoB algorithm is the first constant-factor approximation algorithm for the problem that is simple or that has a reasonably small constant performance guarantee.

The SSRoB special case or MRoB, and the closely related *connected facility location* problem, have been extensively studied in the operations research literature [47, 51, 52] and by the computer science community [33, 45, 46, 56, 60]. Karger and Minkoff [45], motivated by the so-called *maybecast* problem, gave the first constant-factor approximation algorithm for the problem. This algorithm is simple and combinatorial, but has a relatively large performance guarantee. Gupta et al. [33] subsequently employed an LP-rounding approach to improve the approximation ratio. Prior to our work, the best algorithm for the problem was the primal-dual 4.55-approximation algorithm due to Swamy and Kumar [60].

Finally, our random sampling approach to the MRoB problem is reminiscent of and partially inspired by previous work that gave online algorithms with polylogarithmic competitive ratios for many rent-or-buy-type problems [6, 7, 10, 11].

1.3.2 Virtual Private Network Design

The virtual private network design problem considered in this paper was defined by Fingerhut et al. [24] and, subsequently and independently, by Duffield et al. [20]. The model is motivated by the many difficulties in estimating or assuming knowledge of a fixed traffic matrix for a network (see [20, 24]). The VPND problem was later studied by Gupta et al. [33] with an eye toward approximation algorithms.

Prior to our work, the best known algorithm for the VPND problem was a straightforward application of probabilistic tree embeddings [23], which only guarantees a $O(\log n)$-approximation, where n is the number of vertices. For the special case of VPND where $b_{in}(j) = b_{out}(j)$ for every demand $j \in D$, a 2-approximation is known [24, 33]. Also, Gupta

et al. [33] gave a 10-approximation algorithm for the special case of the VPND problem in which the the union of the routing paths $\{P_{ij}\}_{i,j \in D}$ is required to form a tree.

1.3.3 Buy-at-Bulk Network Design

Rent-or-buy problems are a special case of *buy-at-bulk network design*, where the goal is the same but the cost of installing capacity is given by an arbitrary concave function (or, nearly equivalently, by a set of cable types). Buy-at-bulk network design has been intensely studied over the last several years. After the problem was introduced by Salman et al. [59], a long line of papers have presented successively superior algorithms for increasingly general versions of the problem.

For the SSBaB problem considered here (Problem 1.3), the first non-trivial approximation was found by Awerbuch and Azar [5], using the tree embeddings of Bartal [8], and the first constant-factor approximation algorithm was given by Guha, Meyerson, and Munagala [32]. The performance guarantee of the combinatorial algorithm in [32] was not stated explicitly, though Talwar [61] estimated it to be roughly 2000. Talwar [61] subsequently gave an LP-rounding algorithm with an improved performance guarantee of 216, the best known before our work.

Many researchers have studied other types of single-sink network design problems with economies of scale, including the more specialized Access Network Design problem [3, 31, 32, 54], and the generalizations of SSBaB in which the capacity cost function can be edge-dependent [16, 53] or unknown to the algorithm [28]. The best known approximation ratios for these three problems are 68 [54], $O(\log n)$ [16, 53], and $O(\log n)$ [28], respectively. Recent results of Chuzhoy et al. [17] rule out constant-factor approximation algorithms for the second problem under reasonable complexity-theoretic assumptions.

For the multicommodity buy-at-bulk network design problem, the best known approximation ratio is $O(\log n)$, which follows from combining the algorithm of Awerbuch and Azar [5] with the probabilistic tree embeddings given by Fakcharoenphol, Rao, and Talwar [23]. Andrews [2] recently proved that, under reasonable complexity-theoretic assumptions, there is no constant-factor approximation algorithm for this problem. Very recently, Charikar and Karagiozova [15] gave the first non-trivial approximation algorithm for the generalization of this problem in which the concave capacity cost function can vary from edge to edge.

1.3.4 Steiner Forest

The first non-trivial approximation algorithm for the Steiner Forest problem was the 2-approximation algorithm due to Agrawal, Klein, and Ravi [1]. Subsequently, Goemans and Williamson [29, 30] reinterpreted the algorithm and analysis of [1], and generalized them to a wide class of network design problems. Very recently, Könemann, Leonardi, and Schäfer [49] gave a somewhat different 2-approximation algorithm for the Steiner Forest problem. Their algorithm is related to the Steiner Forest algorithm that we present in Subsection 3.3.

1.3.5 Cost Sharing

Cost sharing has long been a fundamental subject in game theory and economics; see e.g. [63] and the references therein. Our definition of strict cost-sharing methods in Section 2 is somewhat reminiscent of well-known concepts in cooperative game theory, including the *core* and the *nucleolus*. However, we are not aware of any work in the game theory literature that studies our notion of strict cost sharing.

Techniques from approximation algorithms have recently yielded new progress on several cost-sharing problems [39, 43, 44, 49, 55]. We believe the present work to be the first showing that ideas from cost sharing can lead to better approximation algorithms.

1.4 Paper Organization

Section 2 presents our analysis framework, defines strict cost shares, and proves that random sampling, a Steiner Forest subroutine that admits strict cost shares, and greedy augmentation leads to a constant-factor approximation algorithm for MRoB. Section 3 applies this framework to the SSRoB, MRoB, and MuRoB problems. In Section 4, we build on our SSRoB algorithm and analysis and design a constant-factor approximation algorithm for the VPND problem. Section 5 applies our analysis tools to the SSBaB problem. Sections 3–5 all logically depend on the concepts in Section 2. Sections 4 and 5 also depend on Subsection 3.1, though Sections 3–5 are otherwise independent. Finally, Section 6 discusses recent work motivated by this paper and possible directions for future research.

2 The Analysis Framework

This section describes our high-level algorithm and analysis framework for the MRoB problem. Subsection 2.1 presents our MRoB algorithm. Subsection 2.2 bounds its expected cost when solving a randomly sampled subproblem. Subsection 2.3 defines strict cost shares, and Subsection 2.4 uses them to bound the expected cost of the greedy augmentation step of our MRoB algorithm.

2.1 Random Sampling and Greedy Augmentation

Our algorithm for the MRoB problem is given in Figure 1. It first randomly samples a subset of demand pairs, with probabilities proportional to weights and inversely proportional to the ratio M of the buying and renting costs. It then buys capacity on edges so that each demand pair in the random sample is connected by an infinite-capacity path. Finally, our algorithm augments the capacity of the bought edges by greedily renting capacity for all demand pairs that did not participate in the random sample.

The sampling step in Figure 1 is self-explanatory. For the subproblem step, we will employ an algorithm that is a good approximation algorithm for Steiner Forest and also satisfies an additional property that we describe in Subsection 2.3 below. We implement the augmentation step as follows. After the subproblem step, every demand pair (s_i, t_i) in the subset $\mathcal{S}$ is connected by a path of (infinite-capacity) bought edges in F. Let G/F denote

Input: an MRoB instance $(G, \mathcal{D}, w, M)$.

1. (*Sampling step*) Choose a random subset $\mathcal{S} \subseteq \mathcal{D}$ of demand pairs, by including each pair $(s_i, t_i) \in \mathcal{D}$ in $\mathcal{S}$ independently with probability $\min\{w_i/M, 1\}$.

2. (*Subproblem step*) Compute a feasible solution F to the Steiner Forest instance $(G, \mathcal{S})$, and buy (infinite) capacity on the edges of F.

3. (*Augmentation step*) Greedily rent capacity to produce a feasible solution.

Figure 1: The algorithm Sample-Augment.

the graph obtained from G by contracting all of the edges of F. Independently for each demand pair $(s_i, t_i) \notin \mathcal{S}$, we compute a shortest s_i-t_i path $\widehat{P}_i$ of G/F, and rent w_i units of capacity on each edge of $\widehat{P}_i$ that are reserved for exclusive use by (s_i, t_i). Each path $\widehat{P}_i$ corresponds to an s_i-t_i path P_i of G, where each edge of P_i either has infinite capacity or has w_i units of capacity reserved for the demand pair (s_i, t_i). The augmentation step thus installs sufficient capacity for all of the demand pairs to simultaneously route their traffic on the paths $\{P_i\}_{i=1}^{k}$.

The following lemma will be used in the next subsection and also motivates the Sample-Augment algorithm.

Lemma 2.1 *For every* MRoB *instance, there is an optimal solution such that the flow of each demand pair can be routed on a single path.*

Proof: Fix an arbitrary MRoB instance $(G, \mathcal{D}, w, M)$ and an optimal solution for it. Let F denote the edges on which the optimal solution buys infinite capacity. This optimal solution must also, independently for each demand pair (s_i, t_i), reserve w_i units of capacity on s_i-t_i paths of the contracted graph G/F. The minimum-cost way to accomplish this is to rent w_i units of capacity for each demand pair (s_i, t_i) on a shortest s_i-t_i path of G/F, as in the augmentation step of the Sample-Augment algorithm. Applying this augmentation step to the set F thus results in an optimal solution in which the traffic of each demand pair can be routed on a single path. ■

The proof of Lemma 2.1 shows that the augmentation step of the algorithm Sample-Augment extends the subproblem solution into a feasible solution in an optimal way. The crux of the MRoB problem is to identify a good set of edges on which to buy infinite capacity. We will show that the random Steiner Forest instance defined by the sampling step of the Sample-Augment algorithm leads to such a good set of edges.

The rest of this section is devoted to proving that, provided the right type of Steiner Forest algorithm is used in the subproblem step, the algorithm Sample-Augment is a good approximation algorithm for MRoB. In Section 3 we design algorithms for Steiner Forest that possess the requisite properties.

2.2 Bounding the Subproblem Cost

Algorithm Sample-Augment incurs cost both in the subproblem step (for buying capacity) and in the augmentation step (for renting capacity). We first prove a key lemma that is useful for bounding both of these costs. The lemma states that, in expectation, there is a low-cost solution to the random Steiner Forest instance solved in the subproblem step of the algorithm Sample-Augment.

Lemma 2.2 *For every instance $\mathcal{I} = (G, \mathcal{D}, w, M)$ of* MRoB*,*

$$\mathbf{E}\left[OPT_{\mathcal{S}}\right] \leq \frac{OPT_{MRoB}}{M}, \tag{1}$$

where OPT_{MRoB} is the cost of an optimal solution for $\mathcal{I}$, $OPT_{\mathcal{S}}$ is the cost of an optimal solution for the Steiner Forest *instance $(G, \mathcal{S})$, and the expectation is over the random sample $\mathcal{S}$ chosen in the sampling step of the algorithm* Sample-Augment.

Proof: Fix an instance $\mathcal{I}$ of MRoB. We prove (1) by exhibiting one feasible solution for each possible Steiner Forest instance $(G, \mathcal{S})$, such that the expected cost (over $\mathcal{S}$) of this solution is at most OPT_{MRoB}/M. Since this goal is only for the analysis, and is independent of the algorithm Sample-Augment, we can freely make use of an optimal solution for $\mathcal{I}$. By Lemma 2.1, we can consider an optimal solution that routes all of the traffic of each demand pair $(s_i, t_i) \in \mathcal{D}$ on a single path P_i^*. For an edge e, let $x_e^* = \sum_{i:\, e \in P_i^*} w_i$ denote the amount flow routed on the edge e. Let E_b denote the edges e with $x_e^* \geq M$ and E_r the rest of the edges. The cost OPT_{MRoB} of the optimal solution is

$$OPT_{MRoB} = \sum_{e \in E_b} c_e M + \sum_{e \in E_r} c_e x_e^*. \tag{2}$$

To prove (1), fix a possible random sample $\mathcal{S} \subseteq \mathcal{D}$, and define a Steiner forest $F_{\mathcal{S}}$ by

$$F_{\mathcal{S}} = E_b \cup \bigcup_{(s_i,t_i) \in \mathcal{S}} P_i^*.$$

Note that $F_{\mathcal{S}}$ consists of one part (E_b) that does not depend on $\mathcal{S}$, and one part ($\cup_{(s_i,t_i)\in\mathcal{S}} P_i^*$) that does, and is certainly a feasible solution for the Steiner Forest instance $(G, \mathcal{S})$. The cost of the first part is deterministically $c(E_b) = \sum_{e \in E_b} c_e$, a factor of M less than the cost incurred by the optimal solution for $\mathcal{I}$ for buying capacity on these edges. The expected cost of the second part is a factor of M less than the cost incurred by the optimal solution for renting capacity, because we include a demand pair (s_i, t_i) in the sample $\mathcal{S}$ with probability only w_i/M. Formally, we bound the expected cost of $F_{\mathcal{S}}$ as follows:

$$\begin{aligned}
\mathbf{E}\left[c(F_{\mathcal{S}})\right] &= \mathbf{E}\left[c(E_b)\right] + \mathbf{E}\left[c\left(E_r \cap \left(\cup_{(s_i,t_i)\in\mathcal{S}} P_i^*\right)\right)\right] \\
&= c(E_b) + \sum_{e \in E_r} c_e \cdot \mathbf{Pr}\left[e \in \cup_{(s_i,t_i)\in s} P_i^*\right] \\
&\leq c(E_b) + \sum_{e \in E_r} c_e \sum_{i:\, e \in P_i^*} \mathbf{Pr}[(s_i, t_i) \in \mathcal{S}] \\
&= c(E_b) + \sum_{e \in E_r} c_e \frac{x_e^*}{M},
\end{aligned}$$

where the inequality follows from the Union Bound. Thus the expected cost of $F_{\mathcal{S}}$ is at most the cost of an optimal solution (2) divided by M. Since $\mathbf{E}[OPT_{\mathcal{S}}] \leq \mathbf{E}[c(F_{\mathcal{S}})]$, this proves the lemma. ■

Lemma 2.2 easily implies that the expected cost of the subproblem step of Sample-Augment is small provided a good approximation algorithm for Steiner Forest is used.

Lemma 2.3 *If an α-approximation algorithm for* Steiner Forest *is used in the subproblem step of* Sample-Augment, *then the expected cost incurred in this step is at most α times the cost of an optimal* MRoB *solution.*

Proof: Fix an arbitrary instance $\mathcal{I}$ of MRoB. Let $\mathcal{A}$ be the α-approximation algorithm used in the subproblem step of Sample-Augment. The cost incurred in this step is M times that of the Steiner forest F returned by $\mathcal{A}$, since Sample-Augment buys infinite capacity on the edges of F. This cost is at most $M \cdot \alpha \cdot OPT_{\mathcal{S}}$ for every possible random sample $\mathcal{S}$ of demand pairs. The expected cost is thus at most $M \cdot \alpha \cdot \mathbf{E}[OPT_{\mathcal{S}}]$, which by Lemma 2.2 is at most $\alpha \cdot OPT_{MRoB}$. ■

The next two subsections undertake the more challenging task of bounding the expected cost of the augmentation step of the Sample-Augment algorithm.

2.3 Strict Cost Shares

Our analysis of the expected cost of the augmentation step of the Sample-Augment algorithm hinges on a type of cost sharing for the Steiner Forest problem. We next define what we call *strict cost shares*. While our definition is motivated solely by our analysis of Sample-Augment, it can also be interpreted as formalizing a natural approximate fairness condition.

The next definition states that a cost-sharing method is a way of allocating cost to the demand pairs of a Steiner Forest instance $(G, \mathcal{D})$, with the total cost allocated bounded above by that of a minimum-cost Steiner forest for $(G, \mathcal{D})$.

Definition 2.4 Let χ be a function that, for every instance $\mathcal{I} = (G, \mathcal{D})$ of Steiner Forest, assigns a nonnegative real value $\chi(\mathcal{I}, (s_i, t_i))$ to every demand pair $(s_i, t_i) \in \mathcal{D}$. The function χ is a *(Steiner forest) cost-sharing method* if, for every such instance $\mathcal{I}$,

$$\sum_{(s_i,t_i)\in\mathcal{D}} \chi(\mathcal{I}, (s_i, t_i)) \leq OPT(\mathcal{I}), \tag{3}$$

where $OPT(\mathcal{I})$ is the cost of an optimal solution to $\mathcal{I}$.

Definition 2.4 permits some rather uninteresting cost-sharing methods, including 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 each demand pair a cost share 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*. Strict cost shares will allow us to charge, in a demand pair-by-demand pair fashion, a constant fraction of the expected cost of the augmentation step of SAMPLE-AUGMENT to the expected cost of an optimal solution to the Steiner Forest subproblem. We have already bounded the latter cost in Lemma 2.2.

To make this idea precise, we require further notion. Let $\ell_G(u, v)$ denote the length of a shortest path between the vertices u and v in the graph G (with respect to the edge costs of G). As in Subsection 2.1, for a graph G and a set of edges F of G, G/F denotes the graph obtained from G by contracting all of the edges of F. As in the augmentation step of the algorithm SAMPLE-AUGMENT, the minimum per-unit cost of renting capacity between s_i and t_i, given that infinite capacity has already been bought on the edges in F, is precisely $\ell_{G/F}(s_i, t_i)$. Our main definition is then the following.

Definition 2.5 Let $\mathcal{A}$ be a deterministic algorithm for the Steiner Forest problem. A Steiner forest cost-sharing method χ is *β-strict for $\mathcal{A}$* if for all instances $\mathcal{I} = (G, \mathcal{D})$ and for all demand pairs $(s_i, t_i) \in \mathcal{D}$,

$$\ell_{G/F}(s_i, t_i) \leq \beta \cdot \chi(\mathcal{I}, (s_i, t_i)),$$

where F is the Steiner forest returned for the instance $(G, \mathcal{D} \setminus \{(s_i, t_i)\})$ by the algorithm $\mathcal{A}$.

Remark 2.6 Definition 2.4 makes no reference to an algorithm for Steiner Forest, but Definition 2.5 does. Thus a Steiner forest cost-sharing method can be β-strict for one algorithm and not for another. For example, every cost-sharing method is strict with respect to the (highly suboptimal) algorithm that always returns the entire graph G as the Steiner forest solution F. Our challenge will be to give a strict cost-sharing method for a good approximation algorithm for Steiner Forest.

We say that an algorithm is strict if it admits a strict cost-sharing method.

Definition 2.7 An algorithm $\mathcal{A}$ for the Steiner Forest problem is *β-strict* if there exists a cost-sharing method that is β-strict for $\mathcal{A}$.

Strict cost shares will pay dividends in Lemma 2.9 below, where we show that they are the key property of a Steiner Forest algorithm that allows us to bound the expected augmentation cost of the algorithm SAMPLE-AUGMENT.

Example 2.8 (Prim Cost Shares) We now give an example of a strict cost-sharing method for the special case of the SSRoB problem, where all demand pairs share the same sink vertex t. In this case, the subproblem step is an instance $(G, \mathcal{S})$ of Steiner Tree, where we must output a set F of edges spanning t and all of the source vertices s_i in demand pairs of $\mathcal{S}$. Suppose we use the well-known MST heuristic as our Steiner Tree algorithm $\mathcal{A}$, implemented with Prim's MST algorithm (see e.g. [62]). In more detail, we iteratively build up a feasible solution to $(G, \mathcal{S})$ as follows. Initially, set $D = \{t\}$ and $F = \emptyset$. At each iteration, among all sources in a demand pair of $\mathcal{S}$ but not in D, find the source s_i closest to some source or sink already in D; add s_i to D; and add to F a shortest path between s_i and its nearest neighbor in D.

For an instance $\mathcal{I} = (G, \mathcal{S})$ of Steiner Tree, define the cost share $\chi(\mathcal{I}, (s_i, t))$ of (s_i, t) as half of the length of the shortest path used in the iteration of the algorithm that adds s_i to D. We call these *Prim cost shares*. We claim that the function χ satisfies both Definition 2.4 and Definition 2.5 with $\beta = 2$. Definition 2.4 is met because the sum of all of the cost shares is exactly half of the cost of the Steiner tree output by the MST heuristic, which in turn is at most twice the cost of a minimum-cost Steiner tree (see e.g. [62]).

To see why the cost shares χ are 2-strict for the algorithm $\mathcal{A}$, consider an arbitrary Steiner Tree instance $\mathcal{I} = (G, \mathcal{S})$ and demand pair $(s_i, t) \in \mathcal{S}$. Consider running the algorithm $\mathcal{A}$ in parallel on the instances $\mathcal{I}$ and $\widehat{\mathcal{I}} = (G, \mathcal{S} \setminus \{(s_i, t)\})$. The key observation is this: *these two executions of $\mathcal{A}$ are identical, until the demand pair (s_i, t) of $\mathcal{I}$ is considered.* In other words, if $\mathcal{A}$ chooses (s_i, t) in iteration $j \geq 1$ of its execution for the original instance $\mathcal{I}$, then the partial solution F_{j-1} that $\mathcal{A}$ has constructed after $j-1$ iterations is the same in both executions of the algorithm. Suppose when algorithm $\mathcal{A}$ is run on the instance $\mathcal{I}$, it connects s_i to F_{j-1} via the path P, where P is a shortest path between s_i and some previously added source or sink. Since $\mathcal{A}$'s final solution $\widehat{F}$ to the instance $\widehat{\mathcal{I}}$ includes F_{j-1}, the shortest-path distance $\ell_{G/\widehat{F}}(s_i, t)$ is at most the cost $c(P)$ of P. Since the cost share $\chi(\mathcal{I}, (s_i, t))$ is precisely $c(P)/2$, Definition 2.5 is satisfied with $\beta = 2$.

2.4 Bounding the Augmentation Cost

The definition of strict cost shares is engineered so that the following upper bound on the expected augmentation cost of the algorithm Sample-Augment holds.

Lemma 2.9 *If a β-strict algorithm for* Steiner Forest *is used in the subproblem step of* Sample-Augment*, then the expected cost incurred in the augmentation step of* Sample-Augment *is at most β times the cost of an optimal* MRoB *solution.*

Proof: Suppose the β-strict Steiner Forest algorithm $\mathcal{A}$ is used in the subproblem step of the algorithm Sample-Augment and fix an MRoB instance $(G, \mathcal{D}, w, M)$. For each demand pair $(s_i, t_i) \in \mathcal{D}$, we define two random variables. First, the random variable R_i ("renting cost") has value 0 if (s_i, t_i) is included in the random sample $\mathcal{S}$, and otherwise has value equal to the renting cost $w_i \cdot \ell_{G/F}(s_i, t_i)$ caused by (s_i, t_i) in the augmentation step, where F is the Steiner Forest solution returned by $\mathcal{A}$ for the instance $(G, \mathcal{S})$. Second, the random variable B_i ("buying cost") has value $M \cdot \chi((G, \mathcal{S}), (s_i, t_i))$ if (s_i, t_i) is included in the random sample $\mathcal{S}$ and 0 otherwise. Note that the cost incurred by Sample-Augment in the augmentation step is precisely the total renting cost $\sum_i R_i$. The total buying cost satisfies

$$\sum_{i=1}^{k} B_i = \sum_{(s_i,t_i) \in \mathcal{S}} M \cdot \chi((G, \mathcal{S}), (s_i, t_i)) \leq M \cdot OPT(G, \mathcal{S}), \tag{4}$$

where the inequality follows from Definition 2.4. Lemma 2.2 then implies that the expected total buying cost is at most the cost OPT_{MRoB} of an optimal solution to $(G, \mathcal{D}, w, M)$:

$$\mathbf{E}\left[\sum_{i=1}^{k} B_i\right] \leq OPT_{MRoB}. \tag{5}$$

The rest of the proof shows how to use strict cost shares to charge, up to a factor of β, the expected renting cost incurred by Sample-Augment to the expected buying cost.

Fix a demand pair (s_i, t_i). Condition on the set $\mathcal{S} \subseteq \mathcal{D} \setminus \{(s_i, t_i)\}$ of other demand pairs that Sample-Augment includes in its random sample. Let $\widehat{\mathcal{S}}$ denote $\mathcal{S} \cup \{(s_i, t_i)\}$. Thus the subproblem step will involve either the Steiner Forest instance $\widehat{\mathcal{I}} = (G, \widehat{\mathcal{S}})$ (with probability $\min\{w_i/M, 1\}$) or the instance $\mathcal{I} = (G, \mathcal{S})$ (with the remaining probability). The expected renting cost incurred by (s_i, t_i), conditioned on $\mathcal{S}$, can therefore be crudely bounded by

$$\mathbf{E}\left[R_i|\mathcal{S}\right] = \left(1 - \min\left\{\frac{w_i}{M}, 1\right\}\right) \cdot w_i \cdot \ell_{G/F}(s_i, t_i) \leq \min\{w_i, M\} \cdot \ell_{G/F}(s_i, t_i), \tag{6}$$

where F is the output of $\mathcal{A}$ for the Steiner Forest instance $\mathcal{I}$. The expected buying cost is

$$\mathbf{E}\left[B_i|\mathcal{S}\right] = \min\left\{\frac{w_i}{M}, 1\right\} \cdot M \cdot \chi(\widehat{\mathcal{I}}, (s_i, t_i)) = \min\{w_i, M\} \cdot \chi(\widehat{\mathcal{I}}, (s_i, t_i)). \tag{7}$$

Strict cost shares provide the key relation between renting and buying costs. Specifically, since $\mathcal{A}$ is β-strict, inequality (6) and equation (7) imply that

$$\mathbf{E}\left[R_i|\mathcal{S}\right] \leq \beta \cdot \mathbf{E}\left[B_i|\mathcal{S}\right].$$

Since this inequality holds for every set $\mathcal{S} \subseteq \mathcal{D} \setminus \{(s_i, t_i)\}$, it also holds unconditionally:

$$\mathbf{E}\left[R_i\right] \leq \beta \cdot \mathbf{E}\left[B_i\right].$$

Linearity of expectations and inequality (5) complete the proof:

$$\begin{aligned} \mathbf{E}\left[\sum_{i=1}^{k} R_i\right] &\leq \beta \cdot \mathbf{E}\left[\sum_{i=1}^{k} B_i\right] \\ &\leq \beta \cdot OPT_{MRoB}. \end{aligned} \tag{8}$$

■

Lemmas 2.3 and 2.9 immediately imply the main result of this section: Sample-Augment is a good approximation algorithm for MRoB, provided a good, strict Steiner Forest algorithm is used in the subproblem step.

Theorem 2.10 *If a β-strict α-approximation algorithm for* Steiner Forest *is used in the subproblem step of* Sample-Augment*, then* Sample-Augment *is a randomized $(\alpha + \beta)$-approximation algorithm for* MRoB*.*

3 Rent-or-Buy Problems

We next apply the analysis framework of Section 2, and Theorem 2.10 in particular, to several rent-or-buy problems. We begin in Subsection 3.1 with the special case of the SSRoB problem, and show how the results of Section 2 easily give a simple algorithm with a better performance guarantee than all previously known approximation algorithms for the problem.

We then consider the more general MRoB problem. We first show (Subsection 3.2) that the well-known primal-dual 2-approximation approximation algorithm for the Steiner Forest problem [1, 29] does not admit simple $O(1)$-strict cost shares. In Subsection 3.3 we modify this algorithm so that it remains an $O(1)$-approximation algorithm for Steiner Forest and also admits simple $O(1)$-strict cost shares, which leads to an $O(1)$-approximation algorithm for MRoB via Theorem 2.10. Finally, Subsection 3.4 extends our MRoB algorithm and analysis to the MuRoB problem.

3.1 Single-Sink Rent-or-Buy

A good approximation algorithm for the SSRoB problem follows immediately from the Prim cost shares of Example 2.8 and Theorem 2.10. Specifically, in Example 2.8 we argued that the MST heuristic is a 2-approximation algorithm for the Steiner Tree problem and admits 2-strict cost shares. Theorem 2.10 then implies the following.

Theorem 3.1 *Algorithm* Sample-Augment*, with the subproblem step implemented with the MST heuristic, is a 4-approximation algorithm for the* SSRoB *problem.*

Theorem 3.1 already improves over the previously best algorithm for the SSRoB problem, the primal-dual 4.55-approximation algorithm of Swamy and Kumar [60].

We can achieve a slightly better approximation ratio by refining Definition 2.5 and Theorem 2.10 for the SSRoB problem. For the rest of this subsection, we call a source or sink of a Steiner Tree instance a *demand*.

Definition 3.2 A Steiner tree cost-sharing method χ is *universally β-strict* if for all Steiner Tree instances $\mathcal{I} = (G, \mathcal{D})$ and for all demand pairs $(s_i, t) \in \mathcal{D}$,

$$\ell(s_i, D \setminus \{s_i\}) \leq \beta \cdot \chi(\mathcal{I}, (s_i, t)),$$

where D denotes the set of demands of $\mathcal{I}$ and $\ell(s_i, D \setminus \{s_i\})$ the length of a shortest path between s_i and some other demand.

Example 3.3 Recall that the Prim cost shares defined in Example 2.8 assign to each demand pair (s_i, t) a cost share equal to half of the length of a shortest path between s_i and some other demand. This is at least half of the length $\ell(s_i, D \setminus \{s_i\})$ of the shortest such path. Prim cost shares are therefore universally 2-strict.

The next lemma justifies the use of the word "universal" in Definition 3.2: universally strict cost shares are strict with respect to every Steiner Tree algorithm.

Lemma 3.4 *If χ is a universally β-strict Steiner tree cost-sharing method and $\mathcal{A}$ is a* Steiner Tree *algorithm, then χ is β-strict for $\mathcal{A}$.*

Proof: To satisfy Definition 2.5, we must show that $\ell_{G/F}(s_i, t) \leq \beta \cdot \chi(\mathcal{I}, (s_i, t))$ for every Steiner Tree instance $\mathcal{I} = (G, \mathcal{D})$ and every demand pair $(s_i, t) \in \mathcal{D}$, where F is the output

of $\mathcal{A}$ for the Steiner Tree instance $(G, \mathcal{D} \setminus \{(s_i, t)\})$. Letting D denote the set of demands of $\mathcal{I}$, this inequality holds as

$$\ell_{G/F}(s_i, t) \leq \ell(s_i, D \setminus \{s_i\}) \leq \beta \cdot \chi(\mathcal{I}, (s_i, t)),$$

where the first inequality follows the fact that F must include a path between t and every other demand in $D \setminus \{s_i\}$, and the second inequality follows from Definition 3.2. ■

Theorem 2.10 and Lemma 3.4 immediately give the following result.

Theorem 3.5 *Suppose there is a universally β-strict Steiner tree cost sharing method. If an α-approximation algorithm for* Steiner Tree *is used in the subproblem step of* Sample-Augment*, then* Sample-Augment *is a randomized $(\alpha + \beta)$-approximation algorithm for* SSRoB.

Theorem 3.5 decouples the tasks for finding a good Steiner Tree approximation algorithm and finding (universally) strict cost shares. Combining the universally 2-strict Prim cost shares and the 1.55-approximation algorithm for Steiner Tree due to Robins and Zelikovsky [58] then yields a 3.55-approximation algorithm for SSRoB.

Corollary 3.6 *There is a randomized 3.55-approximation algorithm for the* SSRoB *problem.*

Remark 3.7 The same graphs that show that the MST heuristic is no better than a 2-approximation algorithm for Steiner Tree (see e.g. [62, Example 3.4]) prove that for every constant $\beta < 2$, there is no universally β-strict Steiner tree cost sharing method. On the other hand, better upper bounds on the approximation ratio of Sample-Augment could follow from stricter cost shares that are not universally strict, or from improvements to the upper bound in Theorem 3.5.

Remark 3.8 In the proof of Lemma 3.4, we crucially used the fact that every feasible solution to a Steiner Tree instance is a single connected component. Since different feasible solutions to a Steiner Forest instance can have different sets of connected components, there do not seem to be useful analogues of Definition 3.2 and Theorem 3.5 for the Steiner Forest and MRoB problems, respectively.

3.2 Multicommodity Rent-or-Buy: Motivation

In the next subsection, we design a constant-factor approximation algorithm for the MRoB problem. The algorithm, and especially the analysis, will be more involved than in Subsection 3.1. This subsection motivates our algorithm. We first review the primal-dual 2-approximation algorithm for Steiner Forest due to Agrawal, Klein, and Ravi [1] and Goemans and Williamson [29], which is closely related to our Steiner Forest subroutine. We then present an instance of MRoB that suggests that the algorithm of [1, 29] should be made "more aggressive" to facilitate the definition of strict cost shares (and the application of Theorem 2.10).

3.2.1 The AKR-GW Algorithm

We now review the 2-approximation algorithm for Steiner Forest due to Agrawal, Klein, and Ravi [1] and Goemans and Williamson [29], which we refer to as the AKR-GW algorithm. Our exposition will be similar to that in [29]. Until very recently [49], this was the only known constant-factor approximation algorithm for the problem.

Fix an instance $\mathcal{I} = (G, \mathcal{D})$ of Steiner Forest. For a subset $S \subseteq V$ of vertices and a demand pair (s_i, t_i), we say that S *separates* (s_i, t_i) if S contains exactly one of s_i or t_i. The set S is a *Steiner cut* of $\mathcal{I}$ if it separates some demand pair. Let $\mathcal{C}$ denote set of Steiner cuts of $\mathcal{I}$. Finally, let $\delta(S)$ denote the set of edges with exactly one endpoint in the vertex set $S \subseteq V$. The AKR-GW algorithm iteratively constructs a feasible integral solution to the linear relaxation

$$
\begin{aligned}
& \min \quad \sum_{e \in E} c_e x_e \\
& \text{subject to:} \\
(PLP) \qquad & \sum_{e \in \delta(S)} x_e \geq 1 && \text{for every Steiner cut } S \in \mathcal{C} \\
& x_e \geq 0 && \text{for every edge } e \in E,
\end{aligned}
$$

and a feasible solution to the corresponding dual linear program

$$
\begin{aligned}
& \max \quad \sum_{S \in \mathcal{C}} y_S \\
& \text{subject to:} \\
(DLP) \qquad & \sum_{S \in \mathcal{C} \,:\, e \in \delta(S)} y_S \leq c_e && \text{for every edge } e \in E \\
& y_S \geq 0 && \text{for every Steiner cut } S \in \mathcal{C}.
\end{aligned}
$$

The 0-1 integer solutions to (PLP) are precisely the incidence vectors of the feasible solutions of $\mathcal{I}$. By weak linear programming duality (see e.g. [18]), the objective function value of every feasible (fractional) solution to the dual program (DLP) is a lower bound on the objective function value of every feasible (fractional) solution to (PLP), and in particular on the value of a minimum-cost Steiner forest for $(G, \mathcal{D})$.

The AKR-GW algorithm is shown in Figure 2. It maintains a set of edges, initially empty; a feasible dual solution, initially the all-zero solution; and a partition of the vertices, initially with all vertices in their own class of the partition. Edges in the current primal solution are called *tight*. We will call classes of the vertex partition *clusters*. The algorithm will maintain the invariant that clusters correspond to the connected components of the set of tight edges. A cluster is *active* if it is a Steiner cut and *inactive* otherwise.

In every iteration of the first part of the AKR-GW algorithm, the dual variables of the currently active clusters are increased by the largest common amount that does not violate any of the dual packing constraints of the form $\sum_{S \in \mathcal{C} \,:\, e \in \delta(S)} y_S \leq c_e$. (If these dual variables can be increased by an arbitrarily large amount, then the instance $(G, \mathcal{D})$ is infeasible.) After this dual increase, there is at least one edge whose packing constraint is satisfied with

Input: a Steiner Forest instance $(G, \mathcal{D})$.

1. Initialize all of the dual variables y_S to 0 and the clusters to the vertices $\{v\}_{v \in V}$.
2. While there is at least one active cluster (a cluster separating some demand pair):
   (a) Uniformly raise the dual variables of the active clusters as much as possible without violating dual feasibility.
   (b) Let e be an edge satisfying $\sum_{S \in \mathcal{C} : e \in \delta(S)} y_S = c_e$, where the endpoints of e are in distinct clusters, at least one of which is active. Declare e to be tight.
   (c) Merge the two clusters containing the endpoints of e into a single cluster.
3. Output the tight edges essential for feasibility.

Figure 2: Outline of the AKR-GW algorithm.

equality and with endpoints in different clusters, at least once of which is active. One such edge e is then deemed *tight*, and the two clusters containing the endpoints of e are merged into a single cluster. Note that one of these two old clusters could have been inactive, and the new cluster could be active or inactive. Eventually, all clusters are inactive and this portion of the algorithm halts.

For convenience, we associate a notion of *time* with this phase of the AKR-GW algorithm. At the beginning of the algorithm the time τ is set to 0. Every time dual variables are increased, the current time increases by the same amount as the dual variables.

Ties between different potentially tight edges at a given time can be broken arbitrarily. However, we assume throughout this paper, and particularly in Lemma 3.25 below, that the AKR-GW algorithm is implemented with a consistent tie-breaking rule (such as a lexicographic rule).

The final and most subtle step of the AKR-GW algorithm identifies a subset of the tight edges that is a feasible solution and also has low cost. Several slightly different implementations of this "delete step" have been proposed [1, 29, 30]. With an eye toward our Steiner Forest algorithm in the next subsection, we adopt that of Goemans and Williamson [29]. Precisely, let F denote the set of tight edges. An edge of F is *inessential* if $F \setminus \{e\}$ is a feasible solution for $(G, \mathcal{D})$, and *essential* otherwise. The final output of the AKR-GW algorithm is the set of essential tight edges. The algorithm can clearly be implemented in polynomial time. For fast implementations, see [19, 26, 48].

It is not immediately obvious that the algorithm AKR-GW outputs a feasible solution, let alone one with low cost. Nonetheless, Agrawal, Klein, and Ravi [1] and Goemans and Williamson [29] proved the following guarantee.

Theorem 3.9 ([1, 29]) *For every* Steiner Forest *instance* $(G, \mathcal{D})$*, the* AKR-GW *algorithm outputs a feasible dual solution* $\{y_S\}_{S \in \mathcal{C}}$ *and a feasible Steiner forest* $F \subseteq E$ *satisfying*

$$\sum_{e \in F} c_e \leq 2 \sum_{S \in \mathcal{C}} y_S. \tag{9}$$

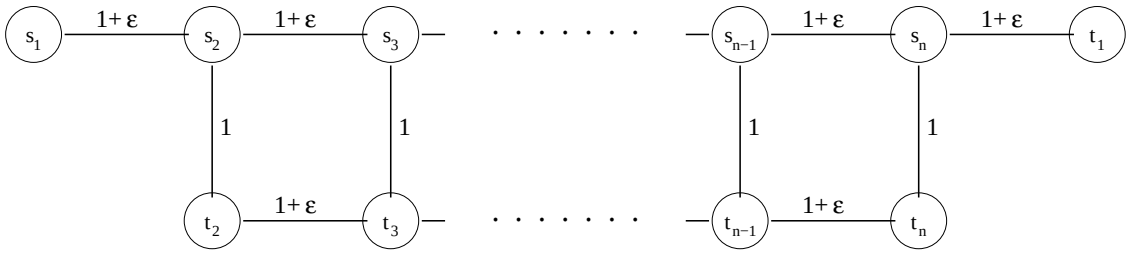

Figure 3: Example 3.10. A Steiner Forest instance showing that no straightforward cost-sharing method for the AKR-GW algorithm is $O(1)$-strict.

Since the sum on the right-hand side of (9) is a lower bound on the value of a minimum-cost Steiner forest of $(G, \mathcal{D})$, Theorem 3.9 implies that the AKR-GW algorithm is a 2-approximation algorithm for the Steiner Forest problem.

We will prove a generalization of Theorem 3.9 in Subsection 3.3 and Appendix A.

3.2.2 A Tricky Instance for the AKR-GW Algorithm

In light of Theorem 2.10, a natural idea is to use the AKR-GW algorithm as the Steiner Forest subroutine in the Sample-Augment algorithm and attempt to define $O(1)$-strict cost shares for it. Such cost shares would give a constant-factor approximation algorithm for MRoB. Moreover, the dual variables constructed by the AKR-GW algorithm suggest the following family of natural Steiner forest cost-sharing methods: when a dual variable y_S is increased by an additive factor of Δ, increase the cost shares of the demand pairs that are separated by S by at most Δ, with this increase split between these cost shares in an arbitrary way. The sum of cost shares defined in this way is at most the value of the dual feasible solution constructed by the AKR-GW algorithm, which in turn is at most the value of a minimum-cost Steiner forest. Such cost shares thus satisfy Definition 2.4. But are they strict?

Our next example shows that no cost-sharing scheme of this type is $O(1)$-strict for the AKR-GW algorithm. Precisely, call a Steiner forest cost-sharing method χ *straightforward for* AKR-GW if, for every Steiner Forest instance $\mathcal{I} = (G, \mathcal{D})$ and every demand pair $(s_i, t_i) \in \mathcal{D}$, the cost share $\chi(\mathcal{I}, (s_i, t_i))$ is at most the sum of the dual variables y_S of the AKR-GW algorithm that correspond to clusters S that separate (s_i, t_i). Note that all of the cost-sharing methods in the aforementioned family are straightforward for AKR-GW in this sense.

Example 3.10 Consider the Steiner Forest instance $\mathcal{I}$ shown in Figure 3, where n is arbitrarily large and $\epsilon < 1/n$. We will show that every cost-sharing method χ that is straightforward for AKR-GW is $\Omega(n)$-strict for AKR-GW.

Consider the execution of the AKR-GW algorithm on the instance $\mathcal{I}$ just after the time $\frac{1}{2}$. There are $n+1$ clusters: s_1 and t_1 are each in an (active) singleton cluster, and s_i and t_i share an (inactive) cluster for $i = 2, 3, \ldots, n$. By the time $\tau^* = (1 + \epsilon n)/2$, all of the vertices lie in the same (inactive) cluster. The maximum cost share that can be allocated to

the demand pair (s_1, t_1) by the straightforward cost-sharing method χ is $2\tau^* = O(1)$.

Now let $\widehat{\mathcal{I}}$ denote the instance $(G, \mathcal{D} \setminus \{(s_1, t_1)\})$ and consider the execution of the AKR-GW algorithm on $\widehat{\mathcal{I}}$. All clusters are inactive by the time $\frac{1}{2}$, and the final output of the algorithm is the set F of unit cost edges. The s_1-t_1 distance $\ell_{G/F}(s_1, t_1)$ is thus $n(1+\epsilon) = \Omega(n)$. The cost-sharing method χ is therefore only $\Omega(n)$-strict.

Example 3.10 suggests the following more delicate strategies for using Theorem 2.10 to obtain a constant-factor approximation algorithm for the MRoB problem.

(1) Modify the AKR-GW algorithm, presumably by forcing it to build a limited number of additional edges, so that there is a straightforward cost-sharing method that is $O(1)$-strict.

(2) Design a non-straightforward $O(1)$-strict cost-sharing method for the AKR-GW algorithm.

In the next subsection, we successfully pursue the first approach and obtain a $(4 + 2\sqrt{2})$-approximation algorithm for MRoB. Very recently, Fleischer et al. [25] followed the second approach and designed a non-straightforward cost-sharing method that is 3-strict for the AKR-GW algorithm, which by Theorem 2.10 gives a 5-approximation algorithm for MRoB. They also show that for every $\beta < 8/3$, there is no β-strict cost-sharing method for the AKR-GW algorithm.

3.3 Multicommodity Rent-or-Buy: Algorithm and Analysis

We now give a constant-factor approximation algorithm for the MRoB problem by designing a constant-factor approximation algorithm for Steiner Forest that admits an $O(1)$-strict cost-sharing method. We first show how to make the AKR-GW algorithm "more aggressive" in a controlled way, and then design strict cost shares for this modified algorithm. The algorithm and the cost-sharing method are both reasonably simple and are closely based on the AKR-GW algorithm; only the analysis of our algorithm is involved.

3.3.1 The γ-AKR-GW Algorithm

To modify the AKR-GW algorithm to build additional edges, we make two changes. First, we prolong the period of time during which tight edges are identified. Second, we modify the delete step of the AKR-GW algorithm so that it does not completely reverse the progress made in the earlier phase of the algorithm.

The first modification is fairly easy to implement by altering the rule used to classify clusters as active or inactive. Recall that we associate a notion of time with the AKR-GW algorithm, which tracks the amounts by which the algorithm increases dual variables. For a demand pair (s_i, t_i) of a Steiner Forest instance, we let T_i denote its *merging time* in the AKR-GW algorithm—the earliest time at which s_i and t_i are contained in a common cluster. The main idea for acquiring extra tight edges is to force s_i and t_i to remain active for γT_i time units for some $\gamma \geq 1$.

Input: a Steiner Forest instance $(G, \mathcal{D})$.

1. Run the AKR-GW algorithm and obtain the induced vector T of merging times.
2. Initialize all of the dual variables y_S to 0; the clusters to the vertices $\{v\}_{v \in V}$; and the partition $\mathcal{P}$ to the demands $\{d\}_{d \in D}$.
3. While there is at least one active cluster (a cluster that contains a demand s_i or t_i for which $\gamma \cdot T_i$ is at least the current time):
   (a) Uniformly raise the dual variables of the active clusters as much as possible without violating dual feasibility.
   (b) Let e be an edge satisfying $\sum_{S \in \mathcal{C} : e \in \delta(S)} y_S = c_e$, where the endpoints of e are in distinct clusters, at least one of which is active. Declare e to be tight.
   (c) Merge the two clusters containing the endpoints of e into a single cluster.
   (d) Merge the classes of the partition $\mathcal{P}$ that contain the active demands in these two clusters into a single class of $\mathcal{P}$.
4. Output the tight edges essential for $\mathcal{P}$-connectivity.

Figure 4: Outline of the γ-AKR-GW algorithm.

Formally, let $(G, \mathcal{D})$ be an instance of Steiner Forest, T the corresponding vector of merging times in the AKR-GW algorithm, and $\gamma \geq 1$ a parameter. Let D denote the set of demands (sources and sinks) of $(G, \mathcal{D})$. The first phase of our algorithm, which we call the γ-AKR-GW algorithm, is identical to that in AKR-GW except for the definition of active and inactive clusters. A demand s_i or t_i of D is defined to be *active* if the current time τ is less than or equal to γT_i and *inactive* otherwise. A cluster is defined to be *active* if it contains at least one active terminal and *inactive* otherwise. Note that a cluster may be active in the γ-AKR-GW algorithm even though it separates no demand pair. Tight edges are identified and clusters are merged as in the AKR-GW algorithm; this phase of the algorithm halts when no active clusters remain.

The γ-AKR-GW algorithm might raise dual variables y_S for sets $S \subseteq V$ that are not Steiner cuts and therefore do not participate in the dual linear program (DLP). Nevertheless, the algorithm is well defined. Our analysis below will bound the contribution of these artificial dual variables.

To implement its delete step, the γ-AKR-GW algorithm maintains a partition $\mathcal{P}$ of the demands D. Each class of the partition $\mathcal{P}$ should be interpreted as a collection of demands that we want to be mutually connected in the output of the algorithm. Initially, each demand lies in a separate class of this partition. When two clusters merge, the partition classes containing currently active demands of the clusters are merged into a single class of the partition $\mathcal{P}$.

Lastly, consider the final partition $\mathcal{P}$, after all of the clusters have become inactive. A set of edges is *$\mathcal{P}$-connected* if it contains a path between every pair of demands that lie in a

common class of $\mathcal{P}$. Let F denote the final set of tight edges. A tight edge e is *inessential* if $F \setminus \{e\}$ is $\mathcal{P}$-connected and *essential* otherwise. The output of the algorithm is the essential tight edges. The γ-AKR-GW algorithm is summarized in Figure 4.

Example 3.11 Suppose we run the γ-AKR-GW algorithm on the Steiner Forest instance $\widehat{\mathcal{I}}$ of Example 3.10, say with $\gamma = 2$. All of the demands $\widehat{D} = \{s_2, \ldots, s_n, t_2, \ldots, t_n\}$ remain active until the time 1. As a result, the algorithm constructs a spanning tree of tight edges that includes all of the unit-cost edges. In the final demand partition $\mathcal{P}$, all of the demands $\widehat{D}$ are in a single class. The only tight edges not essential for $\mathcal{P}$-connectivity are (s_1, s_2) and (s_n, t_1). The final output F of the algorithm is a spanning tree of the demands $\widehat{D}$ that includes all of the unit-cost edges, which is roughly twice the cost of an optimal solution of $\widehat{\mathcal{I}}$. The shortest-path distance $\ell_{G/F}(s_1, t_1)$ in the contracted graph G/F is only $2 + 2\epsilon$.

We next establish that the algorithm outputs a set of edges that is a $\mathcal{P}$-connected, feasible Steiner forest. For Lemmas 3.12–3.18 below, fix an arbitrary instance $\mathcal{I} = (G, \mathcal{D})$ of Steiner Forest and a parameter $\gamma \geq 1$. We begin with the trickiest lemma, which demonstrates a close connection between the clusters formed in the AKR-GW and γ-AKR-GW algorithms. This lemma will also play an important role in our analysis of the performance guarantee of the γ-AKR-GW algorithm. Henceforth, we use the notation $\mathcal{A}(\mathcal{I})$ to denote the execution of the algorithm $\mathcal{A}$ on the input $\mathcal{I}$.

Lemma 3.12 *At each time τ, every cluster of* AKR-GW*$(\mathcal{I})$ at time τ is a subset of a cluster of* γ-AKR-GW*$(\mathcal{I})$ at time τ.*

Proof: Call a time τ *interesting* if $\tau = 0$ or if two clusters are merged in one of the two algorithms at time τ. There are clearly only a finite number of interesting moments in time. Call the time interval between consecutive interesting moments an *epoch*. Note that during an epoch, the clusters of the two algorithms do not change. We will prove the following strengthening of the lemma for every interesting moments in time τ:

(a) every cluster of the algorithm AKR-GW at time τ is a subset of a cluster of the algorithm γ-AKR-GW at time τ;

(b) if $\{y_S^\tau\}_{S \in \mathcal{C}}$ and $\{z_S^\tau\}_{S \subseteq V}$ denote the dual solutions of the AKR-GW and γ-AKR-GW algorithms at time τ, respectively, and e is an edge spanning two clusters of the γ-AKR-GW algorithm at time τ, then

$$\sum_{S \in \mathcal{C} \,:\, e \in \delta(S)} y_S^\tau \leq \sum_{S \subseteq V \,:\, e \in \delta(S)} z_S^\tau.$$

We next prove (a) and (b) by a mutual induction.

The lemma clearly holds when $\tau = 0$. For the inductive step, consider an interesting time $\tau > 0$. Let e be an edge spanning two clusters of the γ-AKR-GW algorithm at the time τ. We claim that an endpoint v of e was contained in an active cluster S of the AKR-GW algorithm during the previous epoch only if it was contained in an active cluster of the γ-AKR-GW algorithm during this epoch. This claim follows from part (a) of the inductive

hypothesis, which implies that the cluster $\widetilde{S}$ of the γ-AKR-GW algorithm containing v in this epoch contains S, and the definition of the γ-AKR-GW algorithm, which implies that a demand is active in the AKR-GW algorithm only when it is also active in the γ-AKR-GW algorithm. This claim and part (b) of the inductive hypothesis prove part (b) of the inductive step.

For part (a) of the inductive step, we need only consider the case where at time τ the AKR-GW algorithm merges two clusters, S_1 and S_2. By the inductive hypothesis, during the epoch preceding the time τ, there were clusters $\widetilde{S}_1$ and $\widetilde{S}_2$ of the γ-AKR-GW algorithm with $S_i \subseteq \widetilde{S}_i$ for $i = 1, 2$. Since S_1 and S_2 are merged at time τ, there is an edge $e \in \delta(S_1) \cap \delta(S_2)$ that is declared tight at time τ. If e is contained in either $\widetilde{S}_1$ or $\widetilde{S}_2$, then $\widetilde{S}_1 = \widetilde{S}_2$ since distinct clusters are disjoint. In this case, $S_1 \cup S_2 \subseteq \widetilde{S}_1$ which proves part (a). Now suppose that the edge e has exactly one endpoint in each of $\widetilde{S}_1$ and $\widetilde{S}_2$. We have already shown that (b) holds at time τ, so the dual constraint for e also holds with equality in the γ-AKR-GW algorithm. Thus $\widetilde{S}_1$ and $\widetilde{S}_2$ will be merged into a common cluster (containing $S_1 \cup S_2$) by the γ-AKR-GW algorithm at time τ, completing the proof of the inductive hypothesis and the lemma. ■

Lemma 3.12 implies that at every time τ, every cluster of the γ-AKR-GW algorithm is a union of clusters of the AKR-GW algorithm. The argument in the proof of Lemma 3.12 will reoccur several times in this section.

Next we note two simple lemmas about the demand partition $\mathcal{P}$ constructed by the γ-AKR-GW algorithm. The first follows from a straightforward induction on the cluster mergings of the algorithm.

Lemma 3.13 *Suppose at some time τ in the execution γ-AKR-GW$(\mathcal{I})$, the demands $d_1, d_2 \in D$ of $\mathcal{I}$ are in a common class of the current demand partition. Then d_1 and d_2 are also in a common cluster at time τ.*

The next lemma is a partial converse of Lemma 3.13.

Lemma 3.14 *Suppose at some time τ in the execution γ-AKR-GW$(\mathcal{I})$, the demands $d_1, d_2 \in D$ are active and in a common cluster. Then d_1 and d_2 are in a common class of the demand partition at time τ.*

Proof: The demands d_1, d_2 were active when their clusters first merged, at which point the algorithm γ-AKR-GW merged the classes of $\mathcal{P}$ that contained them. ■

Lemma 3.14 inductively implies that when two clusters merge, at most two classes of the current demand partition are merged. The next lemma notes that each demand pair occupies only one class of the final demand partition constructed by the γ-AKR-GW algorithm.

Lemma 3.15 *Let $\mathcal{P}$ be the final demand partition constructed by the γ-AKR-GW algorithm and (s_i, t_i) a demand pair. The demands s_i and t_i are in the same class of $\mathcal{P}$.*

Proof: Let T_i denote the merging time of s_i and t_i in AKR-GW$(\mathcal{I})$. By Lemma 3.12, the demands s_i and t_i will reside in a common cluster of γ-AKR-GW$(\mathcal{I})$ at or before time T_i.

Since $\gamma \geq 1$, this cluster is active at time T_i. Lemma 3.14 then implies that s_i and t_i share the same class of the demand partition $\mathcal{P}$. ■

Lemma 3.15 implies that if $\mathcal{P}$ is the final demand partition of the algorithm γ-AKR-GW, then every $\mathcal{P}$-connected solution is also a feasible Steiner forest for $\mathcal{I}$. The next lemma proves that the set of tight edges constructed by the γ-AKR-GW algorithm forms a $\mathcal{P}$-connected solution.

Lemma 3.16 *Let F be the set of tight edges and $\mathcal{P}$ the demand partition at the conclusion of the γ-AKR-GW algorithm. Then F is $\mathcal{P}$-connected.*

Proof: Suppose the demands $d_1, d_2 \in D$ lie in the same class of the final partition $\mathcal{P}$. By Lemma 3.13, the demands d_1 and d_2 lie in the same cluster. Since the γ-AKR-GW algorithm maintains the invariant that the clusters correspond precisely to the connected components of the set of tight edges, there is a path of tight edges between d_1 and d_2. ■

Finally, we show that the algorithm's delete step does not destroy $\mathcal{P}$-connectivity. Our argument is essentially due to Goemans and Williamson [29]; we include the details for completeness. As a preliminary step, we note that the γ-AKR-GW algorithm never constructs a cycle of tight edges.

Lemma 3.17 *The final set F of tight edges constructed by the algorithm γ-AKR-GW is acyclic.*

Proof: Suppose for contradiction that at some point in the execution of the γ-AKR-GW algorithm, an edge $e = (v, w)$ is declared tight and creates a cycle C of tight edges. Immediately prior to e being declared tight, there was a v-w path $C \setminus \{e\}$ of tight edges. But then v and w would have been in the same cluster at this point in the algorithm, ruling out the edge e as a candidate to become tight. ■

Lemma 3.18 *The γ-AKR-GW algorithm outputs a $\mathcal{P}$-connected solution, where $\mathcal{P}$ is the final demand partition constructed by the algorithm.*

Proof: Let F be the final set of tight edges, which is $\mathcal{P}$-connected by Lemma 3.16 and acyclic by Lemma 3.17. Let $d_1, d_2 \in D$ be an arbitrary pair of demands in a common class of $\mathcal{P}$. Since F is acyclic, there is a unique d_1-d_2 path P of tight edges. Each edge of P is therefore essential and will not be deleted by the γ-AKR-GW algorithm. Thus the set of essential tight edges is $\mathcal{P}$-connected. ■

Lemmas 3.15 and 3.18 imply that the γ-AKR-GW algorithm always outputs a feasible Steiner forest.

3.3.2 Performance Guarantee of the γ-AKR-GW Algorithm

Our next goal is to show that for every $\gamma \geq 1$, the γ-AKR-GW algorithm is a $(\gamma + 1)$-approximation algorithm for the Steiner Forest problem. The main challenge, as alluded to above, is to account for the contribution of the artificial dual variables (y_S for non-Steiner

cuts S). Our first lemma bounds the cost of the Steiner forest output by the γ-AKR-GW in terms of both the legitimate and the artificial dual variables. The proof is essentially due to Goemans and Williamson [29]. For completeness, we include the proof in Appendix A.

Lemma 3.19 *For every* Steiner Forest *instance* $(G, \mathcal{D})$ *and* $\gamma \geq 1$*, the* γ-AKR-GW *algorithm outputs a dual solution* $\{z_S\}_{S \subseteq V}$ *and a feasible Steiner forest* $F \subseteq E$ *satisfying*

$$\sum_{e \in F} c_e \leq 2 \sum_{S \subseteq V} z_S. \tag{10}$$

The next lemma proves that the objective function value of the (infeasible) dual solution produced by the γ-AKR-GW algorithm is only a $(\gamma+1)/2$ factor larger than the (feasible) dual solution produced by the AKR-GW algorithm.

Lemma 3.20 *Let* $(G, \mathcal{D})$ *be an instance of* Steiner Forest*,* $\{y_S\}_{S \in \mathcal{C}}$ *the feasible dual solution produced by the* AKR-GW *algorithm, and* $\{z_S\}_{S \subseteq V}$ *the dual solution produced by the* γ-AKR-GW *algorithm. Then*

$$\sum_{S \subseteq V} z_S \leq \frac{\gamma+1}{2} \sum_{S \in \mathcal{C}} y_S.$$

Proof: We split the dual solution $\{z_S\}_{S \subseteq V}$ into two parts and bound each part separately. To define this split, let T_i denote the merging time of s_i and t_i in the AKR-GW algorithm—the earliest time that they are in the same cluster. If the dual variable z_S is increased by the γ-AKR-GW algorithm at a time τ less than T_i for some demand s_i or t_i contained in S, then this increase contributes to the part $z_S^{(1)}$; otherwise it contributes to the part $z_S^{(2)}$. Put differently, the $z_S^{(1)}$ part of the dual variable is increased until the time at which all demands in S become inactive in the AKR-GW algorithm; thereafter, the $z_S^{(2)}$ part is increased. At a given time τ, we accordingly classify an active cluster S of the γ-AKR-GW algorithm as either *good* or *bad*.

The lemma will follow immediately from the following two inequalities:

$$\sum_{S \subseteq V} z_S^{(1)} \leq \sum_{S \in \mathcal{C}} y_S \tag{11}$$

and

$$\sum_{S \subseteq V} z_S^{(2)} \leq \frac{\gamma-1}{2} \sum_{S \in \mathcal{C}} y_S. \tag{12}$$

We can prove (11) by defining, for every time τ, an injective mapping from the good active clusters of the γ-AKR-GW algorithm at time τ to the active clusters of the AKR-GW algorithm at time τ. Fix a time τ and a good active cluster $\widetilde{S}$ of the γ-AKR-GW algorithm at time τ. By the definition of good, the cluster $\widetilde{S}$ contains a demand $d \in D$ that is in an active cluster S of the AKR-GW algorithm at the time τ; map $\widetilde{S}$ to S. Lemma 3.12 implies that this mapping sends each active cluster of the γ-AKR-GW algorithm at time τ to one of its subsets. It is therefore injective, which completes the proof of (11).

To prove (12), order the demands according to increasing merging times in the AKR-GW algorithm. For convenience, we insist that the two demands of a demand pair (s_i, t_i)—which

have equal merging time—are consecutive in the ordering, with the source s_i first. We break other ties arbitrarily. For a demand $d \in D$, we will call a cluster S of the AKR-GW or γ-AKR-GW algorithm a *d-cluster* if d is the last demand in S. For a demand $d \in D$, let Y_d denote the sum of the dual variables y_S for d-clusters S of the AKR-GW algorithm. Similarly, let Z_d denote the sum of the variables $z_S^{(2)}$ for bad d-clusters S of the γ-AKR-GW algorithm. We call a demand pair (s_i, t_i) *good* if $Z_{s_i} = Z_{t_i} = 0$ and *bad* otherwise. Let $\mathcal{B} \subseteq \mathcal{D}$ denote the bad demand pairs. Note that $\sum_{(s_i,t_i)\in\mathcal{B}}(Z_{s_i} + Z_{t_i}) = \sum_{S\subseteq V} z_S^{(2)}$ and $\sum_{d\in D} Y_d = \sum_{S\in\mathcal{C}} y_S$.

We will establish the following four inequalities for every bad demand pair $(s_i, t_i) \in \mathcal{B}$:

$$\begin{aligned} Z_{s_i} &= 0; && (13)\\ Z_{t_i} &\leq (\gamma - 1)T_i; && (14)\\ Y_{s_i} &\geq T_i; && (15)\\ Y_{t_i} &\geq T_i. && (16) \end{aligned}$$

These imply that

$$\sum_{S\subseteq V} z_S = \sum_{(s_i,t_i)\in\mathcal{B}} (Z_{s_i} + Z_{t_i}) \leq (\gamma - 1) \sum_{(s_i,t_i)\in\mathcal{B}} T_i \leq \frac{\gamma - 1}{2} \sum_{d\in D} Y_d = \frac{\gamma - 1}{2} \sum_{S\in\mathcal{C}} y_S,$$

which will complete the proof of the lemma.

Let $(s_i, t_i) \in \mathcal{B}$ be a bad demand pair with $Z_d > 0$ for $d \in \{s_i, t_i\}$. Let $\widetilde{S}$ and S be the clusters of the γ-AKR-GW and AKR-GW algorithms, respectively, that contain the demand d at the merging time T_i, after all cluster mergings at this time have been performed by the algorithms. By the definition of T_i, the cluster S contains both s_i and t_i, and s_i and t_i were in separate clusters of the AKR-GW algorithm at all previous moments in time. Also, by Lemma 3.12, $\widetilde{S}$ contains both s_i and t_i at time T_i. Since t_i follows s_i in the ordering of demands, a cluster of the γ-AKR-GW algorithm can only be an s_i-cluster at time τ if $\tau < T_i$, and such clusters can only be good. This proves (13) and implies that $d = t_i$.

Next, since $Z_{t_i} > 0$, the cluster $\widetilde{S}$ must be a t_i-cluster at the time T_i. Since S is a subset of $\widetilde{S}$ containing t_i, it is also a t_i-cluster at the time T_i. Moreover, every cluster of the AKR-GW algorithm that contains a demand $d \in \{s_i, t_i\}$ at a time $\tau < T_i$ is a d-cluster. Since every such cluster is active in the AKR-GW algorithm, inequalities (15) and (16) follow.

Finally, we upper bound Z_{t_i}. By the definition of the γ-AKR-GW algorithm, a t_i-cluster can only be active at time τ if $\tau \leq \gamma \cdot T_i$. On the other hand, such a cluster can only be bad at time τ if $\tau \geq T_i$. Since only one cluster of the γ-AKR-GW algorithm contains t_i at a given moment in time, $Z_{t_i} \leq (\gamma - 1)T_i$. This proves (14) and the lemma. ■

Since the feasible dual solution constructed by the AKR-GW algorithm is a lower bound on the value of a minimum-cost Steiner forest, Lemmas 3.19 and 3.20 imply the following approximation ratio for the γ-AKR-GW algorithm.

Theorem 3.21 *For every $\gamma \geq 1$, the γ-AKR-GW algorithm is a $(\gamma + 1)$-approximation algorithm for the* Steiner Forest *problem.*

Remark 3.22 A preliminary version of this work [34] contained a weaker version of Theorem 3.21, which claimed an approximation ratio of 2γ for the γ-AKR-GW algorithm. Subsequent to [34], Becchetti et al. [12] proposed a different way to force the AKR-GW algorithm to build additional edges. They proved that, for a fixed value of a parameter $\gamma \geq 2$, their algorithm is a $(\gamma+1)$-approximation algorithm and admits $[2\gamma/(\gamma-1)]$-strict cost shares. While the arguments in [12] do not seem to carry over to the γ-AKR-GW algorithm, this result nevertheless inspired us to revisit Theorem 3.21 and prove the improved approximation ratio of $\gamma+1$ with a new proof.

3.3.3 Strict Cost Shares for the γ-AKR-GW Algorithm

Finally, we prove that the γ-AKR-GW algorithm is $O(1)$-strict provided $\gamma \geq 2$. To define our cost shares, we introduce some new terminology. A Steiner cut of a Steiner Forest instance *separates* a demand d if it does not contain the other demand of d's demand pair. A Steiner cut is *d-isolating* if it separates d and no other demand. We then use the AKR-GW algorithm to define our cost shares as follows.

Definition 3.23 (Isolated Cost Shares) Let $\mathcal{I} = (G, \mathcal{D})$ be a Steiner Forest instance. Let $\{y_S\}_{S \in \mathcal{C}}$ be the dual solution constructed by the AKR-GW algorithm for $\mathcal{I}$. For a demand $d \in D$, let $\mathcal{C}_d$ denote the d-isolating Steiner cuts of $\mathcal{I}$. The *isolated cost share* $\chi(\mathcal{I}, d)$ of a demand $d \in D$ is $\sum_{S \in \mathcal{C}_d} y_S$. The *isolated cost share* $\chi(\mathcal{I}, (s_i, t_i))$ of a demand pair $(s_i, t_i) \in \mathcal{D}$ is $\chi(\mathcal{I}, s_i) + \chi(\mathcal{I}, t_i)$.

In Definition 3.23, every Steiner cut can contribute to the cost share of at most one demand. The sum of the isolated cost shares for a Steiner Forest instance is therefore at most the value of the dual solution constructed by the AKR-GW algorithm, which in turn is at most the value of a minimum-cost Steiner forest. Isolated cost shares are thus a cost-sharing method in the sense of Definition 2.4. Isolated cost shares are also straightforward in the sense of Example 3.10. Our goal is the following theorem.

Theorem 3.24 *For every $\gamma \geq 2$, the isolated cost shares are $\frac{2\gamma}{\gamma-1}$-strict for the γ-AKR-GW algorithm.*

Our proof of Theorem 3.24 requires a number of steps. We remind the reader that Sections 4 and 5 do not depend on any of the ideas in the following proof.

First, fix $\gamma \geq 2$, a Steiner Forest instance $\mathcal{I} = (G, \mathcal{D})$, and a demand pair $(s_i, t_i) \in \mathcal{D}$. Let $\widehat{\mathcal{I}}$ denote the Steiner Forest instance $(G, \mathcal{D} \setminus \{(s_i, t_i)\})$. Let D and $\widehat{D} = D \setminus \{s_i, t_i\}$ denote the sets of demands of $\mathcal{I}$ and $\widehat{\mathcal{I}}$, respectively. We need to show that

$$\ell_{G/F}(s_i, t_i) \leq \beta \cdot \chi(\mathcal{I}, (s_i, t_i)), \tag{17}$$

where χ is the isolated cost share of (s_i, t_i) in $\mathcal{I}$, F is the Steiner forest returned by the execution γ-AKR-GW$(\widehat{\mathcal{I}})$, and $\beta = 2\gamma/(\gamma-1)$.

One main obstacle to proving Theorem 3.24 lies in relating the behavior of the AKR-GW and γ-AKR-GW algorithms on the instances $\mathcal{I}$ and $\widehat{\mathcal{I}}$, respectively. Despite the similarities

between the two instances and the two algorithms, the executions AKR-GW($\mathcal{I}$) and γ-AKR-GW($\widehat{\mathcal{I}}$) could be dramatically different. Indeed, 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 [27] and Charikar and Guha [14]. We next aim to partially avoid the detailed analyses of [14, 27] by transforming the instances $\mathcal{I}$ and $\widehat{\mathcal{I}}$. We emphasize that these transformations are only for our analysis, and in particular for the proof of Theorem 3.24.

We first modify the execution of the γ-AKR-GW algorithm on $\widehat{\mathcal{I}}$ so that it behaves more similarly to AKR-GW($\mathcal{I}$). Let T and $\widehat{T}$ denote the vectors of demand pair merging times in the executions AKR-GW($\mathcal{I}$) and AKR-GW($\widehat{\mathcal{I}}$), respectively. By definition, γ-AKR-GW($\widehat{\mathcal{I}}$) uses the vector $\gamma\widehat{T}$ to classify demands and clusters as active or inactive. The *modified execution of* γ-AKR-GW*($\widehat{\mathcal{I}}$)* instead uses the vector γT (restricted to the demand set $\widehat{D}$ of $\widehat{\mathcal{I}}$) for these classifications. The next several lemmas show that the inequality (17) is only more difficult to show for the modified execution of γ-AKR-GW($\widehat{\mathcal{I}}$) than for the original execution. We begin with a monotonicity result, similar to Lemma 3.12, which states that up to its merging time T_i, the addition of the demand pair (s_i, t_i) can only increase the rate of growth of clusters in the AKR-GW algorithm.

Lemma 3.25 *For every time $\tau \leq T_i$, every cluster of* AKR-GW*($\widehat{\mathcal{I}}$) at time τ is a subset of a cluster of* AKR-GW*($\mathcal{I}$) at time τ.*

Proof: The proof is nearly identical to that of Lemma 3.12, with AKR-GW($\mathcal{I}$) playing the role of the γ-AKR-GW algorithm in the latter proof. The only statement in the proof of Lemma 3.12 which requires a new argument here is the following: if the vertex v is in the active cluster $\widetilde{S}$ in AKR-GW($\widehat{\mathcal{I}}$) at time $\tau < T_i$ and the cluster $S \supseteq \widetilde{S}$ in AKR-GW($\mathcal{I}$) at time τ, then S is also active at this time. If S contains s_i or t_i, then S is active at time τ by the definition of the merging time T_i. Otherwise, assuming that the AKR-GW algorithm is implemented with a consistent tie-breaking rule (see Subsection 3.2), the clusters that do not contain s_i or t_i are always identical in the two executions. This fact follows from a straightforward induction on the cluster mergings of the two executions. Thus if S contains neither s_i nor t_i, then $S = \widetilde{S}$ and S is active at time τ, completing the proof. ■

Lemma 3.25 implies that demand pairs that merge before time T_i in AKR-GW($\widehat{\mathcal{I}}$) can only merge earlier in AKR-GW($\mathcal{I}$).

Corollary 3.26 *For every demand pair $(s_j, t_j) \in \mathcal{D} \setminus \{(s_i, t_i)\}$ with $\widehat{T}_j \leq T_i$, $T_j \leq \widehat{T}_j$.*

Proof: By definition, s_j and t_j are in a common cluster of AKR-GW($\widehat{\mathcal{I}}$) at time $\widehat{T}_j$. If $\widehat{T}_j \leq T_i$, then Lemma 3.25 implies that they are also in a common cluster at time $\widehat{T}_j$ in AKR-GW($\mathcal{I}$), and hence $T_j \leq \widehat{T}_j$. ■

Corollary 3.26 immediately implies that $\widehat{T}_j \geq \min\{T_i, T_j\}$ for every demand pair $(s_j, t_j) \in \mathcal{D} \setminus \{(s_i, t_i)\}$. It also leads to the next lemma, which states that clusters in the original execution of γ-AKR-GW($\widehat{\mathcal{I}}$) are only larger than in its modified execution.

Lemma 3.27 *For every time $\tau \leq \gamma T_i$, every cluster of the modified execution of γ-AKR-GW$(\widehat{\mathcal{I}})$ at time τ is a subset of a cluster of its original execution at time τ.*

Proof: As in the proof of Lemma 3.25, we only need to show that if a vertex v is in an active cluster $\widetilde{S}$ at time $\tau \leq \gamma T_i$ in the modified execution and in a cluster $S \supseteq \widetilde{S}$ in the original execution at time τ, then S is an active cluster. Since $\widetilde{S}$ is active, it contains a demand $d \in \{s_j, t_j\}$ with $\tau \leq \gamma T_j$. Since $\tau \leq \gamma \cdot \min\{T_i, T_j\}$, Corollary 3.26 implies that d and hence S are also active at time τ in the original execution of γ-AKR-GW$(\widehat{\mathcal{I}})$, which completes the proof. ■

A similar result holds for the demand partitions of the original and modified executions of γ-AKR-GW$(\widehat{\mathcal{I}})$.

Lemma 3.28 *For every time $\tau \leq \gamma T_i$, every class of the demand partition of the modified execution of γ-AKR-GW$(\widehat{\mathcal{I}})$ at time τ is a subset of a class of the demand partition of its original execution at time τ.*

Proof: We proceed by induction on the cluster mergings of the modified execution of γ-AKR-GW$(\widehat{\mathcal{I}})$. The lemma clearly holds before any cluster mergings have occurred. For the inductive step, consider a time $\tau \leq \gamma T_i$ when the modified execution merges the clusters $\widetilde{S}_1$ and $\widetilde{S}_2$. Since no partition classes are merged unless both clusters are active, we can assume that $\widetilde{S}_1$ and $\widetilde{S}_2$ contain active demands at time τ. By Lemma 3.14, the active demands of $\widetilde{S}_j$ are contained in a single demand partition class $\widetilde{C}_j$ at time τ for $j = 1, 2$. By the inductive hypothesis, there are partition classes C_1, C_2 in the original execution at time τ with $\widetilde{C}_j \subseteq C_j$ for $j = 1, 2$. After $\widetilde{S}_1$ and $\widetilde{S}_2$ are merged, $\widetilde{C}_1$ and $\widetilde{C}_2$ are merged into a single class $\widetilde{C}_1 \cup \widetilde{C}_2$. Lemma 3.27 implies that after all cluster mergings of the original execution at time τ have occurred, there is a cluster S of the original execution that contains $\widetilde{S}_1 \cup \widetilde{S}_2$. As in the proof of Lemma 3.27, since $\tau \leq \gamma T_i$, Corollary 3.26 implies that every demand that is active at time τ in the modified execution is also active in the original execution at this time. The cluster S thus contains active demands from both $\widetilde{C}_1 \subseteq C_1$ and $\widetilde{C}_2 \subseteq C_2$. By Lemma 3.14, these demands must be in the same partition class after the cluster mergings of the original execution of γ-AKR-GW$(\widehat{\mathcal{I}})$ at time τ, and this partition class must contain $C_1 \cup C_2 \supseteq \widetilde{C}_1 \cup \widetilde{C}_2$. The inductive step and the lemma are proved. ■

Let $\mathcal{P}^*$ denote the demand partition of the modified execution of γ-AKR-GW$(\widehat{\mathcal{I}})$ at the time γT_i. Call the demands of a single class of the partition $\mathcal{P}^*$ a *$\mathcal{P}^*$-group*. Recall that the demand set $\widehat{D}$ of $\widehat{\mathcal{I}}$ is $D \setminus \{s_i, t_i\}$. In particular, neither s_i nor t_i lies in any $\mathcal{P}^*$-group.

Obtain the graph H from G by, for every $\mathcal{P}^*$-group, identifying the set of vertices hosting demands from this group into a single vertex. Note that while H typically has a smaller vertex set than G, it has the same edge set and edge costs as G. The next lemma relates shortest s_i-t_i paths in H to those in G/F, where F is the Steiner forest returned by the original execution of γ-AKR-GW$(\widehat{\mathcal{I}})$.

Lemma 3.29 *Let F be the Steiner forest returned by the original execution of γ-AKR-GW$(\widehat{\mathcal{I}})$. Then*

$$\ell_{G/F}(s_i, t_i) \leq \ell_H(s_i, t_i),$$

where $\ell_H(s_i, t_i)$ denotes the value of a minimum-cost s_i-t_i path in the graph H.

Proof: Lemmas 3.16 and 3.28 imply that the output of the original execution of γ-AKR-GW($\widehat{\mathcal{I}}$) is a $\mathcal{P}^*$-connected Steiner forest F, containing a path between every two demands that lie in a common $\mathcal{P}^*$-group. All demands of a $\mathcal{P}^*$-group therefore reside in a single node of the contracted graph G/F. Every s_i-t_i path in H thus corresponds to one of no greater length in G/F, which proves the lemma. ■

Lemma 3.29 completes the first part of our proof of Theorem 3.24 and reduces the theorem to showing that

$$\ell_H(s_i, t_i) \leq \beta \cdot \chi(\mathcal{I}, (s_i, t_i)), \tag{18}$$

where $\beta = 2\gamma/(\gamma - 1)$.

We will prove (18) by, conceptually, rerunning the AKR-GW and γ-AKR-GW algorithms on the instances $\mathcal{I}_H = (H, \mathcal{D})$ and $\widehat{\mathcal{I}}_H = (H, \mathcal{D} \setminus \{(s_i, t_i)\})$, respectively. While these two new executions behave similarly to their analogues with the original graph G—as we show below—the inequality (18) is easier to establish for the instances $\mathcal{I}_H$ and $\widehat{\mathcal{I}}_H$ than for $\mathcal{I}$ and $\widehat{\mathcal{I}}$.

As before, for the analysis we need to modify the executions AKR-GW($\mathcal{I}_H$) and γ-AKR-GW($\widehat{\mathcal{I}}_H$) to use the merging times T of AKR-GW($\mathcal{I}$). Precisely, we make the following definitions, which are crucial for the following analysis.

- The *modified execution of* AKR-GW*($\mathcal{I}_H$)* deems a cluster S *active* at time τ if and only if S contains a demand s_j or t_j of D with $\tau \leq T_j$ (as opposed to if S separates some demand pair of $\mathcal{D}$).

- The *modified execution of* γ-AKR-GW*($\widehat{\mathcal{I}}_H$)* deems a cluster S *active* at time τ if and only if S contains a demand s_j or t_j of $\widehat{D}$ with $\tau \leq \gamma T_j$ (as opposed to using the merging times of demand pairs in AKR-GW($\widehat{\mathcal{I}}_H$)).

Henceforth, we abuse notation and use AKR-GW($\mathcal{I}_H$) and γ-AKR-GW($\widehat{\mathcal{I}}_H$) to denote these modified executions.

We first show that the isolated cost share $\chi(\mathcal{I}_H, (s_i, t_i))$ accrued by (s_i, t_i) in AKR-GW($\mathcal{I}_H$) is at most that in AKR-GW($\mathcal{I}$). For this result, we need an auxiliary lemma. In it, we say that a cluster $\tilde{S}$ of H *includes* a cluster S of G if every vertex of S is mapped to a vertex of $\tilde{S}$ under the vertex identification map used to obtain H from G. In particular, if $\tilde{S}$ includes S, then all demands contained in S are also contained in $\tilde{S}$.

Lemma 3.30 *For every time τ, every cluster of* AKR-GW*($\mathcal{I}$) at time τ is included in some cluster of* AKR-GW*($\mathcal{I}_H$) at time τ.*

The proof of Lemma 3.30 is almost identical to that of Lemma 3.12, and we omit further details.

We now compare the original isolated cost share $\chi(\mathcal{I}, (s_i, t_i))$ to its analogue in $\mathcal{I}_H$. Recall that demands are deemed active or inactive in AKR-GW($\mathcal{I}_H$) based on the merging times T rather than on the separated demand pairs. We accordingly say that a cluster S of AKR-GW($\mathcal{I}_H$) *isolates the demand* d at time τ if d is the sole active demand in S at the time τ.

The cost share χ^* of (s_i, t_i) in $\mathcal{I}_H$ is then defined as the total amount of time that s_i and t_i spend in isolating clusters in AKR-GW($\mathcal{I}_H$).

Lemma 3.31 *Let χ^* denote the total amount of time that s_i and t_i spend in isolating clusters in* AKR-GW*($\mathcal{I}_H$). Then*

$$\chi^* \leq \chi(\mathcal{I}, (s_i, t_i)).$$

Proof: First, observe that by the definition of (the modified execution of) AKR-GW($\mathcal{I}_H$) and the meeting time T_i in AKR-GW($\mathcal{I}$), a demand is active at time τ in one execution if and only if it is active at time τ in the other execution. Next, suppose that $d \in \{s_i, t_i\}$ is in an active, isolating cluster $\tilde{S}$ in AKR-GW($\mathcal{I}_H$) at time τ. The cluster S that contains d in AKR-GW($\mathcal{I}$) at time τ must then also be active. Moreover, Lemma 3.30 implies that S contains only fewer demands than $\tilde{S}$ at time τ, and is thus also d-isolating. Since each of s_i and t_i is only in an active, isolating cluster at time τ in AKR-GW($\mathcal{I}_H$) when it is in such a cluster at time τ in AKR-GW($\mathcal{I}$), the isolating cost share of (s_i, t_i) in the former execution is at most that in the latter. ■

Next, by Lemma 3.30, s_i and t_i are first contained in the same cluster of AKR-GW($\mathcal{I}_H$) at some time $T_i^* \leq T_i$. Since clusters correspond to connected components of tight edges, at time T_i^* there is an s_i-t_i path P of H that comprises only tight edges. Moreover, active clusters of AKR-GW($\mathcal{I}_H$) can only intersect this path in a restricted way.

Lemma 3.32 *If the cluster S is active at time τ in* AKR-GW*($\mathcal{I}_H$), then the vertices that lie in both S and P appear consecutively on P.*

Proof: Suppose for contradiction that there are vertices u, v, w on P, with v between u and w on P, such that $u, w \in S$ and $v \notin S$. Since clusters correspond to connected components of tight edges, there is a u-w path Q_1 of tight edges incident only to vertices in S at time τ in AKR-GW($\mathcal{I}_H$). On the other hand, at time T_i^* there is a u-w path Q_2 of tight edges incident to the vertex $v \notin S$—the u-w subpath of P. By the time $\max\{\tau, T_i^*\}$, all of the edges in $Q_1 \cup Q_2$ are tight in AKR-GW($\mathcal{I}_H$), at which point there is a cycle of tight edges. Since this contradicts Lemma 3.17, the proof is complete. ■

We will use the path P as a proxy for the shortest s_i-t_i path in H. This completes the second part of our proof of Theorem 3.24, and reduces the theorem to showing that the isolated cost share χ^* of (s_i, t_i) in AKR-GW($\mathcal{I}_H$) recovers a significant fraction of the cost of the path P.

For the next part of the proof, we will need to make a careful comparison of the dual variables in AKR-GW($\mathcal{I}_H$) and γ-AKR-GW($\widehat{\mathcal{I}}_H$). We call a moment τ of time *interesting* if two clusters merge in AKR-GW($\mathcal{I}_H$) at time τ, if two clusters merge in γ-AKR-GW($\widehat{\mathcal{I}}_H$) at time $\gamma\tau$, or if τ equals the merging time T_i of some demand pair (s_i, t_i) in AKR-GW($\mathcal{I}$). An *epoch* of the former execution is an interval of time between consecutive interesting moments. *Epochs* of the latter execution are the same intervals, scaled by a factor of γ. There is thus a natural bijection between the jth epochs of the two algorithms (for all j), which will play a central role in our argument. Additionally, the sets of active and inactive clusters of AKR-GW($\mathcal{I}_H$) and γ-AKR-GW($\widehat{\mathcal{I}}_H$) remain unchanged during an epoch.

Our first lemma follows immediately from the definitions of (the modified executions of) AKR-GW($\mathcal{I}_H$) and γ-AKR-GW($\widehat{\mathcal{I}}_H$).

Lemma 3.33 *For demand $d \in D \setminus \{s_i, t_i\}$ and time $\tau \geq 0$, d is active at time τ in* AKR-GW*($\mathcal{I}_H$) if and only if it is active in γ-*AKR-GW*($\widehat{\mathcal{I}}_H$) at time $\gamma\tau$.*

Next, we show that the clusters of γ-AKR-GW($\widehat{\mathcal{I}}_H$) can only have a very restricted form before time γT_i^*. We again require an auxiliary monotonicity lemma, which relates clusters of γ-AKR-GW($\widehat{\mathcal{I}}_H$) back to those of the modified execution of γ-AKR-GW($\widehat{\mathcal{I}}$) that was used to define the graph H.

Lemma 3.34 *Suppose the edge e is contained in a single cluster of γ-*AKR-GW*($\widehat{\mathcal{I}}_H$) at the time τ. Then e is also contained in a single cluster of the modified execution of γ-*AKR-GW*($\widehat{\mathcal{I}}$) at time τ.*

Proof: The inductive proof is very similar to that of Lemma 3.12, and we omit most of the details. The only additional fact required for the present proof is the following: if the lemma holds at time τ and a vertex v of H is in an active cluster at time τ in γ-AKR-GW($\widehat{\mathcal{I}}_H$), then the corresponding vertex $\tilde{v}$ of G is in an active cluster of the modified execution of γ-AKR-GW($\widehat{\mathcal{I}}$) at the time τ. We now prove this fact. Since v is in an active cluster S of H at time τ and clusters correspond to connected components of tight edges, there is path of tight edges in S from v to a vertex w that contains a demand $d_1 \in D \setminus \{s_i, t_i\}$ that is active at time τ. Since the lemma holds at time τ, there is a cluster $\tilde{S}$ of G of the modified execution of γ-AKR-GW($\widehat{\mathcal{I}}$) at this time that contains vertices $\tilde{v}$ and $\tilde{w}$ that correspond to v and w, respectively. By the definition of the graph H, there is at least one demand $d_2 \in D \setminus \{s_i, t_i\}$ at the vertex $\tilde{w}$ that is in d_1's $\mathcal{P}^*$-group. Let $\mathcal{P}$ denote the demand partition of the modified execution of γ-AKR-GW($\widehat{\mathcal{I}}$) at the time τ. (Recall that $\mathcal{P}^*$ is defined as the demand partition of this execution at the time γT_i.) By Lemma 3.13, every demand in d_2's $\mathcal{P}$-group A is contained in $\tilde{S}$ at time τ. We can finish the proof the lemma by showing that some demand in the set A is active at time τ in the modified execution of γ-AKR-GW($\widehat{\mathcal{I}}$).

Suppose for contradiction that all demands of A are inactive at the time τ. First, since d_1 is active in γ-AKR-GW($\widehat{\mathcal{I}}_H$) at time τ, it is also active in the modified execution of γ-AKR-GW($\widehat{\mathcal{I}}$) at time τ and thus $d_1 \notin A$. Second, by the definition of the γ-AKR-GW algorithm, the $\mathcal{P}$-group A will never merge with any other $\mathcal{P}$-group after the time τ. These two consequences contradict the fact that d_1 and d_2 lie in the same $\mathcal{P}^*$-group of the modified execution of γ-AKR-GW($\widehat{\mathcal{I}}$) at the time γT_i, completing the proof of the lemma. ■

Now we prove that the clusters of γ-AKR-GW($\widehat{\mathcal{I}}_H$) at a time $\tau < \gamma T_i^*$ are simple.

Lemma 3.35 *Suppose S is an active cluster of γ-*AKR-GW*($\widehat{\mathcal{I}}_H$) at the time $\tau < \gamma T_i^*$. Then S contains active demands from only one $\mathcal{P}^*$-group.*

Proof: Suppose for contradiction that S contains active demands d_1, d_2 from different $\mathcal{P}^*$-groups. Since S corresponds to a connected component of tight edges, S contains an d_1-d_2 path. By Lemma 3.34, this path (and hence d_1 and d_2) is contained in a single cluster of the modified execution of γ-AKR-GW($\widehat{\mathcal{I}}$) at time τ. By the definition of the modified executions

of γ-AKR-GW($\widehat{\mathcal{I}}$) and γ-AKR-GW($\widehat{\mathcal{I}}_H$), d_1 and d_2 are also active in the modified execution of γ-AKR-GW($\widehat{\mathcal{I}}$) at this time. But then Lemma 3.14 implies that d_1 and d_2 are in the same $\mathcal{P}^*$-group, a contradiction. ■

We next show that the active clusters of AKR-GW($\mathcal{I}_H$) are almost as simple before time T_i^*. The following auxiliary lemma is where we use the standing assumption that $\gamma \geq 2$. It roughly states that the missing cluster growth due to the absence of the demands s_i and t_i from the instance $\widehat{\mathcal{I}}_H$ can be made up for by growing the other demands for twice as long.

Lemma 3.36 *Suppose the demands $d_1 \in \{s_i, t_i\}$ and $d_2 \in D \setminus \{s_i, t_i\}$ are active and in the same cluster of* AKR-GW*($\mathcal{I}_H$) at the time $\tau < T_i^*$. Then d_1 and d_2 are in the same cluster of γ-*AKR-GW*($\widehat{\mathcal{I}}_H$) at the time 2τ.*

Proof: By symmetry, we can assume that $d_1 = s_i$. Since clusters correspond to connected components of tight edges, there is an s_i-d_2 path Q of tight edges at time τ in AKR-GW($\mathcal{I}_H$). Let $\{y_S\}_{S \subseteq V}$ denote the dual variables at this time; thus $\sum_{S \subseteq V\,:\, e \in \delta(S)} y_S = c_e$ for all $e \in Q$. We claim that the edges of Q are nearly tight at time τ in γ-AKR-GW($\widehat{\mathcal{I}}_H$) in the following sense:

$$\sum_{e \in Q} \sum_{S \subseteq V\,:\, e \in \delta(S)} z_S \geq c(Q) - \tau, \tag{19}$$

where $c(Q)$ is the cost $\sum_{e \in Q} c_e$ of Q and $\{z_S\}_{S \subseteq V}$ are the dual variables in γ-AKR-GW($\widehat{\mathcal{I}}_H$) at time τ.

We first claim that if S is subset of the vertex set of H with $t_i \in S$ and $P \cap \delta(S) \neq \emptyset$, then $y_S = 0$. Indeed, if $y_S > 0$ for such a cluster S, then S is active at or before time τ, which implies that by time τ there is a set of tight edges from t_i to Q. But then s_i and t_i are connected by a path of tight edges at time τ in AKR-GW($\mathcal{I}_H$) and are hence in the same cluster, which contradicts the assumption that $\tau < T_i^*$.

Next consider the clusters that contain neither s_i nor t_i. As in the proof of Lemma 3.25, a straightforward induction shows that the sets of such clusters are identical in AKR-GW($\mathcal{I}_H$) and γ-AKR-GW($\widehat{\mathcal{I}}_H$) at all times. Thus $z_S = y_S$ for all such clusters S. Lastly, the sum of the dual variables y_S of clusters S that contain s_i is exactly τ, and Lemma 3.32 implies that each of such cluster with $y_S > 0$ only contributes to the packing constraint of a single edge of P. Inequality (19) follows.

Finally, in γ-AKR-GW($\widehat{\mathcal{I}}_H$), the cluster containing d_2 will intersect the path Q until s_i and d_2 are in the same cluster or until d_2 becomes inactive. Since $\gamma \geq 2$ and d_2 is active at time τ in AKR-GW($\mathcal{I}_H$), d_2 is active at time 2τ in γ-AKR-GW($\widehat{\mathcal{I}}_H$). Inequality (19) implies that d_2's (active) cluster intersects P for at most τ time units beyond the time τ. Thus s_i and d_2 must be in the same cluster by time 2τ in γ-AKR-GW($\widehat{\mathcal{I}}_H$). ■

Now we use Lemma 3.36 to limit the complexity of active clusters of AKR-GW($\mathcal{I}_H$).

Lemma 3.37 *Suppose S is an active cluster of* AKR-GW*($\mathcal{I}_H$) at the time $\tau < T_i^*$. Then S contains active demands from only one $\mathcal{P}^*$-group, and does not contain both s_i and t_i.*

Proof: The cluster S does not contain both s_i and t_i by the definition of the merging time T_i^*. Suppose for contradiction that S contains active demands from distinct $\mathcal{P}^*$-groups. If S contains neither s_i nor t_i then, as in the previous proof, S is also a cluster of γ-AKR-GW($\widehat{\mathcal{I}}_H$) at the time τ. This contradicts Lemma 3.35. Finally, suppose that S contains demands d_1, d_2 from distinct $\mathcal{P}^*$-groups in addition to either s_i or t_i (s_i, say). Lemma 3.36 implies that s_i, d_1, and d_2 are in the same cluster of γ-AKR-GW($\widehat{\mathcal{I}}_H$) at the time 2τ. Since $\gamma \geq 2$, d_1 and d_2 are active at this time in γ-AKR-GW($\widehat{\mathcal{I}}_H$), which contradicts Lemma 3.35. ■

Lemma 3.37 allows us to classify the active clusters of AKR-GW($\mathcal{I}_H$) at a time $\tau < T_i^*$ into three categories. Recall that a cluster S of AKR-GW($\mathcal{I}_H$) *isolates the demand* $d \in D$ *at time* τ if d is the sole active demand in S at the time τ.

- An active cluster S of AKR-GW($\mathcal{I}_H$) at a time $\tau < T_i^*$ is *isolating* if it is s_i- or t_i-isolating.
- Such a cluster S is *shared* if it contains an active demand from $D \setminus \{s_i, t_i\}$ and either s_i or t_i.
- Such a cluster S is *independent* if it contains neither s_i nor t_i.

Suppose that S is in an active cluster of AKR-GW($\mathcal{I}_H$) during an epoch preceding the time T_i^*. If S is shared or independent, then S contains an active demand $d \in D \setminus \{s_i, t_i\}$, and so by Lemma 3.33 there is an active cluster that contains d in the corresponding epoch of γ-AKR-GW($\widehat{\mathcal{I}}_H$). If S is isolating, then there is no such corresponding cluster, as s_i and t_i are not demands in $\widehat{\mathcal{I}}_H$.

We can now describe our high-level plan for the final part of our proof of Theorem 3.24. Recall that P denotes the s_i-t_i path of tight edges at time T_i^* in AKR-GW($\mathcal{I}_H$). For a cluster S, we will say that S *crosses* P k *times* if $|P \cap \delta(S)| = k$. If the cluster S crosses P a total of k times, then it contributes a total of ky_S to the left-hand sides of the packing constraints $\sum_{S \subseteq V\,:\, e \in \delta(S)} y_S \leq c_e$ for the edges e of P. Since all edges of P are eventually tight, the sum of all such contributions is precisely $c(P)$.

Our key claim will be that for a "typical" shared or independent cluster that is active in a given epoch of AKR-GW($\mathcal{I}_H$), there is a corresponding cluster in the same epoch of γ-AKR-GW($\widehat{\mathcal{I}}_H$) that crosses P the same number of times. Since epochs in the latter execution are γ times as long as those in the former one, the contribution of these clusters to the packing constraints of the edges of P is γ times as large as in the former execution. Since the sum of all such contributions is at most $c(P)$, only a limited number of the active clusters that cross P in AKR-GW($\mathcal{I}_H$) can be shared or independent—the rest must be isolated and thus contribute to the isolated cost share of (s_i, t_i).

Now we supply the details. We first make precise the correspondence between active clusters in the two executions. We define an injective map Λ_j for each epoch j that precedes the time T_i^*. Fix such an epoch j. Let S be a shared or independent active cluster of this epoch in AKR-GW($\mathcal{I}_H$). Since S is not isolated, we can choose (arbitrarily) an active demand $d \in D \setminus \{s_i, t_i\}$ that lies in S. Let $\widetilde{S}$ be the cluster containing d in the jth epoch of

γ-AKR-GW($\widehat{\mathcal{I}}_H$). Set $\Lambda_j(S)$ to $\widetilde{S}$. Note that Λ_j is defined only on the active shared and independent clusters of the jth epoch. Next we prove several basic facts about these maps.

Lemma 3.38 *Fix an epoch j of* AKR-GW*($\mathcal{I}_H$) that concludes at or before the time T_i^*. The map Λ_j satisfies the following properties.*

(a) Λ_j is injective.

(b) Λ_j maps active clusters to active clusters.

(c) If S is an active independent cluster in the jth epoch of AKR-GW*($\mathcal{I}_H$), then $S \subseteq \Lambda_j(S)$.*

(d) If S is an active shared cluster containing $d \in \{s_i, t_i\}$ in the jth epoch of AKR-GW*($\mathcal{I}_H$), then $\Lambda_j(S)$ also contains d.*

Proof: Part (a) follows from the facts that each active cluster in the jth epoch of γ-AKR-GW($\widehat{\mathcal{I}}_H$) only contains demands from one $\mathcal{P}^*$-group (Lemma 3.35), and that each $\mathcal{P}^*$-group of demands is contained in a unique cluster of the jth epoch of AKR-GW($\mathcal{I}_H$). Part (b) follows immediately from Lemma 3.33. For part (c), recall from the proof of Lemma 3.36 that since S is an independent cluster at time τ in AKR-GW($\mathcal{I}_H$), it is also a cluster at time τ in γ-AKR-GW($\widehat{\mathcal{I}}_H$). Since clusters only grow with time, in the jth epoch of γ-AKR-GW($\widehat{\mathcal{I}}_H$) there is a cluster $\widetilde{S}$ that contains S, and Λ_j will map S to $\widetilde{S}$. Finally, part (d) follows directly from Lemma 3.36 and our standing assumption that $\gamma \geq 2$. ■

Recall that the map Λ_j is intended to set up a correspondence between clusters in the jth epoch of AKR-GW($\mathcal{I}_H$) that cross P and clusters in the jth epoch of γ-AKR-GW($\widehat{\mathcal{I}}_H$) that cross P. We next seek to prove that each map Λ_j approximately preserves the number of crossings of P. We first note the following corollary of Lemma 3.32, which limits the number of times that active clusters of AKR-GW($\mathcal{I}_H$) can cross the s_i-t_i path P.

Corollary 3.39 *Let S be an active cluster of* AKR-GW*($\mathcal{I}_H$).*

(a) If S is isolating or shared, then S crosses P at most once.

(b) If S is independent, then S crosses P at most twice.

The second consequence of Lemma 3.32 is that the image $\Lambda_j(S)$ of a shared or independent active cluster S crosses P as many times as S does, unless either s_i or t_i lies outside S and inside $\Lambda_j(S)$.

Lemma 3.40 *Let S be an active shared or independent cluster of* AKR-GW*($\mathcal{I}_H$) in an epoch j that ends at or before time T_i^*.*

(a) If $\Lambda_j(S)$ crosses P fewer times than S, then $\Lambda_j(S) \setminus S$ contains either s_i or t_i.

(b) If $\Lambda_j(S)$ crosses P two fewer times than S, then $\Lambda_j(S) \setminus S$ contains both s_i and t_i.

Proof: For part (b), Corollary 3.39 implies that we can assume that S is independent and crosses P twice, while $\Lambda_j(S)$ does not cross P. By Lemma 3.38(c), $\Lambda_j(S) \supseteq S$. Since $\Lambda_j(S)$ does not cross S, it must contain P, and in particular both s_i and t_i. A similar argument also proves (a) for independent clusters.

Finally, suppose S is shared. By symmetry, we can assume that S contains s_i. By Corollary 3.39(a), we can assume that S crosses P once while $\Lambda_j(S)$ does not cross P. By Lemma 3.38(d), $\Lambda_j(S)$ also contains s_i. Thus if $\Lambda_j(S)$ does not cross P, it must contain P and in particular t_i. The proof is complete. ■

Next we bound the number of times that s_i or t_i can appear in a cluster $\Lambda_j(S)$ but not in the preimage S.

Lemma 3.41 *Let S be an active cluster of* AKR-GW*$(\mathcal{I}_H)$ in an epoch j that ends at or before time T_i^*. Suppose the demand $d \in \{s_i, t_i\}$ lies in $\Lambda_j(S)$ but not S. Then d is isolated in the jth epoch of* AKR-GW*$(\mathcal{I}_H)$.*

Proof: We proceed by contradiction. Suppose that d is in a shared cluster S' in the jth epoch of AKR-GW$(\mathcal{I}_H)$, with S' containing a demand $d' \in D \setminus \{s_i, t_i\}$. Since this epoch precedes T_i^*, d and hence S' are active during this epoch. By Lemma 3.38(d), $\Lambda_j(S')$ contains both d and d'. By Lemma 3.38(a), Λ_j is injective and hence $\Lambda_j(S) \neq \Lambda_j(S')$. But both $\Lambda_j(S)$ and $\Lambda_j(S')$ contain d, which contradicts the fact that distinct clusters in a common epoch are disjoint. ■

With all of the preliminary results in place, we are finally prepared to finish the proof of Theorem 3.24. The proof will closely follow the outline described following Lemma 3.37.

Proof of Theorem 3.24: We adopt all of the notation used above. Lemmas 3.29 and 3.31 reduce inequality (17), and hence the proof of the theorem, to showing that

$$c(P) \leq \frac{2\gamma}{\gamma - 1} \cdot \chi^*, \tag{20}$$

where $c(P)$ is the cost of the s_i-t_i path P in H, and χ^* is the total amount of time that s_i and t_i spend in isolating clusters in AKR-GW$(\mathcal{I}_H)$.

Let $\mathcal{C}$ denote the set of possible clusters of $\mathcal{I}_H$—the sets of vertices that contain at least one demand of D. Similarly let $\widehat{\mathcal{C}}$ denote the possible clusters of $\widehat{\mathcal{I}}_H$. For a cluster S, let $y_S^{(j)}$ and $z_S^{(j)}$ denote the increment in the dual variables y_S and z_S in the jth epochs of AKR-GW$(\mathcal{I}_H)$ and γ-AKR-GW$(\widehat{\mathcal{I}}_H)$, respectively. Note that such an increment is positive in an epoch if and only if the corresponding cluster is active during the epoch.

For a cluster S, let $\kappa(S)$ denote the number of times that S crosses P. Let epoch p of AKR-GW$(\widehat{\mathcal{I}}_H)$ end at time T_i^*. Since P comprises only tight edges at time T_i^* in AKR-GW$(\mathcal{I}_H)$,

$$\sum_{j=1}^{p} \sum_{S \in \mathcal{C}} y_S^{(j)} \cdot \kappa(S) = c(P). \tag{21}$$

Also,

$$\sum_{j=1}^{p} \sum_{S \in \widehat{\mathcal{C}}} z_S^{(j)} \cdot \kappa(S) \leq c(P). \tag{22}$$

Let $\mathcal{C}_j^I \subseteq \mathcal{C}$ denote the clusters that are s_i- or t_i-isolating during the jth epoch of AKR-GW($\mathcal{I}_H$). For a non-isolating cluster $S \notin \mathcal{C}_j^I$ that is active in an epoch $j \leq p$ of AKR-GW($\mathcal{I}_H$), let $\mu_j(S) = \max\{0, \kappa(S) - \kappa(\Lambda_j(S))\}$ denote the number of "missing crossings of P" from $\Lambda_j(S)$, relative to S. Since epochs in γ-AKR-GW($\widehat{\mathcal{I}}_H$) are γ times as long as in AKR-GW($\mathcal{I}_H$), inequality (22) and Lemma 3.38(a) and (b) imply that

$$\sum_{j=1}^{p} \sum_{S \notin \mathcal{C}_j^I} \gamma \cdot y_S^{(j)} \cdot [\kappa(S) - \mu_j(S)] \leq c(P). \tag{23}$$

Next, we can use Lemma 3.41 to associate each active, non-isolating cluster $S \notin \mathcal{C}_j^I$ of the jth epoch of AKR-GW($\mathcal{I}_H$) with $\mu_j(S)$ isolating clusters from the same epoch. Moreover, the injectivity of Λ_j (Lemma 3.38(a)) implies that no such isolating cluster is mapped to more than once: for $d \in \{s_i, t_i\}$, an isolating d-cluster in the jth epoch of AKR-GW($\mathcal{I}_H$) can only be mapped to by a cluster S for which $d \in \Lambda_j(S)$. We thus have

$$\sum_{S \notin \mathcal{C}_j^I} y_S^{(j)} \cdot \mu_j(S) \leq \sum_{S \in \mathcal{C}_j^I} y_S^{(j)}$$

for each epoch $j \leq p$. Summing over all such epochs and using the definition of the isolated cost share χ^*, we have

$$\sum_{j=1}^{p} \sum_{S \notin \mathcal{C}_j^I} y_S^{(j)} \cdot \mu_j(S) \leq \chi^*. \tag{24}$$

Combining (23) and (24) then gives

$$\sum_{j=1}^{p} \sum_{S \notin \mathcal{C}_j^I} y_S^{(j)} \cdot \kappa(S) \leq \frac{c(P)}{\gamma} + \chi^*. \tag{25}$$

Subtracting inequality (25) from equation (21) gives

$$\sum_{j=1}^{p} \sum_{S \in \mathcal{C}_j^I} y_S^{(j)} \cdot \kappa(S) \geq c(P) - \frac{c(P)}{\gamma} - \chi^*. \tag{26}$$

Since $\kappa(S) \leq 1$ for all isolating clusters $S \in \mathcal{C}_j^I$ in all epochs $j \leq p$ (Corollary 3.39(a)), the left-hand side of (26) is a lower bound on χ^*. Using this fact and rearranging we obtain

$$\chi^* \geq \frac{\gamma - 1}{2\gamma} \cdot c(P),$$

which completes the proof. ■

Theorems 2.10, 3.21, and 3.24 imply that for every $\gamma \geq 2$, the algorithm Sample-Augment, using the algorithm γ-AKR-GW as its Steiner Forest subroutine, is a constant-factor approximation algorithm for MRoB.

Theorem 3.42 *Algorithm* Sample-Augment*, with the subproblem step implemented with the* γ-AKR-GW *algorithm with* $\gamma \geq 2$*, is a* $[\gamma + 1 + 2\gamma/(\gamma - 1)]$*-approximation algorithm for the* MRoB *problem.*

Choosing $\gamma = 1 + \sqrt{2}$ we obtain an approximation ratio of $4 + 2\sqrt{2} \approx 6.83$.

Corollary 3.43 *Algorithm* Sample-Augment*, with the subproblem step implemented with the* $(1 + \sqrt{2})$-AKR-GW *algorithm, is a* $(4 + 2\sqrt{2})$*-approximation algorithm for the* MRoB *problem.*

Prior to our work, the best approximation ratio known for the MRoB problem was more than one thousand [50]. In addition, the algorithm in [50] is fairly complicated. We emphasize that while our proof of Theorem 3.42 is involved, our MRoB algorithm is relatively simple, with complexity comparable to that of the AKR-GW algorithm.

Remark 3.44 In a preliminary version of this work [34], we presented a 12-approximation algorithm for MRoB. The improvement in Theorem 3.42 above comes from two sources. First, the preliminary version [34] contained a weaker version of Lemma 3.41, which led to a looser analysis in the proof of Theorem 3.42. Second, as discussed in Remark 3.22, the preliminary version [34] also contained a weaker version of Theorem 3.21. We discovered the first improvement soon after the publication of [34]; this optimization alone gives an 8-approximation algorithm for MRoB. As noted in Remark 3.22, we discovered the second refinement of our analysis only after an analogous improvement was presented by Becchetti et al. [12] for a different algorithm. Our approximation ratio of $4 + 2\sqrt{2}$ in Theorem 3.42 matches that of the algorithm in [12].

3.4 Multicast Rent-or-Buy

In this subsection we extend our algorithm and analysis for the MRoB problem to the more general MuRoB problem, where there are arbitrary demand groups in place of demand pairs. Formally, an instance of MuRoB is given by the usual graph $G = (V, E)$ with edge costs c, a parameter M, and a set $\mathcal{D} = \{D_1, \ldots, D_k\}$ of *demand groups*. Each demand group D_i is an arbitrary set of two or more demands and has a corresponding weight w_i. A feasible solution to a MuRoB instance buys and rents capacity on edges as usual, and also specifies a tree A_i for each demand group D_i that spans all of the demands of D_i. The capacity on each edge e must be at least the weight $\sum_{i:\, e \in A_i} w_i$ of the trees that include it. In other words, the capacity installed must be sufficient for simultaneous "multicast" communication within each demand group.

3.4.1 Extending the Sample-Augment and γ-AKR-GW Algorithms

Most of the algorithmic and analytic techniques of Subsection 3.3 carry over to the MuRoB problem, but a few additional ideas are needed. The high-level approach of the Sample-Augment algorithm also applies to the MuRoB problem: sample each demand group D_i independently with probability $\min\{w_i/M, 1\}$, buy infinite capacity on edges to connect demand groups in the randomly sampled subproblem, and greedily rent capacity for the

remaining demand groups. There are clearly no subtleties in implementing the sampling step. The problem that arises in the subproblem step, which we will call the Generalized Steiner Tree (GST) problem [1, 29], seems more general than the Steiner Forest problem, since the connectivity requirements now involve demand groups rather than demand pairs. An instance of GST can be converted into an equivalent instance of Steiner Forest, however, for example by replacing each demand group D_i with a set of demand pairs, one for each unordered pair of demands of D_i. Thus every α-approximation algorithm for Steiner Forest can be converted into an α-approximation algorithm for GST. Alternatively, the AKR-GW and γ-AKR-GW algorithms are easily modified to directly approximate the GST problem. First, modify the AKR-GW algorithm so that it deems a cluster S active whenever there is a demand group D_i for which S contains a non-empty and strict subset of the demands of D_i. The merging time T_i of a demand group D_i is then the earliest time at which all demands of D_i lie in a common cluster. The γ-AKR-GW algorithm is then defined for the GST problem in the obvious way.

In either case, the following analogue of Theorem 3.21 holds for the (suitably modified) γ-AKR-GW algorithm.

Theorem 3.45 *For every $\gamma \geq 1$, the γ-AKR-GW algorithm is a $(\gamma + 1)$-approximation algorithm for the GST problem.*

3.4.2 Strict Cost Shares for the γ-AKR-GW Algorithm: The Multicast Case

We next discuss strict cost-sharing methods for GST algorithms and for the γ-AKR-GW algorithm in particular. Extending the definition of a strict cost-sharing method is straightforward. By a *GST cost-sharing method* we mean a function χ that assigns a non-negative cost share $\chi(\mathcal{I}, D_i)$ to each demand group D_i of an instance $\mathcal{I}$ of GST, such that the sum of the cost shares is at most the cost of an optimal solution to $\mathcal{I}$.

Definition 3.46 Let $\mathcal{A}$ be a deterministic algorithm for the GST problem. A GST cost-sharing method χ is *β-strict for $\mathcal{A}$* if for all instances $\mathcal{I} = (G, \mathcal{D})$ of GST and for all demand groups $D_i \in \mathcal{D}$,

$$\ell_{G/F}(D_i) \leq \beta \cdot \chi(\mathcal{I}, D_i),$$

where F is the solution returned for the instance $(G, \mathcal{D} \setminus \{D_i\})$ by the algorithm $\mathcal{A}$, and $\ell_{G/F}(D_i)$ denotes the value of a minimum-cost tree in G/F that spans all of the demands of D_i.

An algorithm for the GST problem is then *β-strict* if it admits some β-strict cost-sharing method. One new complication is that the value $\ell_{G/F}(D_i)$, which represents the cheapest way of renting capacity between the demands of D_i given that infinite capacity has already been bought on the edges of F, is NP-hard to compute for general demands groups. We discuss this issue further at the end of the section.

We noted above that an α-approximation algorithm for Steiner Forest naturally induces an α-approximation algorithm for GST. Unfortunately, a strictness guarantee (in the sense of Definition 2.5) for a Steiner Forest approximation algorithm does not necessarily carry over to a strictness guarantee (in the sense of Definition 3.46) for the corresponding GST

approximation algorithm. In particular, we must reprove a strictness guarantee for the γ-AKR-GW algorithm for the GST problem.

We next outline how to modify the proof of Theorem 3.24 to show the following result.

Theorem 3.47 *For every $\gamma > 2$, the γ-AKR-GW algorithm for the GST problem is $\frac{4\gamma}{\gamma-2}$-strict.*

As in the proof of Theorem 3.24, we will show that the *isolated cost-sharing method* (Definition 3.23) is $4\gamma/(\gamma-2)$-strict for the γ-AKR-GW algorithm. In the context of the GST problem, a cluster S is called *D_i-separating* for a demand group D_i if S contains a non-empty strict subset of the demands of D_i, and is *D_i-isolating* if it separates D_i and no other demand group. The isolated cost share $\chi(\mathcal{I}, D_i)$ of a demand group D_i of a GST instance $\mathcal{I}$ is then defined as the sum of the dual variables constructed by the AKR-GW algorithm for $\mathcal{I}$ that correspond to D_i-isolating clusters.

Fix an instance $\mathcal{I} = (G, \mathcal{D})$ of GST and a demand pair $D_i \in \mathcal{D}$. Let $\widehat{\mathcal{I}}$ denote the GST instance $(G, \mathcal{D} \setminus \{D_i\})$. The first part of the proof of Theorem 3.47 is identical to that of Theorem 3.24. In particular, we define T_i to be the merging time of the demand group D_i in AKR-GW($\mathcal{I}$), the *modified execution* of γ-AKR-GW($\widehat{\mathcal{I}}$) as the execution that uses the merging times T of AKR-GW($\mathcal{I}$) (rather than of AKR-GW($\widehat{\mathcal{I}}$)) to classify clusters as active or inactive, $\mathcal{P}^*$ to be the demand partition of this modified execution at the time γT_i, and H as the graph obtained from G by identifying vertices that host demands from a common $\mathcal{P}^*$-group. Following the proofs of Lemmas 3.25–3.29 establishes the following.

Lemma 3.48 *Let F be the solution returned by (the original execution of) γ-AKR-GW($\widehat{\mathcal{I}}$). Then*

$$\ell_{G/F}(D_i) \leq \ell_H(D_i),$$

where $\ell_{G/F}(D_i)$ and $\ell_H(D_i)$ denote the values of minimum-cost trees spanning all demands of D_i in G/F and H, respectively.

The second part of the proof of Theorem 3.47 is also similar to that of Theorem 3.24. Define the GST instances $\mathcal{I}_H = (H, \mathcal{D})$ and $\widehat{\mathcal{I}}_H = (H, \mathcal{D} \setminus \{D_i\})$, and the (modified) executions AKR-GW($\mathcal{I}_H$) and γ-AKR-GW($\widehat{\mathcal{I}}_H$), which use the merging times T in AKR-GW($\mathcal{I}$) to classify clusters as active or inactive.

We next define the isolated cost share of D_i in (the modified execution of) AKR-GW($\mathcal{I}_H$). In AKR-GW($\mathcal{I}_H$), a demand of D_i is active at the time τ if and only if $\tau \leq T_i$. We call a cluster S of AKR-GW($\mathcal{I}_H$) *D_i-isolating* at the time $\tau \leq T_i$ if S contains at least one (active) demand of D_i and no active demand of another demand group. We define the *isolated cost share* χ^* of D_i in AKR-GW($\mathcal{I}_H$) as the sum of the dual variable increases in AKR-GW($\mathcal{I}_H$) that correspond to D_i-isolating clusters. More formally, call a time τ *interesting* in AKR-GW($\mathcal{I}_H$) if $\tau = 0$, if two clusters merge at time τ, or if some cluster becomes inactive at time τ. For future convenience, we also call a time τ interesting if two clusters merge in γ-AKR-GW($\widehat{\mathcal{I}}_H$) at the time $\gamma\tau$. As usual, an *epoch* is an interval between consecutive interesting moments of time and the set of active clusters remains unchanged within an epoch. Let $\mathcal{C}_j^I$ denote the clusters of AKR-GW($\mathcal{I}_H$) that are D_i-isolating in an epoch j that ends at or before the time T_i. Let $y_S^{(j)}$ denote the increment in the dual variables

y_S in the jth epoch of AKR-GW$(\mathcal{I}_H)$. The isolated cost share χ^* of D_i in AKR-GW$(\mathcal{I}_H)$ is formally defined as

$$\chi^* = \sum_{j=1}^{p} \sum_{S \in \mathcal{C}_j^I} y_S^{(j)}, \tag{27}$$

where p is the number of epochs that precede the time T_i. Following the proofs of Lemmas 3.30 and 3.31 gives the next lemma.

Lemma 3.49 *The isolated cost share χ^* of D_i in* AKR-GW$(\mathcal{I}_H)$ *satisfies*

$$\chi^* \leq \chi(\mathcal{I}, D_i).$$

Lemmas 3.48 and 3.49 reduce the proof of Theorem 3.47 to showing that

$$\ell_H(D_i) \leq \frac{4\gamma}{\gamma - 2} \cdot \chi^*.$$

The analogue of Lemma 3.30 implies that by some time $T_i^* \leq T_i$, all of the demands of D_i lie in a common cluster of AKR-GW$(\mathcal{I}_H)$. Since the set of tight edges in AKR-GW$(\mathcal{I}_H)$ is acyclic (see Lemma 3.17), there is a unique minimal tree A_i of tight edges that spans the demands of D_i at the time T_i^* in AKR-GW$(\mathcal{I}_H)$; we use A_i as a proxy for $\ell_H(D_i)$.

We now arrive at the point at which the proofs of Theorems 3.24 and 3.47 diverge in some small but important ways. First, the analogue of Lemma 3.32 is the following.

Lemma 3.50 *If the cluster S is active at time τ in* AKR-GW$(\mathcal{I}_H)$*, then the vertices that lie in both S and A_i form a subtree of A_i.*

While Lemmas 3.33–3.35 carry over without change to the present setting, the proof of Lemma 3.36 only gives the following.

Lemma 3.51 *Suppose at the time $\tau < T_i^*$ a cluster of* AKR-GW$(\mathcal{I}_H)$ *contains exactly one demand d_1 from D_i as well as an active demand d_2 not in D_i. Then d_1 and d_2 are in the same cluster of γ-*AKR-GW$(\widehat{\mathcal{I}}_H)$ *at the time 2τ.*

We will use the following corollary of Lemma 3.51.

Corollary 3.52 *Suppose the demands $d_1 \in D_i$ and $d_2 \notin D_i$ are active and in the same cluster of* AKR-GW$(\mathcal{I}_H)$ *at the time $\tau < T_i^*$. Then the cluster of γ-*AKR-GW$(\widehat{\mathcal{I}}_H)$ *that contains d_2 at the time 2τ also contains at least one demand of D_i.*

Proof: Lemma 3.51 implies that the cluster of γ-AKR-GW$(\widehat{\mathcal{I}}_H)$ that contains d_2 at the time 2τ also contains the first demand of D_i to share a cluster of AKR-GW$(\mathcal{I}_H)$ with d_2. ■

While Lemma 3.37 does not completely carry over to the present setting—in particular, an active cluster of AKR-GW$(\mathcal{I}_H)$ might contain active demands from distinct $\mathcal{P}^*$-groups—we can still classify the active clusters of AKR-GW$(\mathcal{I}_H)$ as *(D_i-)isolating*, *shared*, or *independent*. We can also define the maps Λ_j for each epoch j preceding T_i^* for active shared and independent clusters as in Subsection 3.3. Recall that for such a cluster S in a such

an epoch j, we pick (arbitrarily) a demand d that lies in S but not D_i and set Λ_j to the cluster that contains d in the jth epoch of γ-AKR-GW($\widehat{\mathcal{I}}_H$). (Recall also that epochs of γ-AKR-GW($\widehat{\mathcal{I}}_H$) are those of AKR-GW($\mathcal{I}_H$), scaled by a γ factor.) Corollary 3.52 and the proof of Lemma 3.38 then yield the following.

Lemma 3.53 *Fix an epoch j of* AKR-GW*($\mathcal{I}_H$) that concludes at or before the time T_i^*. The map Λ_j satisfies the following properties.*

(a) Λ_j is injective.

(b) Λ_j maps active clusters to active clusters.

(c) If S is an active independent cluster in the jth epoch of AKR-GW*($\mathcal{I}_H$), then $S \subseteq \Lambda_j(S)$.*

(d) If S is an active shared cluster in the jth epoch of AKR-GW*($\mathcal{I}_H$), then $\Lambda_j(S)$ contains at least one demand of D_i.*

The main consequence of our concessions in the above lemmas is that, in the language of the proof of Theorem 3.24, we can no longer precisely control the number $\mu_j(S)$ of "missing crossings" of A_i by $\Lambda_j(S)$, relative to those by a cluster S. In particular, we cannot establish analogues of Corollary 3.39 and Lemma 3.40 in the GST setting, which in turn will lead to a degradation in our strictness bound. Before completing the proof of Theorem 3.47 we state a final lemma, which follows from the argument in the proof of Lemma 3.41.

Lemma 3.54 *Let S be an active cluster of* AKR-GW*($\mathcal{I}_H$) in an epoch j that ends at or before time T_i^*. Suppose $\Lambda_j(S)$ contains all of the demands of D_i. Then every demand of D_i that lies in $\Lambda_j(S)$ but not S is in an isolating cluster in the jth epoch of* AKR-GW*($\mathcal{I}_H$).*

We now complete the proof of Theorem 3.47. As foreshadowed above, it differs from the proof of Theorem 3.24 primarily in that we bound the impact of missing crossings in a relatively crude way.

Proof of Theorem 3.47: Our goal is to show that

$$c(A_i) \leq \frac{4\gamma}{\gamma - 2} \cdot \chi^*, \tag{28}$$

where $c(A_i)$ is the cost of the tree A_i and χ^* is defined as in (27). For a cluster S, let $\kappa(S)$ denote the number of edges in both A_i and $\delta(S)$. Let $\mathcal{C}$ and $\widehat{\mathcal{C}}$ denote the possible clusters S of $\mathcal{I}_H$ and $\widehat{\mathcal{I}}_H$, respectively, with $\kappa(S) > 0$. For a cluster S, let $y_S^{(j)}$ and $z_S^{(j)}$ denote the increment in the dual variables y_S and z_S in the jth epochs of AKR-GW($\mathcal{I}_H$) and γ-AKR-GW($\widehat{\mathcal{I}}_H$), respectively. Let epoch p end at time T_i^*. First, the following crude bound holds (cf. (22)):

$$\sum_{j=1}^{p} \sum_{S \in \widehat{\mathcal{C}}} z_S^{(j)} \leq c(A_i). \tag{29}$$

As in (21), since A_i comprises only tight edges at time T_i^* in AKR-GW($\mathcal{I}_H$),

$$\sum_{j=1}^{p} \sum_{S \in \mathcal{C}} y_S^{(j)} \cdot \kappa(S) = c(A_i). \tag{30}$$

Next, we claim that for every epoch $j \leq p$, ignoring the number of crossings in (30) neglects at most half of the sum of the dual increments:

$$\sum_{S \in \mathcal{C}} y_S^{(j)} \geq \frac{1}{2} \sum_{S \in \mathcal{C}} y_S^{(j)} \cdot \kappa(S).$$

To prove this claim, fix an epoch $j \leq p$ and consider the set $\mathcal{C}_j \subseteq \mathcal{C}$ of clusters that are active in this epoch. Note that $y_S^{(j)}$ is the same—namely, the length of the jth epoch—for all clusters $S \in \mathcal{C}_j$. The claim is therefore equivalent to the assertion that the average value of $\kappa(S)$ among clusters in $\mathcal{C}_j$ is at most 2. By Lemma 3.50, each such cluster S intersects A_i in a subtree of A_i, with distinct clusters corresponding to vertex-disjoint subtrees. Obtain a new tree X from A_i by contracting each of these disjoint subtrees. Call a vertex x of X *contracted* if it corresponds to a cluster S of $\mathcal{C}_j$ and *original* otherwise. If x is a contracted vertex corresponding to the cluster S, then the degree of x in X is precisely $\kappa(S)$. Since X is a tree, the average degree of a vertex of X is at most 2. Since A_i is a minimal tree that spans the vertices of D_i, every leaf of A_i is a demand of D_i. Since every such demand lies in an active cluster of AKR-GW($\mathcal{I}_H$) in the epoch $j \leq p$, every leaf of X is a contracted vertex. Since original vertices of X all have degree at least 2, the average degree of a contracted vertex of X is at most 2, which completes the proof of the claim.

Combining the claim with (30) yields

$$\sum_{j=1}^{p} \sum_{S \in \mathcal{C}} y_S^{(j)} \geq \frac{c(A_i)}{2}; \tag{31}$$

the symmetry between (29) and (31) now allows us to proceed similarly to the proof of Theorem 3.24. Let $\mathcal{C}_j^I \subseteq \mathcal{C}$ denote the D_i-isolating clusters during the jth epoch of AKR-GW($\mathcal{I}_H$). For a shared or independent cluster $S \in \mathcal{C}_j \setminus \mathcal{C}_j^I$ that is active in an epoch $j \leq p$ of AKR-GW($\mathcal{I}_H$), let $\mu_j(S)$ equal 1 if $\Lambda_j(S) \notin \widehat{\mathcal{C}}$ (i.e., if $\kappa(\Lambda_j(S)) = 0$) and 0 otherwise. Crucially, parts (c) and (d) of Lemma 3.53 imply that $\mu_j(S) = 1$ for such a cluster only if $\Lambda_j(S)$ contains all of the demands of D_i.

Since epochs in γ-AKR-GW($\widehat{\mathcal{I}}_H$) are γ times as long as in AKR-GW($\mathcal{I}_H$), inequality (29) and Lemma 3.53(a) and (b) imply that

$$\sum_{j=1}^{p} \sum_{S \notin \mathcal{C}_j^I} \gamma \cdot y_S^{(j)} \cdot [1 - \mu_j(S)] \leq c(A_i). \tag{32}$$

Applying Lemma 3.54, the injectivity of Λ_j (Lemma 3.53(a)), and the definition (27) of χ^* then gives

$$\sum_{j=1}^{p} \sum_{S \notin \mathcal{C}_j^I} y_S^{(j)} \leq \frac{c(A_i)}{\gamma} + \chi^*. \tag{33}$$

Subtracting inequality (33) from inequality (31) gives

$$\sum_{j=1}^{p} \sum_{S \in \mathcal{C}_j^I} y_S^{(j)} \geq \frac{c(A_i)}{2} - \frac{c(A_i)}{\gamma} - \chi^*;$$

recalling the definition (27) of χ^* and rearranging establishes (28) and completes the proof. ■

3.4.3 A 12.66-Approximation Algorithm for MuRoB

Finally, we combine Theorems 3.45 and 3.47 to obtain an approximation algorithm for the MuRoB problem. Naively, there is a new source of error in the Sample-Augment algorithm for the MuRoB problem: greedily renting capacity in the augmentation step now corresponds to solving the NP-hard Steiner Tree problem, and therefore requires an approximation algorithm. The obvious analogue of Theorem 2.10 for the MuRoB problem is: if a β-strict α-approximation algorithm for GST is used in the subproblem step of Sample-Augment, and a γ-approximation algorithm for Steiner Tree is used in the augmentation step, then Sample-Augment is a randomized $(\alpha + \beta \cdot \gamma)$-approximation algorithm for MuRoB.

Tracing through the proof of Theorem 3.47, however, we see that it can be used to give a polynomial-time implementation of the augmentation step of the Sample-Augment algorithm. Since the strictness guarantee of Theorem 3.47 applies directly to this particular implementation, there is no further loss in approximation ratio and the bound of $\alpha + \beta$ from Theorem 2.10 applies. Precisely, by implementing the subproblem and augmentation steps of the Sample-Augment algorithm with the γ-AKR-GW subroutine and the subroutine implicit in the corresponding strictness proof (Theorem 3.47), respectively, the following bound applies to the Sample-Augment algorithm for the MuRoB problem.

Theorem 3.55 *For every $\gamma > 2$, there is a randomized $[\gamma + 1 + 4\gamma/(\gamma - 2)]$-approximation algorithm for the* MuRoB *problem.*

Choosing $\gamma = 2 + 2\sqrt{2}$, we obtain an approximation ratio of $7 + 4\sqrt{2} \approx 12.66$.

4 Virtual Private Network Design

In this section and the next, we show that strict cost-sharing methods lead to improved approximation algorithms for two problems to which our analysis framework does not directly apply. In this section, we build on our algorithm and analysis for the SSRoB problem and give a simple 5.55-approximation algorithm for the VPND problem. We study the SSBaB problem in the next section.

4.1 The VPND Algorithm

Recall from Subsection 1.1 that in an instance of the VPND problem (Problem 1.2) we are given *thresholds* $b_{in}(j)$ and $b_{out}(j)$ on the amount of traffic that enters and leaves each demand

Input: an VPND instance (G, D, b).
Assumptions: each demand $j \in D$ is either a sender or a receiver; there are more receivers than senders.

1. (*Sampling step*) Pick a sender $\hat{s}$ uniformly at random.

2. (*Subproblem step*) Use the algorithm Sample-Augment to compute a feasible solution to the SSRoB instance $(G, \mathcal{D}, 1, M)$, where $\mathcal{D}$ is the set of all pairs of the form $(r, \hat{s})$ for a receiver r, 1 the vector of unit weights, and M is the number of senders. Let F denote the edges bought by the algorithm. For every edge $e \in F$, set $u_e = M$; for every other edge e, set u_e equal to the amount of capacity rented for e by the Sample-Augment algorithm.

3. (*Augmentation step*) Greedily and independently reserve one unit of capacity from each sender other than $\hat{s}$ to F.

Figure 5: The algorithm VPN-Sample-Augment.

$j \in D \subseteq V$ of a network $G = (V, E)$ with edge costs c_e. The objective is to design a network will sufficient capacity for every traffic pattern that respects these upper bounds. Formally, a traffic pattern is specified by a $D \times D$ matrix of nonnegative real numbers, with entry f_{ij} denoting the amount of traffic sent from demand i to demand j. A traffic matrix is *valid* if for every demand j, the amount of traffic $\sum_i f_{ij}$ incoming to j is at most $b_{in}(j)$ and the amount $\sum_i f_{ji}$ of outgoing traffic is at most $b_{out}(j)$. We assume that all of these thresholds are rational numbers. By scaling both these thresholds and the edge costs of G, we can then assume, without loss of generality, that these thresholds are integral.

A feasible solution to a VPND instance reserves capacity u_e on each edge e of the graph G, and selects paths P_{ij} between each ordered pair $i, j \in D$ of demands so that all valid traffic matrices can be routed using these paths without violating the reserved capacities. The cost of a solution is $\sum_e c_e u_e$ and we seek a solution of minimum cost.

To simplify our exposition, we assume for most of this section that each demand j is a either a *sender* (with $b_{in}(j) = 0$ and $b_{out}(j) = 1$) or a *receiver* (with $b_{in}(j) = 1$ and $b_{out}(j) = 0$). In Remark 4.9, we indicate how to extend our algorithm and analysis to general VPND instances. We will also assume that the receivers of the VPND instance outnumber the senders; the algorithm and analysis in the other case are symmetric.

Figure 5 presents our algorithm for the VPND problem, which we call VPN-Sample-Augment. Its high-level outline is the same as for the Sample-Augment algorithm. Given an instance $\mathcal{I}$ of VPND, we first define a random subproblem, which in this case is an instance $\mathcal{I}_{SSRoB}$ of SSRoB. The only random parameter of $\mathcal{I}_{SSRoB}$ is the sink vertex, which is a sender $\hat{s}$ of $\mathcal{I}$ that is picked uniformly at random. The source vertices of $\mathcal{I}_{SSRoB}$ are defined to be the receivers of $\mathcal{I}$, and each corresponding demand pair is given unit weight. Finally, the cost M of buying capacity on an edge is defined to be the number of senders. We then solve the random subproblem $\mathcal{I}_{SSRoB}$ with the Sample-Augment algorithm of Subsection 3.1. We interpret the resulting feasible solution of $\mathcal{I}_{SSRoB}$ as follows. Let F be the set of edges

on which the Sample-Augment subroutine bought capacity. In our VPND solution, we reserve M units of capacity on each edge $e \in F$. If the Sample-Augment algorithm rents capacity for an edge e, then in our VPND solution we reserve the same amount of capacity on e. Finally, we greedily augment this partial solution to a feasible solution for the VPND instance $\mathcal{I}$ as follows: independently for each sender $s \neq \hat{s}$, reserve one unit of capacity for s's exclusive use on a shortest path between s and F. For each sender s and receiver r, the s-r path P_{sr} is defined as the concatenation of s's shortest path to F, a path through F to $\hat{s}$, and the $\hat{s}$-r path defined by the Sample-Augment subroutine's solution to the instance $\mathcal{I}_{SSRoB}$.

Next we prove some basic facts about the algorithm VPN-Sample-Augment. For the remainder of the analysis, fix an instance $\mathcal{I} = (G, D, b)$ of VPND that satisfies our two standing assumptions. Let R and S denote the sets of receivers and senders of $\mathcal{I}$, respectively. Let F denote the set of edges bought by the Sample-Augment algorithm in the subproblem step of VPN-Sample-Augment.

Lemma 4.1 *The algorithm* VPN-Sample-Augment *produces a feasible solution with probability 1.*

Proof: Fix a valid demand matrix $\{f_{sr}\}_{s \in S, r \in R}$. We need to show that routing f_{sr} units of flow on the path P_{sr} defined above for every $s \in S$ and $r \in R$ does not violate any capacity constraint (with probability 1). We first claim that no edge $e \in F$ bought by the Sample-Augment subroutine in the subproblem step is used beyond its capacity. This follows because M units of capacity are reserved on each such edge and, since there are only M senders, $\sum_{s,r} f_{sr} \leq \sum_s b_{out}(s) = M$.

On the other hand, the VPN-Sample-Augment algorithm explicitly reserves capacity on each edge outside F for each path that uses it. In more detail, for every sender s, all paths of the form P_{sr} begin with a shortest path from s to F, and the augmentation step of the VPN-Sample-Augment algorithm reserves one unit of capacity on this subpath for exclusive use by s. Since $\sum_r f_{sr} \leq b_{out}(s) = 1$, there is sufficient capacity for the traffic on these subpaths. Similarly, for each receiver r, all paths of the form P_{sr} conclude with the $\hat{s}$-r path P_r defined by the Sample-Augment algorithm's solution to the instance $\mathcal{I}_{SSRoB}$. Moreover, $\sum_s f_{sr} \leq 1$. By the definition of the augmentation step of the Sample-Augment algorithm, there is one unit of capacity on the edges of $P_r \setminus F$ reserved for exclusive use by the sender r. There is thus sufficient capacity on every edge for the traffic of every path P_{sr}, and the proof is complete. ■

Also, the union of the routing paths produced by the VPN-Sample-Augment algorithm form a tree with probability 1.

Lemma 4.2 *If a consistent tie-breaking rule is used to compute shortest paths, then with probability 1 the algorithm* VPN-Sample-Augment *produces a solution in which the edges with non-zero capacity form a tree.*

Proof: Since the set F is the output of a Steiner Tree instance algorithm, it is (or can be assumed to be) a tree. By the definition of the augmentation steps of the Sample-Augment and VPN-Sample-Augment algorithms, all other edges with non-zero capacity lie on a

shortest path between a demand j and the set F—equivalently, are contained in the shortest-path tree in the contracted graph G/F rooted at the vertex corresponding to F. This implies that if a consistent tie-breaking rule is used to compute shortest paths, the set of all edges with non-zero capacity forms a tree. ■

4.2 Analysis

We now bound the expected cost of the solution produced by the VPN-Sample-Augment algorithm for the VPND instance $\mathcal{I}$. We will do this by bounding three parts of this cost separately: the expected cost corresponding to the set F of edges bought by the Sample-Augment subroutine in the subproblem step; the expected cost corresponding to the rented edges in the subproblem step; and the expected cost of the augmentation step. The first two steps hinge on the following lemma, which bounds the expected cost of an optimal solution to the (random) instance of Steiner Tree that arises in the subproblem step of the Sample-Augment subroutine (cf., Lemma 2.2).

Lemma 4.3 *Let OPT_{VPN} denote the cost of an optimal solution for the* VPND *instance $\mathcal{I}$. Let $OPT_{\hat{s},\hat{R}}$ denote the cost of an optimal solution for the* Steiner Tree *instance in the subproblem step of the* Sample-Augment *subroutine, given the random choices of the sender $\hat{s} \in S$ and receivers $\hat{R} \subseteq R$ in the sampling steps of the* VPN-Sample-Augment *and* Sample-Augment *algorithms, respectively. Then*

$$\mathbf{E}\left[OPT_{\hat{s},\hat{R}}\right] \leq \frac{OPT_{VPN}}{M}, \tag{34}$$

where the expectation is over the random choices of $\hat{s}$ and $\hat{R}$.

Proof: We begin with the following equivalent description of the random choices made in the sampling steps of the VPN-Sample-Augment and Sample-Augment algorithms. Suppose each receiver picks a sender independently and uniformly at random. Let $D_s \subseteq R$ denote the random set of receivers that pick the sender s. Then, independently choose a sender $\hat{s}$ uniformly at random and consider the Steiner Tree instance $\mathcal{I}_{\hat{s}}$ defined by $D_{\hat{s}} \cup \{\hat{s}\}$. We claim that this random process induces the same distribution over Steiner Tree instances that the algorithm VPN-Sample-Augment does. In both processes, one sender $\hat{s}$, chosen uniformly at random from the set of all senders, is included in the Steiner Tree instance. In the VPN-Sample-Augment algorithm, each receiver has a $1/M$ probability of being included in the Steiner Tree instance by the definition of the sampling step of the Sample-Augment subroutine. In the new random process, since there are M senders, the probability that a receiver picks the sender $\hat{s}$ and is included in the resulting Steiner Tree instance is also $1/M$. Moreover, these events are independent of each other and of the choice of the sender $\hat{s}$, just as in the VPN-Sample-Augment algorithm. The two random processes therefore induce the same distribution over Steiner Tree instances, and we can prove the lemma by establishing (34) for the new random process above.

We now prove that the expected cost of an optimal solution to the random Steiner Tree instance $\mathcal{I}_{\hat{s}}$ is at most OPT_{VPN}/M. We will prove this inequality after conditioning on the

partition $\{D_s\}_{s \in S}$ of receivers, with the expectation only over the choice of $\hat{s}$; the unconditional inequality (34) then follows. Fix an optimal solution to the VPND instance $\mathcal{I}$ that reserves the paths $\{P^*_{sr}\}_{s \in S, r \in R}$ and capacities $\{u^*_e\}_{e \in E}$. We next show how to pack feasible solutions for all M of the Steiner Tree instances $\{\mathcal{I}_s\}_{s \in S}$ into this optimal solution.

For each sender $s \in S$, let G^*_s denote the subgraph of G with the edge set $\cup_{r \in D_s} P^*_{sr}$. Since G^*_s spans $D_s \cup \{s\}$, the cost $c(G^*_s)$ of the subgraph G^*_s is at least denote the value OPT_s of an optimal solution to $\mathcal{I}_s$. Moreover, if an edge e appears in k subgraphs of the form G^*_s, then it is a member of k sender-receiver paths that share no endpoints. Since simultaneous routing of traffic on these k paths must be supported, OPT_{VPN} must install at least k units of capacity on the edge e. Therefore,

$$OPT_{VPN} \geq \sum_{s \in S} c(G^*_s) \geq \sum_{s \in S} OPT_s.$$

Thus, if we pick a sender uniformly at random from the M senders, $\mathbf{E}_s[OPT_s] \leq OPT_{VPN}/M$, which completes the proof. ■

A proof identical to that of Lemma 2.3 bounds the expected cost incurred by the VPN-Sample-Augment algorithm for bought edges in its subproblem step.

Lemma 4.4 *If an α-approximation algorithm for* Steiner Tree *is used in the subproblem step of the* Sample-Augment *subroutine, then the expected cost incurred by the* VPN-Sample-Augment *algorithm for bought edges in its subproblem step is at most $\alpha \cdot OPT_{VPN}$.*

We next use the universally strict cost shares for Steiner Tree (Subsection 3.1) to bound the expected cost incurred by the VPN-Sample-Augment algorithm in the subproblem step for edges that were rented by its Sample-Augment subroutine.

Lemma 4.5 *The expected cost incurred by the* VPN-Sample-Augment *algorithm for rented edges in its subproblem step is at most $2 \cdot OPT_{VPN}$.*

Proof: Let C denote the cost paid by the VPN-Sample-Augment algorithm for rented edges in its subproblem step. Recall from Definition 3.2 and Example 3.3 that the Prim cost-sharing method of Example 2.8 is universally 2-strict. In particular, Lemma 3.4 implies that these cost shares are 2-strict no matter what Steiner Tree algorithm is used in the subproblem step of the Sample-Augment algorithm.

We next condition on the choice of $\hat{s}$ in the sampling step of the VPN-Sample-Augment algorithm. For a subset $\hat{R} \subseteq R$ of receivers, let $OPT_{\hat{s},\hat{R}}$ denote the value of a minimum-cost Steiner tree spanning $\hat{s}$ and all of the receivers in $\hat{R}$. The proof of Lemma 2.9, and the inequalities (4) and (8) in particular, imply that

$$\mathbf{E}_{\hat{R}}[C|\hat{s}] \leq 2M \cdot \mathbf{E}_{\hat{R}}\left[OPT_{\hat{s},\hat{R}}\right],$$

where the expectations are over the random choice of the set $\hat{R}$ of receivers in the Sample-Augment subroutine's sampling step. Taking expectations over the choice of $\hat{s}$, we obtain

$$\mathbf{E}_{\hat{s},\hat{R}}[C] \leq 2M \cdot \mathbf{E}_{\hat{s},\hat{R}}\left[OPT_{\hat{s},\hat{R}}\right] \leq 2 \cdot OPT_{VPN},$$

where the second inequality follows from Lemma 4.3. The proof is complete. ■

Our final lemma bounds the expected cost of the augmentation step of the VPN-Sample-Augment algorithm.

Lemma 4.6 *The expected cost incurred in the augmentation step of the* VPN-Sample-Augment *algorithm is at most* $2 \cdot OPT_{VPN}$.

Proof: Since the set F of bought edges contains the sender $\hat{s}$, we can prove the lemma by showing that, if a sender $\hat{s}$ is picked uniformly at random, then

$$\mathbf{E}\left[\sum_{s \in S} \ell(s, \hat{s})\right] \leq 2 \cdot OPT_{VPN},$$

where $\ell(\cdot, \cdot)$ denotes shortest-path distance in G. To prove this inequality, we fix a set $\hat{R} \subseteq R$ of M receivers. Every perfect matching $\mathcal{M}$ of S and $\hat{R}$ provides a lower bound $\sum_{(s,r) \in \mathcal{M}} \ell(s, r)$ on OPT_{VPN}, since a feasible solution must support the simultaneous communication of all of the matched pairs of $\mathcal{M}$. Averaging over all of the $M!$ possible perfect matchings of S and $\hat{R}$, we obtain

$$\frac{1}{M} \sum_{s \in S, r \in \hat{R}} \ell(s, r) \leq OPT_{VPN},$$

as each sender-receiver pair (s, r) appears in $(M-1)!$ of the $M!$ perfect matchings. This inequality implies that

$$\mathbf{E}_{\hat{s}}\left[\sum_{r \in \hat{R}} \ell(\hat{s}, r)\right] \leq OPT_{VPN}. \tag{35}$$

Also, by the Triangle inequality for shortest-path distances,

$$\sum_{s \in S} \ell(s, \hat{s}) \leq \sum_{r \in \hat{R}} \ell(\hat{s}, r) + \sum_{(s,r) \in \mathcal{M}} \ell(s, r) \leq \sum_{r \in \hat{R}} \ell(\hat{s}, r) + OPT_{VPN}, \tag{36}$$

where $\mathcal{M}$ is an arbitrary perfect matching of S and $\hat{R}$. Taking expectations (over the choice of $\hat{s}$) in (36) and combining with (35) proves the lemma. ■

Combining Lemmas 4.4–4.6 with the 1.55-approximation algorithm for the Steiner Tree problem due to Robins and Zelikovsky [58] yields the main theorem of this section.

Theorem 4.7 *There is a randomized 5.55-approximation algorithm for the* VPND *problem.*

Lemma 4.2 states that the VPN-Sample-Augment algorithm always outputs a tree solution. Our analysis of the algorithm, however, does not assume that the paths chosen by the optimal solution form a tree. Indeed, there are instances in which no optimal solution forms a tree [33]. Theorem 4.7 implies that for every instance of VPND, there is a tree solution within a (small) constant factor of the optimal (graph) solution. This resolves one of the main open questions from [33].

Corollary 4.8 *Every instance of* VPND *admits a tree solution with cost no more than 5.55 times that of an optimal (graph) solution. Moreover, this solution can be computed in polynomial time.*

If the constraint of polynomial-time computation is dropped, then the constant in Corollary 4.8 can be improved to 5 by using an (exponential-time) optimal Steiner Tree subroutine in the VPN-Sample-Augment algorithm.

Remark 4.9 The VPN-Sample-Augment algorithm and its analysis extend to the case of arbitrary (integral) thresholds b_{in} and b_{out} as follows. Given an instance of VPND, suppose we modify the instance by splitting each demand j into $b_{in}(j)$ receivers and $b_{out}(j)$ senders, all of which are co-located. This increases the set of feasible solutions, since it allows the traffic of an original demand pair to be routed on more than one path. The modification can therefore only decrease the cost of an optimal solution. On the other hand, if the VPN-Sample-Augment algorithm uses a consistent tie-breaking rule for computing shortest paths as in Lemma 4.2, then it will output a solution for the modified instance that is also feasible for the original instance. Running the VPN-Sample-Augment algorithm after splitting demands into senders and receivers therefore produces a feasible solution to the original instance that is at most 5.55 times as costly as an optimal solution (for the original or the modified instance).

Splitting demands into senders and receivers is only a polynomial transformation if all of the demand thresholds are polynomially bounded. However, by adjusting the sampling probabilities in the sampling steps of the VPN-Sample-Augment algorithm and its Sample-Augment subroutine, we can easily modify the algorithm to mimic its behavior on the modified instance in polynomial time.

5 Single-Sink Buy-at-Bulk Network Design

This section gives a simple constant-factor approximation algorithm for the widely studied SSBaB problem. Our algorithm is closely related to that of Guha, Meyerson, and Munagala [32], but the analysis tools developed in this paper permit a tighter and equally simple analysis. Subsection 5.1 introduces notation for our analysis and reviews some well-known transformations of SSBaB instances. Subsection 5.2 presents our algorithm and analysis.

5.1 Preliminaries

Recall that an instance of the SSBaB problem (Problem 1.3) comprises an undirected graph G and edge costs c; a set $\mathcal{D}$ of demand pairs $\{(s_i, t)\}_{i=1}^{k}$; a weight $w_i \geq 0$ for each demand pair (s_i, t), denoting the amount of flow that s_i wants to send to t; and K *cable types* $\{1, 2, \ldots, K\}$, where the jth cable has capacity u_j and cost σ_j per cable per unit length. The goal is to compute a minimum-cost way of installing cables so that there is sufficient capacity for all sources to route flow simultaneously.

Fix an instance $\mathcal{I}$ of SSBaB. We will assume that each parameter u_j and σ_j is a power of 2. Similarly to [32], this assumption can be enforced while losing a factor of 4 in the

approximation ratio, by rounding each capacity u_j down to the nearest power of 2 and each σ_j up to the nearest power of 2. By scaling and reordering cable types, we can assume that $1 = u_1 < \cdots < u_K$ and $1 = \sigma_1 < \cdots < \sigma_K$; if $u_i \leq u_j$ and $\sigma_i \geq \sigma_j$, then cable type i is redundant and can be eliminated.

Define $\delta_j = \sigma_j/u_j$, which intuitively is the "incremental cost" of using cable type j. For all j, δ_j is a power of 2. We can assume that $\delta_1 > \ldots > \delta_K$, since if $\delta_i \leq \delta_j$ for some $i < j$, then cable type j is redundant and can be eliminated.

Finally, we define $g_j = \frac{\sigma_{j+1}}{\sigma_j} u_j$. Since $\delta_j > \delta_{j+1}$, $g_j < u_{j+1}$ and hence

$$1 = u_1 < g_1 < u_2 < g_2 < \ldots < u_K < g_K = \infty. \tag{37}$$

Next, we would like to assume that all weights w_i are integral. This assumption is not without loss of generality, as we have already scaled the cable capacities. Instead, we enforce this assumption with the following "redistribution lemma." Roughly speaking, this lemma shows how to take a grouping parameter U, along with a tree with weights on its vertices, and randomly move weights throughout the tree so that the total weight at every node of the tree becomes either 0 or U. (For ensuring integral demands, we will take U to be 1). Moreover, this random process has two important properties: the probability that a vertex in the tree receives weight U is proportional to its initial weight, and no edge of the tree carries too much flow during the reallocation.

Lemma 5.1 (Redistribution Lemma) *Let T be a tree and $U > 0$ a parameter. Suppose each vertex $j \in T$ has a nonnegative weight $w_j < U$ and that the sum $\sum_j w_j$ of the weights is a multiple of U. Then there is an efficiently computable (random) flow f in T with the following properties.*

(a) With probability 1, f sends at most U units of flow across each edge of T.

(b) After rerouting weights according to the flow f, for every vertex $j \in T$, the new weight of j is U with probability w_j/U and 0 with probability $1 - w_j/U$.

A deterministic version of this lemma appears in [40, Lemma 1]. We include the simple proof for completeness.

Proof: Replace each edge of T by two oppositely directed arcs. We first show that the lemma holds in this bidirected tree $\widetilde{T}$. We start by rooting $\widetilde{T}$ at an arbitrary vertex r and taking an Euler tour of $\widetilde{T}$ starting at r. Order the vertices $j_1, \ldots, j_n$ of T according to their first appearance in this Euler tour. For each $i \in \{1, 2, \ldots, n\}$, let W_i denote the sum of the weights of the first i vertices in this ordering. Define W_0 to be 0.

Pick a value Y drawn uniformly at random from $(0, U]$. Call vertex j_i *unlucky* if for some integer x, $W_{i-1} < xU + Y \leq W_i$—if the running sum of weights just crossed the point Y modulo U—and *lucky* otherwise. After this procedure concludes, we define the flow $\widetilde{f}$ to reroute weights as follows. If a vertex j_i is lucky, we add a flow path to $\widetilde{f}$ that routes all of j_i's weight to the unlucky vertex that is next according to the ordering $j_{i+1}, \ldots, j_n, j_1, \ldots, j_{i-1}$. Otherwise, the vertex j_i is only allowed to route $W_i - (xU + Y)$ units of its weight to the next unlucky vertex, where x is the integer defined above.

After this rerouting, a vertex has weight U if it is unlucky and weight 0 if it is lucky. The probability that the vertex j is unlucky is precisely w_j/U. Thus the flow $\widetilde{f}$ satisfies part (a) of the lemma. The flow need not satisfy part (b), however: while $\widetilde{f}$ routes at most U units of flow on each arc of $\widetilde{T}$, this corresponds to routing at most $2U$ units of flow on each edge of the original undirected tree T. But since $\widetilde{f}$ routes at most U units of flow in each direction across each edge of T, we can perform rudimentary flow-canceling independently on each edge of T. This yields a flow f in T that satisfies part (b) of the lemma and, since it redistributes weights identically to $\widetilde{f}$, also satisfies part (a). ■

We will use Lemma 5.1 as a preprocessing step to collect integral demands at some subset of the sources of the instance $\mathcal{I}$. First, we can assume that the sum of the demand pair weights in $\mathcal{I}$ is greater than 1; otherwise even the cheapest cable type effectively has infinite capacity, and $\mathcal{I}$ is equivalent to a Steiner Tree instance. We also assume that the sum W of the demand pair weights in $\mathcal{I}$ is a power of 2 and is at least u_K; this assumption can be removed by adding a dummy demand pair (t, t) with an appropriate weight, and by modifying the following algorithm and analysis to ensure that this dummy weight never leaves the sink t.

As a preprocessing step of the algorithm in the next subsection, we use an α-approximation algorithm for the Steiner Tree problem to compute a tree T_0 that spans all of the sources, and build one cable of type 1 on each edge of T_0. We then apply Lemma 5.1 to the tree T_0, with $U = 1$ and the weight of the source s_i defined as the fractional part $w_i - \lfloor w_i \rfloor$ of its weight in $\mathcal{I}$. After this procedure concludes, there is an integral amount of weight at every source of $\mathcal{I}$.

We now bound the cost of T_0. Fix an optimal solution to $\mathcal{I}$ and let OPT denote its cost. Let $C^*(j)$ denote the cost of the cables of type j in this solution. Note that $OPT = \sum_{j=1}^{K} C^*(j)$. This solution must install nonzero capacity on a subgraph G^* of G that spans all of the sources of $\mathcal{I}$. Thus one candidate for a Steiner Tree solution T_0 is to build one type 1 cable on each edge of G^*. Since $\sigma_1 = 1$, the cost of this candidate solution is at most

$$\sum_{j=1}^{K} \frac{C^*(j)}{\sigma_j}. \tag{38}$$

Since we use an α-approximation algorithm to compute the Steiner tree solution T_0, the cost of T_0 is at most α times the quantity in (38).

5.2 The Algorithm SSBaB-Sample-Augment

We now present our constant-factor approximation algorithm for the SSBaB problem. The algorithm is similar to that of Guha, Meyerson, and Munagala [32], where the network is designed incrementally in stages. At the beginning of each stage j there will be a set of *demands*, each of which represents a group of u_j units of traffic that must be routed to the sink. During the jth stage, we use the value u_{j+1} as an "aggregation threshold", and reroute groups of u_{j+1}/u_j demands (each of weight u_j) into a single demand of weight u_{j+1}. We buy cables on the paths required for this agglomeration. At the end of all of the stages, every

demand reaches the sink. The final solution is the union of all of the cables bought in all of the stages. Since this capacity is sufficient to move all of the prescribed traffic from the sources to the sink (via the concatenation of the rerouting paths used in each stage of the algorithm), this solution is feasible.

Let W denote the sum of the demand pair weights; recall from Subsection 5.1 that we can assume that W is a power of 2. Our preprocessing step from Subsection 5.1 ensures that at the beginning of the first stage there is an integral weight at every source vertex. If the source s_i has weight w_i at the beginning of the first stage, we interpret this as w_i co-located demands, each of weight 1. Let D_1 denote the set of these unit-weight demands. While naively replicating demands could result in a pseudopolynomial-time algorithm, non-uniform sampling can be added to the SSBaB-Sample-Augment algorithm to simulate the effect of this replication in polynomial time (see also Remark 4.9).

More generally, at the beginning of the jth stage, there is a set D_j of W/u_j demands, located at the source vertices of $\mathcal{I}$, with weight u_j each. We now describe each stage j of the algorithm in more detail; see also Figure 6. In the sampling step, we choose a random subset $\widehat{D}_j \subseteq D_j$ of demands, with each demand of D_j picked independently with probability $p_j = u_j/g_j = \sigma_j/\sigma_{j+1}$. Note that the sampling probability p_j is the ratio between the costs of the relatively low-capacity type j cables and relatively high-capacity type $(j+1)$ cables, analogous to the sampling step in the algorithm Sample-Augment for rent-or-buy problems. In the subproblem step, we compute a Steiner tree T_j spanning the set F_j, which is the union of the sink t and the source vertices that contain a demand of $\widehat{D}_j$. We build one cable of type $(j+1)$ on each edge of T_j. In the augmentation step, we route the demands outside $\widehat{D}_j$ to vertices of F_j along shortest paths, while building cables of type j on these shortest paths. In the gathering step, for each co-located group of u_{j+1}/u_j demands, we send all of these demands back to the originating location (at the beginning of this stage) of one of them, chosen uniformly at random. This group of u_{j+1}/u_j demands is then treated as a single demand of D_{j+1} with weight u_{j+1} in the next stage. Finally, the rounding step is like the preprocessing step of Subsection 5.1 and uses Lemma 5.1 to gather the remaining demands into groups of u_{j+1}/u_j demands. Each such group is then rerouted as in the gathering step, and is a single demand of D_{j+1} of weight u_{j+1} in the next stage. In the Kth stage, $g_K = \infty$ and $p_K = 0$. Thus, the sampling step of the final stage is vacuous and all demands are sent to the sink t in the augmentation step.

Each demand d of D_{j+1} can be naturally associated with a demand of D_j—the demand that participated in the complete group of u_{j+1}/u_j demands of D_j that corresponds to d, and that was randomly chosen in the gathering or rounding step. Put differently, we can view the jth stage of the algorithm as, for each complete group of u_{j+1}/u_j demands identified in the gathering and rounding steps, multiplying the weight of a random such demand by a u_{j+1}/u_j factor and discarding the rest of them. We can thus sensibly write $D_{j+1} \subseteq D_j$ for every $j \in \{1, 2, \ldots, K-1\}$. Finally, recall that D_1 is the result of the preprocessing step of Subsection 5.1 and is not the original set of demands of $\mathcal{I}$. Define D_0 as the initial set of demands, with each demand pair (s_i, t) with weight w_i of $\mathcal{I}$ giving rise to $\lceil w_i \rceil$ demands of D_0 ($\lfloor w_i \rfloor$ unit-weight demands and one demand with weight $w_i - \lfloor w_i \rfloor$). Lemma 5.1(b) implies that the probability that a demand of D_0 is also in D_1 is exactly its weight.

We now analyze the algorithm on the fixed SSBaB instance $\mathcal{I}$ with a sequence of lemmas.

1. (*Sampling step*) Construct a random subset $\widehat{D}_j$ of the demands in D_j by choosing each such demand independently with probability $p_j = u_j/g_j = \sigma_j/\sigma_{j+1}$.

2. (*Subproblem step*) Let F_j denote the union of the sink and the sources that contain a demand from $\widehat{D}_j$. Construct an α-approximate Steiner tree T_j that spans F_j. Install a cable of type $(j+1)$ on each edge of T_j.

3. (*Augmentation step*) For each demand in D_j, route its u_j weight to the closest vertex in F_j. Install one cable of type j on each edge of this shortest path.

4. *(Gathering step)* For each vertex $v \in F_j$, split the demands at v into *complete groups* of u_{j+1}/u_j demands plus one *residual group* of $r_v < u_{j+1}/u_j$ demands. Route each complete group back to the initial location (at the beginning of this stage) of one of the u_{j+1}/u_j contributing demands, chosen uniformly at random. Install cables of type $j+1$ to provide sufficient capacity.

5. *(Rounding step)* Use Lemma 5.1 with the tree T_j, the parameter $U = u_{j+1}$, and the weights of the residual groups, to aggregate the weight of all of the residual groups into complete groups of u_{j+1}/u_j demands, each with total weight exactly u_{j+1}. Reroute a complete group at the vertex $v \in F_j$ back to the initial location of one of the r_v demands that were routed to v in the augmentation step, chosen uniformly at random. Again, build new cables of type $j+1$ to provide sufficient capacity.

Figure 6: The jth stage of the algorithm SSBaB-Sample-Augment.

Lemma 5.2 *For every unit-weight demand $d \in D_1$ and every stage $j \in \{1, 2, \ldots, K\}$,*

$$\mathbf{Pr}[d \in D_j] = \frac{1}{u_j}.$$

Proof: The proof is by induction. The lemma is clearly true when $j = 1$. For $j > 1$, we have

$$\mathbf{Pr}[d \in D_j] = \mathbf{Pr}[d \in D_j \mid d \in D_{j-1}] \cdot \mathbf{Pr}[d \in D_{j-1}].$$

Since $\mathbf{Pr}[d \in D_{j-1}] = 1/u_{j-1}$ by the inductive hypothesis, we only need to show that $\mathbf{Pr}[d \in D_j \mid d \in D_{j-1}] = u_{j-1}/u_j$. If d is gathered into a complete group of u_j/u_{j-1} demands in the gathering step of stage $(j-1)$ of the algorithm, then this equality holds because every such demand is equally likely be chosen for membership in D_j. Suppose d is gathered into a residual group of $r_v < u_j/u_{j-1}$ demands at the vertex $v \in F_{j-1}$ in the gathering step of stage $(j-1)$ of the algorithm. Then d is included in D_j if and only if the Redistribution Lemma gathers a complete group of demands at the vertex v in the rounding step and then d is chosen for membership in D_j from the r_v demands in the residual group at v. By Lemma 5.1(b), the probability of both events occurring is

$$\frac{r_v u_{j-1}}{u_j} \cdot \frac{1}{r_v} = \frac{u_{j-1}}{u_j},$$

which completes the proof of the lemma. ■

Lemma 5.2 implies that for every stage $j \in \{1, 2, \ldots, K\}$, a demand $d \in D_1$ lies in the set $\widehat{D}_j$ with probability $p_j \times 1/u_j = 1/g_j$. The probability that a demand $d \in D_0$ with weight $w \leq 1$ lies in the set $\widehat{D}_j$ is thus w/g_j.

The next lemma bounds the expected cost of an optimal solution to the Steiner Tree instance arising in the subproblem step of each stage of the SSBaB-Sample-Augment algorithm (cf., Lemmas 2.2 and 4.3).

Lemma 5.3 *For a stage $j \in \{1, 2, \ldots, K-1\}$, let T_j^* be a minimum-cost Steiner tree spanning F_j with cost $c(T_j^*)$. Then*

$$\mathbf{E}\left[c(T_j^*)\right] \quad \leq \quad \sum_{i=j+1}^{K} \frac{C^*(i)}{\sigma_i} + \frac{1}{g_j} \sum_{i=1}^{j} \frac{C^*(i)}{\delta_i}, \tag{39}$$

where $C^*(i)$ denotes the cost of the cables of type i in a fixed optimal solution to $\mathcal{I}$, and the expectation is over the choice of $\widehat{D}_j$.

Proof: As in the proof of Lemma 2.2, we will exhibit a (random) subgraph G_j of G that spans F_j and has low expected cost. Fix an optimal solution for $\mathcal{I}$ and a feasible way of routing all of the traffic with respect to this solution. We first add to G_j all of the edges in the optimal solution that possess a cable of type $j+1$ or higher. The cost of these edges is (deterministically) at most the first sum on the right-hand side of (39).

We complete G_j by considering each demand d of $\widehat{D}_j$ in turn. In the fixed optimal solution, the traffic of the corresponding demand $d \in D_0$ may be routed on multiple paths. (We unfortunately cannot assume without loss of generality that an optimal solution is a tree.) We randomly add to G_j one of these paths, with a path chosen with probability equal to the fraction of d's traffic that it carries.

We now bound the expected cost of adding these edges to G_j. Consider an edge e of G with no cable of type $j+1$ or higher in the optimal solution. First suppose that only one cable is installed on e, say of type $i \leq j$. Then e is included in the random subgraph G_j if and only if the following events occur: for some demand $d \in D_0$ and some path P that routes some of d's traffic across the edge e, the demand d lies in $\widehat{D}_j$, and the path P is selected among all paths that route d's traffic. A demand $d \in D_0$ with weight $w \leq 1$ lies in $\widehat{D}_j$ with probability w/g_j, and a path P is chosen with probability x/w, where x is the amount of d's traffic that is routed on P in the optimal solution. The Union Bound then implies that e lies in G_j with probability at most f_e/g_j, where f_e is the total amount of flow on e in the optimal solution.

Since $f_e \leq u_i$, edge e contributes at most $c_e u_i/g_j$ to the expected cost of G_j. On the other hand, the cable of type i on edge e contributes $\sigma_i c_e$ to $C^*(i)$. Thus the expected cost in G_j for edge e is at most $1/(g_j\,\delta_i)$ times what the optimal solution pays for the cable. For edges on which the optimal solution installs multiple cables, this same analysis can be performed on a cable-by-cable basis. Summing over all edges with no cable of type $j+1$ or higher in the optimal solution gives the second sum on the right-hand side of (39) and proves the lemma. ■

We now relate the expected cost incurred by the SSBaB-Sample-Augment algorithm to the expected cost of an optimal Steiner tree spanning the vertices in F_j.

Lemma 5.4 *Let $j \in \{1, 2 \ldots, K-1\}$ be a stage and T_j^* a minimum-cost Steiner tree spanning F_j with cost $c(T_j^*)$. The expected cost incurred in stage j of the algorithm* SSBaB-Sample-Augment *is at most*

$$(3+\alpha)\,\sigma_{j+1}\,\mathbf{E}\,[c(T_j^*)],$$

where α is the approximation ratio of the Steiner Tree *algorithm used in the subproblem step.*

Proof: Since we install one cable of type $(j+1)$ on each edge of the tree T_j that we compute in the subproblem step, the expected cost incurred in this step is at most $\alpha\,\sigma_{j+1}\,\mathbf{E}\,[c(T_j^*)]$. As in Lemma 4.5, the universally 2-strict Prim cost shares of Example 2.8 imply that the expected cost of the augmentation step is at most $2\,\sigma_{j+1}\,\mathbf{E}\,[c(T_j^*)]$. In more detail, we abuse notation and write $\chi(\widehat{D}_j, d_i)$ for the Prim cost share of a demand pair (d_i, t) in the Steiner Tree instance $(G, \mathcal{D})$, where $\mathcal{D} = \{(d_i, t) : d_i \in \widehat{D}_j\}$. As in the proof of Lemma 2.9, we define two random variables B_i and R_i for each demand $d_i \in D_j$. The random variable B_i is equal to σ_{j+1} times the Prim cost share $\chi(\widehat{D}_j, d_i)$ when $d_i \in \widehat{D}_j$, and to 0 otherwise. By Definition 2.4, $\sum_{d_i \in D_j} B_i$ is at most $\sigma_{j+1}\,c(T_j^*)$ (with probability 1). The variable R_i is defined to be zero when $d_i \in \widehat{D}_j$, and is equal to σ_j times the length $\ell(d_i, F_j)$ of a shortest path between d_i and a vertex of F_j. Since the probability that d_i lies in $\widehat{D}_j$ is $p_j = \sigma_j/\sigma_{j+1}$, following the proof of Lemma 2.9 shows that the expected cost $\mathbf{E}\,[\sum_i R_i]$ of the augmentation step is at most $2 \cdot \mathbf{E}\,[\sum_i B_i]$, and hence is at most $2\sigma_{j+1} \cdot \mathbf{E}\,[c(T_j^*)]$.

We complete the proof by showing that the expected cost of the gathering and rounding steps is at most $\sigma_{j+1}\,\mathbf{E}\,[c(T_j^*)]$. Intuitively, we will charge the expected cost of these steps to that of the earlier augmentation step. Lemma 5.1 ensures that the rerouting of residual demands in the rounding step can be accomplished using the cables of type $j+1$ purchased in the subproblem step, and no new cables need to be built. For every edge e of G, one cable of type j was installed on e in the augmentation step of the jth stage for each demand of D_j that used e to travel to a vertex of F_j. In the gathering and rounding steps, one cable of type $(j+1)$ is installed on e for each such demand that is chosen for membership in the set D_{j+1}. Recall from the proof of Lemma 5.2 that for every demand $d \in D_j$, the probability that d is included in D_{j+1} is precisely u_j/u_{j+1}. The expected cost of rerouting demands in the gathering and rounding steps is therefore at most

$$\frac{u_j}{u_{j+1}} \cdot \frac{\sigma_{j+1}}{\sigma_j} = \frac{\delta_{j+1}}{\delta_j}$$

times the expected cost of the augmentation step. Since $\delta_{j+1} \leq \delta_j/2$, the expected cost of the gathering and routing steps is at most $\sigma_{j+1}\,\mathbf{E}\,[c(T_j^*)]$. The lemma is proved. ■

Putting together our bounds on the expected costs incurred in the preprocessing steps and in all of the stages of the SSBaB-Sample-Augment algorithm implies that it is a constant-factor approximation algorithm for the SSBaB problem.

Theorem 5.5 *Algorithm* SSBaB-Sample-Augment *is a 76.8-approximation algorithm for the* SSBaB *problem.*

Proof: Fix an optimal solution with cost $OPT = \sum_j C^*(j)$. By Lemmas 5.3 and 5.4, the expected cost incurred by the algorithm in stages 1 through $K-1$ is at most

$$\sum_{j=1}^{K-1}(3+\alpha)\sigma_{j+1} \cdot \mathbf{E}\left[c(T_j^*)\right] \quad = \quad (3+\alpha)\sum_{i=1}^{K} C^*(i) \cdot \left[\sum_{j=1}^{i-1}\frac{\sigma_{j+1}}{\sigma_i} + \sum_{j=i}^{K}\frac{\sigma_{j+1}}{\delta_i g_j}\right].$$

Recalling that $\sigma_{j+1}/g_j = \delta_j$ for each j and adding in the cost (38) of the preprocessing step that produces unit-weight demands for stage 1, we get that the total expected cost incurred by the SSBaB-Sample-Augment algorithm after the initial rounding of cable costs and capacities and before stage K is at most

$$(3+\alpha)\sum_{i=1}^{K} C^*(i) \cdot \left[\sum_{j=0}^{i-1}\frac{\sigma_{j+1}}{\sigma_i} + \sum_{j=i}^{K}\frac{\delta_j}{\delta_i}\right].$$

Since $\sigma_{j+1} \geq 2\sigma_j$ and $\delta_{j+1} \leq \delta_j/2$ for every $j \in \{1, 2, \ldots, K-1\}$, this cost is at most $4(3+\alpha) \cdot OPT$.

In the final stage K of the algorithm, we route demands of size u_K to the sink t along shortest paths, building cables of type K to support this flow. This costs

$$\sum_{d \in D_K} \sigma_K \cdot \ell(d,t),$$

where $\ell(d,t)$ denotes the length of a shortest d-t path in G. Since every demand $d \in D_0$ with weight $w_d \leq 1$ corresponds to a demand of D_K in the final stage with probability w_d/u_K (Lemma 5.2), the expected cost of these cables of type K is

$$\sum_{d \in D_0} \frac{w_d}{u_K} \cdot \sigma_K \cdot \ell(d,t) = \delta_K \sum_{d \in D_0} w_d \ell(d,t). \tag{40}$$

Since δ_K is the smallest-possible incremental cost, the right-hand side of (40) is a lower bound on the cost of the optimal solution to $\mathcal{I}$. Thus the expected cost in the Kth stage of the SSBaB-Sample-Augment algorithm is at most OPT.

Finally, our initial rounding of the cable costs and capacities increases our approximation ratio by a factor of 4. The final approximation ratio of the SSBaB-Sample-Augment algorithm is thus $4\left[4(3+\alpha)+1\right]$. Using the Steiner Tree algorithm of Robins and Zelikovsky [58], we can take $\alpha = 1.55$ to achieve an approximation ratio of 76.8. ■

6 Recent and Future Work

We conclude by discussing recent research motivated by the present paper and some directions for future work.

6.1 Recent Work

The initial publication of our MRoB algorithm [34] led to two subsequent papers on the problem. As discussed in Remark 3.44, Becchetti et al. [12] designed an alternative way to force the AKR-GW algorithm to build additional edges, and used it and Theorem 2.10 to give a 6.83-approximation algorithm for MRoB. Very recently, Fleischer et al. [25] designed a (non-straightforward) 3-strict cost sharing method for the AKR-GW algorithm. In conjunction with the Sample-Augment algorithm and Theorem 2.10, this gives a 5-approximation algorithm for MRoB, the best that is currently known. No improvements to our SSRoB algorithm have yet appeared in the literature, although Gupta, Srinivasan, and Tardos [39] recently derandomized the algorithm. Their approach is based on an alternative analysis of the algorithm and results in a deterministic 4.2-approximation algorithm, slightly better than the deterministic 4.55-approximation algorithm of Swamy and Kumar [60].

For buy-at-bulk problems, no improvement of our SSBaB algorithm is known. On the other hand, Charikar and Karagiozova [15] recently gave the first non-trivial approximation algorithm for the generalization of the multicommodity buy-at-bulk network design problem in which the concave capacity cost function (or, equivalently, the available cable types) can vary from edge to edge. The algorithm in [15], inspired by the Sample-Augment algorithm of this paper, randomly inflates the weight of demand pairs and then runs a greedy heuristic.

Two improvements of our VPND algorithm and analysis have recently been given. The first is a 4.74-approximation algorithm due to Eisenbrand and Grandoni [21], the second a 3.55-approximation algorithm of Eisenbrand et al. [22]. Both papers are based on variations of our algorithm and refinements of our analysis.

Most significantly, our definition of strict cost shares has been generalized and applied to give the first constant-factor approximation algorithms for several problems in *stochastic optimization*. As an example, consider the following Stochastic Steiner Tree problem. The input is a graph G with edges costs c, a sink vertex t, a set $S = \{s_1, \ldots, s_k\}$ of sources, a distribution π over sets of sources and an "inflation factor" $\sigma > 1$. The setup is as follows: an algorithm chooses a set F_1 of edges in the first stage; a set $\widehat{S} \subseteq S$ of sources is chosen randomly according to π; and then the algorithm chooses a set F_2 of edges so that $F_1 \cup F_2$ spans t and the sources of $\widehat{S}$. The incentive for selecting edges in the first stage (without knowledge of the realization $\widehat{S}$) is that each edge e costs c_e in the first stage but σc_e in the second stage. The goal is to design an algorithm that chooses F_1 and F_2 in a way that approximately minimizes the expectation (over π and F_2) of the total cost $c(F_1) + \sigma c(F_2)$.

Gupta et al. [37] showed that random sampling, a Steiner Forest subroutine that admits a strengthened form of strict cost shares, and greedy augmentation can be used to obtain a 3.55-approximation algorithm for the Stochastic Steiner Tree problem. The only assumption on the distribution π in [37] is that independent samples of π can be drawn in polynomial time. Gupta et al. [37] also obtained similar results for stochastic versions of the Vertex Cover and Uncapacitated Facility Location problems. Earlier approximation algorithms for these problems both had weaker performance guarantees and imposed additional restrictions on the distribution π [42, 57]. Strict cost shares and generalizations have since been used to design constant-factor approximation algorithms for many other stochastic optimization problem [36, 38, 41].

6.2 Future Directions

We conclude the paper with several suggestions for future research.

1. An obvious open question is to narrow the gap between the best approximation and inapproximability results for all of the problems studied in this paper. In particular, are any of these problems provably harder than the Steiner Tree problem (assuming $P \neq NP$)?

2. A more modest goal is to understand the limitations of our analysis framework in Section 2. For example, is the guarantee in Theorem 2.10 the best possible? Is it possible to refine the definition of strict cost shares and sharpen this guarantee?

3. Can the ideas in our MRoB and SSBaB algorithms be combined to yield an approximation algorithm for the multicommodity buy-at-bulk problem? While recent results of Andrews [2] rule out constant-factor approximation algorithms under reasonable complexity assumptions, our techniques might give an $O(\log n)$-approximation algorithm for the problem that does not resort to probabilistic tree embeddings [9, 23].

4. Can the constant-factor approximation algorithm for Stochastic Steiner Tree in [37] be extended to the stochastic version of the Steiner Forest problem? Such an extension would follow from a strengthened version of our strict cost shares in Subsection 3.3.

5. Only our SSRoB algorithm has been derandomized [39]. Can our other algorithms also be derandomized?

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. (Preliminary version in *23rd STOC*, 1991).

[2] M. Andrews. Hardness of buy-at-bulk network design. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 115–124, 2004.

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

[4] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, 1998.

[5] B. Awerbuch and Y. Azar. Buy-at-bulk network design. In *Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 542–547, 1997.

[6] B. Awerbuch, Y. Azar, and Y. Bartal. On-line generalized Steiner problem. In *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 68–74, 1996.

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

[8] Y. Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 184–193, 1996.

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

[10] Y. Bartal, M. Charikar, and P. 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.

[11] Y. Bartal, A. Fiat, and Y. Rabani. Competitive algorithms for distributed data management. *Journal of Computer and System Sciences*, 51(3):341–358, 1995. (Preliminary version in *24th STOC*, 1992).

[12] L. Becchetti, J. Könemann, S. Leonardi, and M. Pál. Sharing the cost more efficiently: Improved approximation for multicommodity rent-or-buy. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 375–384, 2005.

[13] M. Bern and P. Plassman. The Steiner problem with edge lengths 1 and 2. *Information Processing Letters*, 32(4):171–176, 1989.

[14] M. Charikar and S. 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 (FOCS)*, pages 378–388, 1999.

[15] M. Charikar and A. Karagiozova. On non-uniform multicommodity buy-at-bulk network design. In *Proceedings of the 37th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 176–182, 2005.

[16] C. Chekuri, S. Khanna, and J. Naor. A deterministic algorithm for the Cost-Distance problem. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 232–233, 2001.

[17] J. Chuzhoy, A. Gupta, J. Naor, and A. Sinha. On the approximability of some network design problems. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 943–951, 2005.

[18] V. Chvátal. *Linear Programming.* Freeman, 1983.

[19] R. Cole, R. Hariharan, M. Lewenstein, and E. Porat. A faster implementation of the Goemans-Williamson clustering algorithm. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 17–25, 2001.

[20] N. G. Duffield, P. Goyal, A. G. Greenberg, P. P. Mishra, K. K. Ramakrishnan, and J. E. van der Merwe. A flexible model for resource management in virtual private networks. In *Proceedings of SIGCOMM*, pages 95–108, 1999.

[21] F. Eisenbrand and F. Grandoni. An improved approximation algorithm for virtual private network design. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 928–932, 2005.

[22] F. Eisenbrand, G. Grandoni, G. Oriolo, and M. Skutella. New approaches for virtual private network design. In *Proceedings of the 32nd Annual International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 3580 of *Lecture Notes in Computer Science*, pages 1151–1162, 2005.

[23] J. Fakcharoenphol, S. Rao, and K. Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC)*, 2003.

[24] J. A. Fingerhut, S. Suri, and J. S. Turner. Designing least-cost nonblocking broadband networks. *Journal of Algorithms*, 24(2):287–309, 1997.

[25] L. K. Fleischer, J. Könemann, S. Leonardi, and G. Schäfer. A tight 4.67-approximation algorithm for the multi-commodity rent-or-buy problem. Technical Report 09-05, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 2005.

[26] H. N. Gabow, M. X. Goemans, and D. P. Williamson. An efficient approximation algorithm for the survivable network design problem. *Mathematical Programming*, 82(1-2):13–40, 1998.

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

[28] A. Goel and D. 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 (SODA)*, pages 499–505, 2003.

[29] M. X. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24(2):296–317, 1995. (Preliminary version in *5th SODA*, 1994).

[30] 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*. PWS Publishing, 1997.

[31] S. Guha, A. Meyerson, and K. Munagala. Hierarchical placement and network design problems. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 603–612, 2000.

[32] S. Guha, A. Meyerson, and K. Munagala. 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.

[33] A. Gupta, A. Kumar, J. Kleinberg, R. Rastogi, and B. 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 (STOC)*, pages 389–398, 2001.

[34] A. Gupta, A. Kumar, M. Pál, and T. Roughgarden. Approximation via cost-sharing: A simple approximation algorithm for the multicommodity rent-or-buy problem. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 606–615, 2003.

[35] A. Gupta, A. Kumar, and T. Roughgarden. Simpler and better approximation algorithms for network design. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC)*, 2003.

[36] A. Gupta and M. Pál. Stochastic Steiner trees without a root. In *Proceedings of the 32nd Annual International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 3580 of *Lecture Notes in Computer Science*, pages 1051–1063, 2005.

[37] A. Gupta, M. Pál, R. Ravi, and A. Sinha. Boosted sampling: Approximation algorithms for stochastic optimization. In *Proceedings of the 36th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 417–426, 2004.

[38] A. Gupta, M. Pál, R. Ravi, and A. Sinha. What about Wednesday? Approximation algorithms for multistage stochastic optimization. In *Proceedings of the 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, volume 3624 of *Lecture Notes in Computer Science*, pages 86–98, 2005.

[39] A. Gupta, A. Srinivasan, and É. Tardos. Cost-sharing mechanisms for network design. In *Proceedings of the 7th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, volume 3122 of *Lecture Notes in Computer Science*, pages 139–150, 2004.

[40] R. Hassin, R. Ravi, and F. S. Salman. Approximation algorithms for a capacitated network design problem. In *Proceedings of the 3rd International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, volume 1913 of *Lecture Notes in Computer Science*, pages 167–176, 2000.

[41] A. Hayrapetyan, C. Swamy, and É. Tardos. Network design for information networks. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 933–942, 2005.

[42] N. Immorlica, D. R. Karger, M. Minkoff, and V. S. Mirrokni. On the costs and benefits of procrastination: Approximation algorithms for stochastic combinatorial optimization problems. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 691–700, 2004.

[43] K. Jain and V. 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.

[44] K. Jain and V. Vazirani. Equitable cost allocations via primal-dual-type algorithms. In *Proceedings of the 34th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 313–321, 2002.

[45] D. R. Karger and M. Minkoff. Building Steiner trees with incomplete global knowledge. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 613–623, 2000.

[46] S. Khuller and A. Zhu. The general Steiner tree-star problem. *Information Processing Letters*, 84(4):215–220, 2002.

[47] 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.

[48] P. N. Klein. A data structure for bicategories, with application to speeding up an approximation algorithm. *Information Processing Letters*, 52(6):303–307, 1994.

[49] J. Könemann, S. Leonardi, and G. Schäfer. A group-strategyproof mechanism for Steiner forests. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 612–619, 2005.

[50] A. Kumar, A. Gupta, and T. 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 (FOCS)*, pages 333–342, 2002.

[51] M. Labbé, G. Laporte, I. M. Martín, 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.

[52] Y. Lee, 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.

[53] A. Meyerson, K. Munagala, and S. Plotkin. Cost-Distance: Two metric network design. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 624–630, 2000.

[54] A. Meyerson, K. Munagala, and S. Plotkin. Designing networks incrementally. In *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 406–415, 2001.

[55] M. Pál and É. Tardos. Group strategyproof mechanisms via primal-dual algorithms. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 584–593, 2003.

[56] R. Ravi and F. S. Salman. Approximation algorithms for the traveling purchaser problem and its variants in network design. In *Proceedings of the 7th Annual European Symposium on Algorithms (ESA)*, volume 1643 of *Lecture Notes in Computer Science*, pages 29–40, 1999.

[57] R. Ravi and A. Sinha. Hedging uncertainty: Approximation algorithms for stochastic optimization problems. Technical Report Working paper 2003-E68, CMU GSIA, 2003.

[58] G. Robins and A. Zelikovsky. Improved Steiner tree approximation in graphs. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 770–779, 2000.

[59] 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. (Preliminary version in *8th SODA*, 1997).

[60] C. Swamy and A. 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.

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

[62] V. V. Vazirani. *Approximation Algorithms*. Springer-Verlag, Berlin, 2001.

[63] 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 Proof of Lemma 3.19

Proof of Lemma 3.19: Fix a Steiner Forest instance $(G, \mathcal{D})$. A time τ is *interesting* if $\tau = 0$, if the γ-AKR-GW algorithm merges two clusters at time τ, or if a cluster becomes deactivated at time τ. Call the time in between consecutive interesting moments an *epoch*. Let C_i denote the set of active clusters during the ith epoch. In the ith epoch, the dual variables of all of the clusters in C_i are raised by some common amount, which we denote by Δ_i. From the definitions, we have

$$z_S = \sum_{i\,:\,S \in C_i} \Delta_i \tag{41}$$

for every possible cluster $S \subseteq V$ and

$$\sum_{S \subseteq V} z_S = \sum_{i=1}^{p} \Delta_i |C_i|, \tag{42}$$

where p is the number of epochs.

Let F be the Steiner forest output by the algorithm. The key claim is the following: in every epoch i,

$$\sum_{S \in C_i} |F \cap \delta(S)| \leq 2|C_i|. \tag{43}$$

In other words, at every moment in time, an average active cluster only intersects two edges of the final output F.

To prove (43), fix an epoch i and obtain the graph H from the graph (V, F) by contracting each cluster (active or inactive) of epoch i into a single vertex. Thus the vertices of H correspond to the clusters in the ith epoch, and the edges of H are the edges of F that span two of these clusters. We will call the vertices of H *active* or *inactive* according to the status of the corresponding cluster of G in the ith epoch. The inequality (43) is equivalent to the assertion that the average degree of the active vertices of H is at most 2.

First, since the edges of F are tight edges, and the γ-AKR-GW algorithm maintains the invariant that clusters correspond to connected components of the set of tight edges, Lemma 3.17 implies that the graph H is acyclic. Second, we claim that no inactive vertex of H has degree 1. This claim follows from the delete step of the γ-AKR-GW algorithm. To see this, consider a cluster S that is inactive during the ith epoch. By the definition of the γ-AKR-GW algorithm, all demands in S must be inactive at this and all future moments in time. Since the algorithm only merges classes of the demand partition that contain currently active vertices, in the final partition $\mathcal{P}$, no partition class will contain both a demand from S and a demand from outside S. If the vertex of H corresponding to this cluster has degree 1, then there is an edge e of F whose removal can only disrupt the connectivity of demand pairs with one demand in S and the other outside S. Thus edge e is not essential for $\mathcal{P}$-connectivity, and should have been removed in the delete step of the γ-AKR-GW algorithm.

These two claims easily imply (43). Obtain $\widetilde{H}$ from H by discarding all the isolated inactive vertices. Since $\widetilde{H}$ is acyclic, the average degree of vertices of $\widetilde{H}$ is at most 2.

Moreover, inactive vertices of $\widetilde{H}$ all have degree at least 2. The active vertices of $\widetilde{H}$ (and H) thus have average degree at most 2.

Having established (43), we can now bound the cost of F as follows:

$$\begin{aligned}
\sum_{e \in F} c_e &= \sum_{e \in F} \sum_{S \subseteq V \,:\, e \in \delta(S)} z_S && (44)\\
&= \sum_{S \subseteq V} z_S \cdot |F \cap \delta(S)| \\
&= \sum_{i=1}^{p} \Delta_i \sum_{S \in C_i} |F \cap \delta(S)| && (45)\\
&\leq \sum_{i=1}^{p} \Delta_i \cdot 2|C_i| && (46)\\
&= 2 \sum_{S \subseteq V} z_S, && (47)
\end{aligned}$$

where (44) follows from the fact that all edges of F are tight, equation (45) follows from (41), inequality (46) follows from (43), and equation (47) follows from (42). ■