Stochastic Steiner Trees without a Root

Anupam Gupta[1] and Martin Pál[2]

[1] Dept. of Computer Science, Carnegie Mellon University, Pittsburgh PA 15213.
anupamg@cs.cmu.edu
[2] DIMACS center, Rutgers University, Piscataway, NJ. Supported by ONR grant N00014-98-1-0589 (while at Cornell University) and NSF grant EIA 02-05116 (at DIMACS).
mpal@acm.org

Abstract. This paper considers the Steiner tree problem in the model of *two-stage stochastic optimization with recourse*. This model, the focus of much recent research [1–4], tries to capture the fact that many infrastructure planning problems have to be solved in the presence of uncertainty, and that we have make decisions knowing merely market forecasts (and not the precise set of demands); by the time the actual demands arrive, the costs may be higher due to inflation.

In the context of the Stochastic Steiner Tree problem on a graph $G = (V, E)$, the model can be paraphrased thus: on Monday, we are given a probability distribution π on subsets of vertices, and can build some subset E_M of edges. On Tuesday, a set of terminals D materializes (drawn from the same distribution π). We now have to buy edges E_T so that the set $E_M \cup E_T$ forms a Steiner tree on D. The goal is to minimize the expected cost of the solution.

We give the first constant-factor approximation algorithm for this problem in this paper. This is, to the best of our knowledge, the first $O(1)$-approximation for the stochastic version of a *non sub-additive problem*[3] In fact, algorithms for the *unrooted* stochastic Steiner tree problem we consider in this paper are powerful enough to solve the Multicommodity Rent-or-Buy problem, themselves a topic of much recent interest [6–8].

1 Introduction

Real world planning problems often have a significant component of uncertainity. For instance, when designing networks, the precise demand patterns and future costs of building capacity are often unknown to begin with, and only become clear as time progresses. However, with our increasing ability to collect statistical data, and the development of sophisticated and realistic forecast models, the paradigm of stochastic optimization has gained much traction. Indeed, we can now aim to solve a wider class of problems: given not a single input, but a distribution over inputs, we want to find a solution that is good in expectation (taken with respect to the randomness in the model).

[3] *In a sub-additive problem, if A and B are instances, then the union of their solutions is also a feasible solution to the instance $A \cup B$; this is not true for Steiner trees. The results [1, 3, 5] only hold for sub-additive problems, and solve stochastic Steiner tree for the case when the tree must contain a specified root r.*

In this paper, we study the problem of connecting a group of terminals by a Steiner tree in a stochastic setting. In the classical Steiner tree problem, we are given an undirected graph $G = (V, E)$ with edge costs c_e, and a group of terminals $g = \{t_1, t_2, \ldots, t_k\}$; the goal is to find a subset E' of edges of minimum cost that connects all these terminals. We consider this problem when the group g is not deterministically given in advance; instead, it is given by a random variable Γ, with $\Pr[\Gamma = g]$ being the probability that we will be required to build a network that connects a particular group $g \subseteq V$ of terminals. As sketched in the abstract, we work the model of two-stage stochastic optimization with recourse.

- In the first stage, we assume to have (some) knowledge of the distribution of the random variable Γ. Armed with this information, we construct a network $F^0 \subseteq E$ of edges bought as the first *anticipatory* part of the solution.
- In the second stage, we learn a group $g \subseteq V$ of terminals that is a realization of the random variable Γ. We have to purchase an additional *augmenting* set $F^1(g)$ of edges to ensure that $F^0 \cup F^1(g)$ connects the terminals of g. The problem is interesting when the edges bought in the second stage have a higher cost (due to inflation, or because the second phase has to be built on short notice). We use $\sigma > 1$ to denote the *inflation factor* by which the edges are more expensive.

Our goal is to minimize the expected cost of the two-stage solution. If we define $c(F) = \sum_{e \in F} c_e$, and denote the first and second stage solutions $F^0 \in E$ and $F^1 : 2^V \mapsto 2^E$ to minimize

$$c(F^0) + \mathbf{E}_\Gamma[\sigma \cdot c(F^1(\Gamma))]. \tag{1.1}$$

Our results. The main quantitative result of this paper is the following:

Theorem 1. *There is a* 12.6*-approximation algorithm for the two-stage stochastic (unrooted) Steiner tree problem.*

Note that while the stochastic Steiner tree problem has been considered in previous papers [1, 3, 5], their model is subtly but significantly different. All these works make the crucial assumption that the there is a *fixed root* r, and the goal is to connect the group g to the root r. This assumption, while a trifling detail in the deterministic case, turns out to make a big difference in the stochastic setting, requiring us to develop new techniques. For example, a fact used in one way or another by all previous results was that the first stage solution F^0 in the rooted case can be assumed to be a connected tree containing the root; this is just not true in the unrooted case: in fact, insisting on a connected first stage network may cost arbitrarily more than the optimum solution. Indeed, our result is the first approximation algorithm given for a problem that is not sub-additive, and requires us to interpret and use cost-sharing ideas in a novel way.

A note on the distributions. The distribution π of the random variable Γ is an object whose size may be exponential in $|V|$, but there are ways to cope with this fact. There may be succint representations of π: in the *independent decisions* model, each vertex $v \in V$ independently has a probability p_v of being included in Γ, which gives us a easy-to-represent product distribution. In the *scenario* model, the distribution π is given by an explicit list of pairs (g_i, p_i), with $\sum_i p_i = 1$; here p_i is the probability that the group

g_i appears. Note that the algorithm is now allowed to run in time polynomial in the length of the list. In the *sampling oracle* model, the distribution π can be arbitrary; the algorithm accesses it only through a sampling oracle. Upon request, the oracle outputs a group g that is drawn from the distribution π (or equivalently, is a realization of the random variable Γ). Our algorithm works in the most general, sampling oracle model. (We can also handle the case when the inflation parameter σ is random as well; for simplicity of exposition, we defer that discussion to the final version of the paper.)

Related work. As already mentioned, several papers studied the *rooted version* of the stochastic Steiner tree problem. Immorlica et al. [1] give a $O(\log n)$ approximation in the independent decisions model, while [3] and [5] give constant approximation algorithms for the oracle and scenario models respectively. Karger and Minkoff [9] and Hayrapetyan et al. [10] study the *maybecast* problem, where one is to output a single tree T, to minimize the expected size of the smallest subtree of T spanning a random set of terminals. While technically this is also a stochastic problem, the recourse action is fixed, and the only randomness present is in the objective function.

Gupta et al. [3] give a simple boosted sampling framework to convert an algorithm for a deterministic minimization problem to an algorithm for its stochastic counterpart. Their framework relies crucially on two ingredients: the deterministic version of the problem at hand has to be *subadditive*, and have an approximation algorithm that admits a *strict* cost sharing function. Since the unrooted Steiner tree problem is not sub-additive (i.e., if T_1 is a solution for terminal set g_1, and T_2 for g_2, then $T_1 \cup T_2$ may not be a solution for $g_1 \cup g_2$), we cannot apply their techniques directly here.

The general area of stochastic optimization is studied heavily in the operations research community, dating back to the seminal works of Dantzig [11] and Beale [12] in the 1950s; the books [13, 14] and monograph [15] could serve as introduction for the interested reader. Much of the work related to combinatorial optimization problems in this area has been concerned with finding and characterizing optimal solutions either for restricted classes of inputs or with algorithms without polynomial running times guarantees. Recently, there has been some work on taking solutions to stochastic linear programs and rounding those to obtain approximation algorithms for the stochastic problems [4]; however, it is not clear how to apply those techniques to the Steiner tree problem.

The Boosted Sampling Framework. Gupta et al. [3] propose the *Boosted Sampling* framework of Figure 1.1 to solve any two-stage stochastic problem Π where the set Γ of demand points is stochastic.

One would naturally expect that in the case of stochastic Steiner tree, the deterministic algorithm of Step 2 would build a Steiner tree on the set of terminals $g_1 \cup g_2 \cup \cdots \cup g_\sigma$. In fact, if the support of Γ was on sets that all contained the fixed root r, the analysis of [3] shows that this is enough to obtain an 3.55-approximation algorithm for stochastic Steiner tree.

Unfortunately, building a Steiner tree fails in the unrooted case. For an example, consider two groups g_1 and g_2 that are very far apart relative to their diameter; assume that $\Pr[\Gamma = g_i] \cdot \sigma$ is large. In this case, the optimum solution must connect up each group g_i in the first stage to avoid high second stage cost, but it should not build a link between g_1 and g_2 (to make F_0 span $g_1 \cup g_2$) if it wants to avoid a high first

```
1: Boosted Sampling: Take $\lfloor\sigma\rfloor$ independent samples $g_1, g_2, \ldots, g_{\lfloor\sigma\rfloor}$ from the sampling oracle
   for $\Gamma$.
2: Building First Stage Solution: Use an algorithm $\mathcal{A}$ to find a solution to the deterministic
   equivalent of the problem $\Pi$ on the groups $g_1, g_2, \ldots, g_{\lfloor\sigma\rfloor}$. Use this solution as the first
   stage solution to the stochastic problem.
3: Building Recourse: Once the group $g$ of required terminals materializes, use an augmenting
   algorithm $\mathsf{Aug}_{\mathcal{A}}$ to augment the first stage solution to a valid solution that satisfies $g$.
```

Fig. 1.1. Algorithm $\mathsf{Boost\text{-}and\text{-}Sample}(\Pi)$

stage cost. On the other hand, if the two groups are interspersed in the same region of the graph, the optimum solution may benefit from link sharing and hence build a single Steiner tree spanning both groups. Hence it seems natural to suggest that the algorithm $\mathcal{A}$ should build a forest ensuring that each group lies within a single connected component; different groups may or may not be in the same component. As it turns out, building a *Steiner Forest* on the groups g_i is a suitable deterministic equivalent of stochastic unrooted Steiner tree; however, proving this requires a lot more work.

To this end, we have to show that the main theorem of [3] which relates the performance of the boosted sampling framework to the notion of *strictness*[4] of certain cost-sharing functions can be proved in our case, even though our problem is not subadditive. The proof of this is simple, and we will sketch it in Section 2. We then define the cost-shares in Section 3, and prove them to be strict in 4.

2 Notation and preliminaries

Let $G = (V, E)$ be an undirected weighted graph with weigths c_e on the edges. A *network* is simply a subset of the edges. We say that a network F is *feasible* for (or *connects*) a group of terminals $g = \{t_1, t_2, \ldots, t_k\}$, if all the terminals of g lie in the same connected component of F. The cost of a network F is simply the sum of costs of its edges; that is $c(F) = \sum_{e \in F} c_e$.

In the *Steiner Forest* problem, given a weighted undirected graph G and a list of groups of terminals $\mathcal{D} = \{g_1, g_2, \ldots, g_n\}$ with each $g_i = \{t_{i1}, \ldots, t_{ik_i}\}$, we want to construct a network F of minimum cost that is feasible for each group g_i. For a set $\mathcal{D}$ of terminal groups, let $\mathsf{Sols}(\mathcal{D})$ denote the set of networks that are feasible for each of the groups in $\mathcal{D}$, and let $\mathsf{OPT}(\mathcal{D})$ be the network in $\mathsf{Sols}(\mathcal{D})$ of minimum cost. An algorithm $\mathcal{A}$ is an α-approximation algorithm for the Steiner Forest problem, if for any set D of terminal groups, it finds a network $F_{\mathcal{D}} \in \mathsf{Sols}(\mathcal{D})$ of cost at most $\alpha\, \mathsf{cost}(\mathsf{OPT}(\mathcal{D}))$.

Given a group g of terminals and an existing network $F \subseteq E$, the goal of an *augmenting algorithm* is buy a set of extra edges F' so that $F \cup F'$ is a network that connects the group g. For instance, given a network $F_{\mathcal{D}} \in \mathsf{Sols}(\mathcal{D})$ that connects each

[4] This concept will shortly be defined in Definition 2. Loosely, a cost sharing function is a scheme to charge the cost of a solution to the participating groups, and strictness relates the cost of the edges bought in the second stage to the group shares.

of the groups in $\mathcal{D}$, and a new group $g \notin \mathcal{D}$, the augmenting algorithm $\mathsf{Aug}_\mathcal{A}$ seeks to find a set of edges F' of minimum cost so that $F_\mathcal{D} \cup F' \in \mathsf{Sols}(\mathcal{D} \cup \{g\})$.

Definition 1. *A cost-sharing function ξ is a function that, for any instance $(G, \mathcal{D})$ of the Steiner forest problem, assigns a non-negative real number $\xi(G, \mathcal{D}, g_i)$ to every participating group $g_i \in \mathcal{D}$.*

We shall drop a reference to the graph G, if clear from the context. *Note that the cost sharing function assigns shares to groups, and not to the individual terminals.*

Since the above definition is so general, let us specify some properties of these functions that we would like to get. A cost-sharing function ξ is *competitive* if $\sum_{g \in \mathcal{D}} \xi(\mathcal{D}, g) \leq \mathsf{cost}(\mathsf{OPT}(\mathcal{D}))$ holds for any Steiner forest instance $(G, \mathcal{D})$. Thus, competitive cost-shares serve as a lower bound on the cost of the optimal solution. The following notion is crucial to the development of the paper, and implicitly places lower bounds on the cost-shares themselves.

Definition 2. *A cost sharing function ξ is β-*strict *with respect to an algorithm $\mathcal{A}$, if there exists an augmenting algorithm $\mathsf{Aug}_\mathcal{A}$, such that for any set of demand groups $\mathcal{D}$ and any group $g \notin \mathcal{D}$,*

$$\mathsf{cost}(\mathsf{Aug}_\mathcal{A}(\mathcal{A}(\mathcal{D}), g)) \leq \beta \xi(\mathcal{D} + g, g). \tag{2.2}$$

Remark 1. There is a fine distinction between the notion of strictness we use here and strictness as defined in [7, 3]. In [7], strictness was defined only for augmentations with groups of size 2; in this paper, we allow for groups of larger sizes. However, the strictness in [3] is stronger than our notion, and allows for multiple group augmentations; the question of proving strictness by this definition remains open despite much effort.

Given all these definitions, we can now state the the following theorem, which can be derived from the proof of [3, Theorem 3.1].

Theorem 2. *Suppose that $\mathcal{A}$ is an α-approximation algorithm for deterministic Steiner forest. Then, the boosted sampling algorithm of Figure 1.1 is an $(\alpha + \beta)$-approximation algorithm for unrooted stochastic Steiner tree whenever there is a cost-sharing function ξ that is β-strict with respect to $\mathcal{A}$ and single group augmentations.*

The proof of this theorem is simple, and closely follows the arguments in the aforementioned paper; we defer the simple details for the final version of the paper.

3 The Algorithm $\mathcal{A}$ and the Cost Shares ξ

In this section we review the Steiner forest algorithm of [7], although the algorithm of Becchetti et al. [6] would serve our purpose equally well. Both algorithms are extensions of the algorithm of Agarwal, Klein, and Ravi (AKR) [16], and Goemans and Williamson (GW) [17], and are designed to “build a few extra edges” over and above the AKR-GW algorithms, while keeping the overall cost of the solution within a constant factor of the cost of the optimum. We also describe our cost-sharing method.

Recall that we are given a graph $G = (V, E)$ and a set $\mathcal{D}$ of groups $g_1, \ldots, g_n$ of terminals, where each group $g_i = \{t_{i1}, t_{i2}, \ldots, t_{ik_i}\} \subseteq V$. Before defining our algorithm, we review the LP relaxation and the corresponding LP dual of the Steiner forest problem that was used in [17]:

$$\begin{aligned} &\min \textstyle\sum_e c_e x_e && \text{(SF-LP)} \\ &x(\delta(S)) \geq f(S) \quad \forall S \subseteq V && \\ &x_e \geq 0 && \end{aligned} \qquad \begin{aligned} &\max \textstyle\sum_S y_S && \text{(SF-DP)} \\ &\textstyle\sum_{S \subseteq V : e \in \delta(S)} y_S \leq c_e && (3.3) \\ &y_S \geq 0, && \end{aligned}$$

where $f(S)$ is equal to 1 if S separates g_i for some i (that is, if both $S \cap g_i$ and $(V - S) \cap g_i$ is nonempty), and is 0 otherwise. Note that variables y_S for sets S that do not separate any group are not contributing to the dual objective function, they still play an important role in our algorithm.

We now describe a general way to define primal-dual algorithms for the Steiner forest problem. As is standard for the primal-dual approach, the algorithm with maintain a feasible (fractional) dual, initially the all-zero dual, and a primal integral solution (a set of edges), initially the empty set. The algorithm will terminate with a feasible Steiner forest, which will be proved approximately optimal with the dual solution (which is a lower bound on the optimal cost by weak LP duality). The algorithms of [16, 17] arise as a particular instantiation of the following algorithm. Our presentation is closer to [16], where the "reverse delete step" of Goemans and Williamson [17] is implicit; this version of the algorithm is more suitable for our analysis.

Our algorithm has a notion of *time*, initially 0 and increasing at a uniform rate. At any point in time, some terminals will be *active* and others *inactive*. All terminals are initially active and eventually become inactive. At any point of time, the vertex set is also partitioned into *clusters*, which can again be either active or inactive. In our algorithm, a cluster will be one or more connected components (w.r.t. the currently built edges). Initially, each vertex is a cluster by itself, and the active clusters are just the terminals. We will consider different rules by which demands and clusters become active or inactive, which we describe shortly. To maintain dual feasibility, whenever the constraint (3.3) for some edge e between two clusters S and S' becomes tight (i.e., first holds with equality), the clusters are *merged* and replaced by the cluster $S \cup S'$. We raise dual variables of active clusters until there are no more such clusters.

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

We build enough edges to maintain the following invariant: the terminals in the same equivalence class are connected by built edges. This clearly holds at the beginning, since the equivalence classes are all singletons. When two active clusters meet, the invariant ensures that, in each cluster, all active terminals lie in a common connected component. To maintain the invariant, we join these two components by adding a path between them. Building such paths without incurring a large cost is simple but somewhat subtle; Agrawal et al. [16] (and implicitly, Goemans and Williamson [17]) show how to do this. We refer the reader to [16] for details of this procedure, instead of repeating it here. Specifying the rule by which clusters are deemed active or inactive now gives us two different algorithms:

1. **Algorithm** $\mathsf{GW}(G, \mathcal{D})$: A terminal $t_{ij} \in g_i$ is active if the current cluster containing it does not contain the entire group g_i. A cluster is active as long as it contains at least one active demand. This implementation of the algorithm is equivalent to the algorithms of Agrawal et al. [16] and Goemans and Williamson [17].
2. **Algorithm** $\mathsf{Timed}(G, D, T)$: This algorithm takes as an additional input a function $T : V \to \mathbb{R}_{\geq 0}$ which assigns a *stopping time* to each vertex. (We can also view T as a vector with coordinates indexed by V.) A vertex j is active at time τ if $j \in D$ *and* $\tau \leq T(j)$. (T is defined for vertices not in D for future convenience, but such values are irrelevant, and can be imagined to be set to 0 for the rest of the paper.) As before, a cluster is said to be active if at least one demand in it is active.

To get a feeling for $\mathsf{Timed}(G, D, T)$, consider the following procedure: run the algorithm $\mathsf{GW}(G, \mathcal{D})$ and set $T_{\mathcal{D}}(j)$ to be the time at which vertex j becomes inactive during this execution. (If $j \notin D$, then $T_{\mathcal{D}}(j)$ is set to zero.) Since a vertex stays active for exactly the same duration of time in the two algorithms $\mathsf{GW}(G, \mathcal{D})$ and $\mathsf{Timed}(G, D, T_{\mathcal{D}})$, the two algorithms clearly have identical outputs. Similarly, if for each $t_{ij} \in g_i$ we set $T(t_{ij}) = \max_{t,t' \in g_i} d_G(t, t')$, we we obtain the recent algorithm of Könemann et al. [18].

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

The Algorithm $\mathcal{A}$: The algorithm **Algorithm** $\mathcal{A}(G, \mathcal{D})$ that we use to build the first stage solution is

1: Run $\mathsf{GW}(G, \mathcal{D})$, and let $T_{\mathcal{D}}(v)$ be the time at which v becomes inactive.

2: Run $\mathsf{Timed}(G, D, \gamma T_{\mathcal{D}})$—the timed algorithm with the above time vector $T_{\mathcal{D}}$ scaled up by a parameter $\gamma \geq 1$—and output the resulting forest $F_{\mathcal{D}}$.

(A technical point: when $\gamma > 1$, algorithm $\mathcal{A}$ may raise the dual variables of vertex sets that do not separate any group, and hence do not contribute to the value of the dual objective function. However, this will not hinder our analysis. The fact that $F_{\mathcal{D}}$ is a feasible Steiner network for $\mathcal{D}$ is easily verified, using the fact that the terminals of each group became inactive at the same time $T_{\mathcal{D}}(g_i)$ (equal to $T_{\mathcal{D}}(t_{ij})$ for any $t_{ij} \in g_i$) when g_i became connected, and that $\gamma \geq 1$. We now define the cost shares ξ.

The Cost Shares ξ: We want the cost share of a group g_i of users to account for the growth of components that grow solely because they contain terminals from g_i. Let $a(g_i, \tau)$ be the number of active clusters in the execution of $\mathsf{GW}(G, \mathcal{D})$ that contain a terminal from g_i but *do not* contain any active terminals outside g_i. We define the cost share of g_i to be

$$\xi(\mathcal{D}, g_i) = \int a(g_i, \tau)\, \mathrm{d}\tau, \tag{3.4}$$

where the integral is over the entire execution of the algorithm. Note that the cost shares defined by Equation (3.4) do not account for the full cost of the dual solution y, as the cost of growth of clusters with active demands from more than one group more than one active demand is not reflected at all. We could fix this by dividing the cost of growing mixed clusters among participating groups in some way; however, we do not see how to use this to improve our approximation ratio.

Augmentation Algorithm $\mathsf{Aug}_{\mathcal{A}}$: A practical augmenting algorithm $\mathsf{Aug}_{\mathcal{A}}$ would simply contract all edges of $F_{\mathcal{D}}$, and then find an approximate Steiner tree on the terminals of g in this contracted graph $G/F_{\mathcal{D}}$. However, in order to bound the second stage cost, we build a specific Steiner tree on g in $G/F_{\mathcal{D}}$, and argue that the cost of *this* tree can be bounded by $\beta\,\xi(\mathcal{D}+g,g)$ for some $\beta \in \mathbb{R}$. The construction of this tree is implicit in the proof of Theorem 4, and can be found efficiently in polynomial time if required. In the following, we let $\mathsf{Aug}_{\mathcal{A}}$ be the algorithm that constructs this implicit tree. Our main technical result is thus the following.

Theorem 3. *For any $\gamma > 2$, $\mathcal{A}$ is a $\alpha = (\gamma+1)$-approximation for the Steiner network problem, and ξ is a $\beta = (4\gamma/(\gamma-2))$-strict cost sharing method with respect to the algorithms $\mathcal{A}$ and $\mathsf{Aug}_{\mathcal{A}}$.*

Proof. The fact that $\mathcal{A}$ is a $(\gamma+1)$-approximation can be proved along the lines of [6, Lemma 3.1] (We postpone the details to the full version of the paper). The proof of strictness (Theorem 4) is the analytical heart of this paper, and is given in the following section.

4 Proving strictness

Our analysis follows a fairly natural line of analysis that was also used in [7]. We start by fixing a set $\mathcal{D}$ of demand groups, and a group $g \notin \mathcal{D}$. To prove strictness of our cost shares, we compare two executions of the GW algorithm: the inflated algorithm $\mathcal{A}(G,\mathcal{D})$ on the set of groups $\mathcal{D}$ that results in the forest $F_{\mathcal{D}}$, and the uninflated algorithm $\mathsf{GW}(G,\mathcal{D}+g)$ which is responsible for computing the cost share $\xi(\mathcal{D}+g,g)$.

Recall that we have to show that g can be connected in $F_{\mathcal{D}}$ with cost at most $O(\xi(\mathcal{D}+g,g))$. We prove this in the following theorem, which also implicitly describes the augmenting algorithm $\mathsf{Aug}_{\mathcal{A}}$. In the rest of the discussion, we will assume that $\gamma > 2$.

Theorem 4. *There is a tree F' in the graph $G/F_{\mathcal{D}}$ that spans all terminals of g and has cost at most $4\gamma/(\gamma-2)\;\xi(\mathcal{D}+g,g)$. The tree F' can be constructed in polynomial time.*

The main difficulty in proving Theorem 4 arises from the fact that the two executions $\mathcal{A}(G,\mathcal{D})$ and $\mathsf{GW}(G,\mathcal{D}+g)$ may be very different. Hence it is not immediately clear how to relate the cost of augmenting the forest $F_{\mathcal{D}}$ produced by the former by the cost share $\xi(\mathcal{D}+g,g)$ computed by the latter. To make a direct comparison possible, we work through some transformations that allow us to find a mapping between dual variables in these two executions. In the grand finale, we produce a tree $\mathbb{T}$ that spans terminals of g, and show that a $1/\beta$ fraction of its edges is covered by dual variables corresponding to the cost share of g, which will complete the proof. Let us introduce some time vectors to facilitate this comparison.

- Let $T_{\mathcal{D}}$ be the time vector obtained by running $\mathsf{GW}(G, \mathcal{D})$. Recall that $F_{\mathcal{D}}$ is the forest constructed by $\mathsf{Timed}(G, \mathcal{D}, \gamma T_{\mathcal{D}})$; we also let $\mathcal{R}_{\mathcal{D}}$ be the equivalence relation constructed by the latter algorithm.
- Let $T_{\mathcal{D}+g}$ be the time vector generated by the execution $\mathsf{GW}(G, \mathcal{D} + g)$ and let $\tau = T_{\mathcal{D}+g}(g)$ be the time when the terminals of g got connected in this execution.
- Let T be the vector obtained by truncating $T_{\mathcal{D}+g}$ at time τ. That is, $T(v) = \min(\tau, T_{\mathcal{D}+g}(v))$ for $v \in V$. (The intuition for T is loosely this: we do not care about time after g has been connected, and this truncation captures this fact.)
- Finally, let T_{-g} be the vector T with g "taken out", that is, $T_{-g}(v) = T(v)$ if $v \notin g$, and $T_{-g}(v) = 0$ if $v \in g$. Let $\mathcal{R}_{-g}$ be the equivalence relation constructed by the execution $\mathsf{Timed}(G, \mathcal{D}, \gamma T_{-g})$.

A side-by-side comparison of the executions $\mathsf{GW}(G, \mathcal{D})$ and $\mathsf{GW}(G, \mathcal{D} + g)$ shows that for all $v \in V$,

$$T_{\mathcal{D}}(v) \geq T_{-g}(v); \tag{4.5}$$

the simple inductive proof is omitted. Hence, we will use the forest constructed by $\mathsf{Timed}(G, \mathcal{D}, \gamma T_{-g})$ as a proxy for the forest $F_{\mathcal{D}}$ created by $\mathsf{Timed}(G, \mathcal{D}, \gamma T_{\mathcal{D}})$; intuitively, since T_{-g} is smaller than $T_{\mathcal{D}}$, it should also produce a forest with fewer edges. We will make this intuition precise in Lemma 1 below.

To state the lemma in a general form that will be useful later, we need some more notation. For two weighted graphs G and G' on the same vertex set V, we write $G' \leq G$ if the shortest path distance between any pair of vertices (u,v) in G' is no more than their distance in G. For a graph $G = (V, E)$ and a set $F \subseteq (V \times V)$, the graph $G' = G/F$ is a *contraction* of G, and is obtained by adding a zero-cost edge in G between every pair $(u, v) \in F$. Since $\mathcal{R} \subseteq V \times V$, we can define $G/\mathcal{R}$ in the same way. It immediately follows that if G' is a contraction of G, then $G' \leq G$. For time vectors, let $T \leq T'$ denote coordinate-wise inequality (and hence we can rewrite (4.5) as $T_{-g} \leq T_{\mathcal{D}}$).

Lemma 1 ([7]). *Let $G \leq G'$ be two weighted graphs and $T \leq T'$ be two time vectors. Then, for the equivalence relations $\mathcal{R}$ and $\mathcal{R}'$ produced by the executions* $\mathsf{Timed}(G, \mathcal{D}, T)$ *and* $\mathsf{Timed}(G', \mathcal{D}, T')$*, it holds that $\mathcal{R} \subseteq \mathcal{R}'$.*

A Simpler graph H: We now define a simpler graph $H = G/\mathcal{R}_{-g}$; this graph H will act as a proxy for $G/F_{\mathcal{D}}$ in the following sense. For two vertices u, v connected by a zero-cost path in H, we know that u and v are connected by a path in $F_{\mathcal{D}}$. This is because the inequality $T_{-g} \leq T_{\mathcal{D}}$ used with Lemma 1 implies that $\mathcal{R}_{-g} \subseteq \mathcal{R}_{\mathcal{D}}$; now the invariant maintained by the algorithm Timed implies that there is a path connecting u and v in $F_{\mathcal{D}}$ whenever $(u, v) \in \mathcal{R}_{\mathcal{D}}$.

Thus, to prove Theorem 4, it suffices to exhibit a tree $\mathbb{T}$ in H that spans all terminals of g, and has cost at most $4\gamma/(\gamma - 2)\xi(\mathcal{D} + g, g)$. By the properties of the graph H, it then follows that the network $\mathbb{T} \cup F_{\mathcal{D}}$ is feasible for the group g.

Note that each equivalence class of $\mathcal{R}_{-g}$ can also be thought of as a single (super)-vertex of the graph H; this view may be more useful in some contexts. To complete the correspondence between the two views, let us extend the definition of a time vector to supernodes in the natural way: if w_C is an equivalence class of the relation $\mathcal{R}_{-g}$, we let $T(w_c) = \max_{v_i \in C} T(v_i)$; this allows us to talk about running the Timed algorithm on H with the vector T.

4.1 The tree $\mathbb{T}$ spanning terminals of g

We will obtain the desired Steiner tree on the group g in H by considering the execution of the algorithm $\mathsf{Timed}(H, \mathcal{D} + g, T)$; we denote this execution by $\mathcal{E}$. Recall that the time vector T was defined to ensure that in the execution $\mathsf{Timed}(G, \mathcal{D} + g, T)$ on the original graph G, the terminals of g eventually merge into a single equivalence class of the respective relation $\mathcal{R}$. Since the graph H is a contraction of G, it follows from Lemma 1 that the terminals of g must end up in the same equivalence class in $\mathcal{E}$, and hence in the same connected component of the forest constructed by $\mathcal{E}$. There is a unique minimal tree that spans the terminals of g in this forest; let $\mathbb{T}$ denote this tree.

Since $\mathbb{T}$ was constructed by the execution $\mathcal{E}$, all of its edges must be fully tight with the dual grown in $\mathcal{E}$. Our plan of attack is to show that the dual variables corresponding to the terminals of g account for a significant fraction of this dual, and hence the cost share of g must be large enough to pay for a $1/\beta$ fraction of the tree. To pursue this plan, we introduce the following notion of layers as in [7]; this terminology is just a convenient way of talking about "dual moats".

In an execution of an algorithm, a *layer* (C, I) corresponds to an active cluster C whose dual variable y_C has been growing during the time interval $I = [\tau_1, \tau_2)$; the *thickness* of this layer is $|I| = \tau_2 - \tau_1$. A *layering* $\mathcal{L}$ of an execution is a set of layers such that, for every time τ and every active cluster C, there is exactly one layer $(C, I) \in \mathcal{L}$ such that $\tau \in I$.

Lonely layers: A layer (C, I) is *lonely*, if it does not contain any active terminals except terminals belonging to g. Thus, the cost share of g can be expressed as the total thickness of lonely layers in any layering of $\mathsf{Timed}(G, \mathcal{D} + g, T)$. Using Lemma 1, we can argue that the total thickness of lonely layers in the execution $\mathcal{E}$ is no more than in $\mathsf{Timed}(G, \mathcal{D} + g, T)$ (see [7] for details). Hence the total thickness of lonely layers in the execution $\mathcal{E}$ is a lower bound on the cost share of g.

We lower bound the thickness of lonely layers by arguing that the thickness of non-lonely layers intersecting $\mathbb{T}$ is significantly smaller than the length of $\mathbb{T}$: since all of $\mathbb{T}$ has to be covered, this leaves a considerable fraction of the tree to be covered by lonely layers. Hence our overall goal can be reduced to giving an upper bound on the thickness of non-lonely layers that intersect the tree $\mathbb{T}$.

To get a hold on this quantity, we proceed to compare a layering $\mathcal{L}$ of the execution $\mathcal{E}$—recall that $\mathcal{E} = \mathsf{Timed}(H, \mathcal{D} + g, T)$—with a layering $\mathcal{L}'$ of its inflated counterpart $\mathcal{E}' = \mathsf{Timed}(H, \mathcal{D}, \gamma T_{-g})$. We construct a mapping that maps every non-lonely layer $\ell = (C, I) \in \mathcal{L}$ to a distinct layer $\ell' = (C', \gamma I) \in L'$ that is γ times thicker. (Note that lonely layers do not have a natural counterpart, as the terminals of g do not appear at all in the execution $\mathcal{E}'$.) To ensure the existence of such a mapping, we align the two layerings to satisfy the following property: if $(C, I) \in \mathcal{L}$ and $(C', I') \in \mathcal{L}'$ with $\gamma I \cap I' \neq \emptyset$, then $I' = \gamma I$. (I.e., $I' = [\gamma\tau_1, \gamma\tau_2)$ and $I = [\tau_1, \tau_2)$.) This condition can easily be imposed by repeatedly *splitting* layers of $\mathcal{L}$ and $\mathcal{L}'$, that is, replacing an offending layer $(C, [\tau_1, \tau_2))$ by two layers $(C, [\tau_1, \hat{\tau}))$ and $(C, [\hat{\tau}, \tau_2))$ for a suitably chosen $\hat{\tau} \in [\tau_1, \tau_2)$.

Mapping non-lonely layers of $\mathcal{L}$ to layers of $\mathcal{L}'$: Every non-lonely layer $\ell = (C, [\tau_1, \tau_2))$ must contain a terminal $t \in C$ such that $t \notin g$, that was active in the interval $[\tau_1, \tau_2)$. Since $T_{-g} \leq T_{\mathcal{D}}$, the terminal t must have been active in the interval $[\gamma\tau_1, \gamma\tau_2)$ in the execution $\mathcal{E}'$, and hence there is a unique layer $\ell' = (C', [\gamma\tau_1, \gamma\tau_2))$ such that $t \in C'$. We thus map ℓ to ℓ'. A layer ℓ may contain multiple active terminals outside g; in that case, pick one of them arbitrarily.

The following two lemmas supply us with all the ammunition we will need to finish our argument. In the next lemma, let $V(\mathbb{T})$ denote the vertex set of the tree $\mathbb{T}$.

Lemma 2. *The mapping from non-lonely layers of $\mathcal{L}$ to layers of $\mathcal{L}'$ is one to one; that is, distinct layers of $\mathcal{L}$ map to distinct layers of $\mathcal{L}'$.*

Lemma 3. *Let $\ell = (C, I) \in \mathcal{L}$ be a non-shared layer, such that $V(\mathbb{T}) \cap C \neq \emptyset$. Then, for its corrsponding layer $\ell' = (C', \gamma I)$ we have that $V(\mathbb{T}) \cap C \neq \emptyset$.*

The proof of the former is in Appendix A; the latter follows from [7, Lemmas 4. 16 and 4.17].

4.2 The book keeping

Let L and N denote the total thickness of lonely and non-lonely layers that intersect the tree $\mathbb{T}$. Note that we count every layer only once, irrespective of how many edges of $\mathbb{T}$ it cuts. We can express the total length of the tree as

$$|\mathbb{T}| = L + N + X, \tag{4.6}$$

where X represents the "extra" contributions of layers that intersect $\mathbb{T}$ more than once. (For example, if a lonely layer intersects $\mathbb{T}$ in three edges, it is counted once in L and twice in X).

At any time instant τ, consider all the active clusters in the execution $\mathcal{E}$ that have a non-empty intersection with the tree $\mathbb{T}$. We claim that any such cluster C "carves out" a connected portion of the tree $\mathbb{T}$, that is, $C \cap \mathbb{T}$ is a connected graph. Hence if we construct a graph with a node for every cluster intersecting $\mathbb{T}$ and an edge between every pair of clusters connected by a direct path along $\mathbb{T}$, this graph will also be a tree. The number of layers intersecting $\mathbb{T}$ is equal to the number of nodes in this graph; the number of times each layer intersects $\mathbb{T}$ is equal to the degree of the corresponding vertex in this graph. Since the average vertex degree in a tree is at most 2, the number of intersections is at any time bounded by twice the number of layers intersecting $\mathbb{T}$. Integrating over the course of the execution $\mathcal{E}$, we obtain that

$$L + N + X \leq 2(L + N). \tag{4.7}$$

A non-lonely layer ℓ is considered *wasted* if ℓ intersects $\mathbb{T}$, but its image ℓ' does not. According to Lemma 3, this happens only if $\mathbb{T}$ is fully contained inside ℓ'. Let W denote the total thickness of wasted layers. The total thickness of layers of $\mathcal{L}'$ intersecting $\mathbb{T}$ is a lower bound on the length of $\mathbb{T}$. Since the image of every non-lonely layer ℓ that intersects $\mathbb{T}$ and is not wasted also intersects $\mathbb{T}$, and because images of distinct layers do not overlap, we get the following lower bound on the length of $\mathbb{T}$.

$$\gamma(N - W) \leq |\mathbb{T}|. \tag{4.8}$$

The final piece of our argument is the following claim: for every layer that is wasted, there must be a lonely layer growing at the same time, and hence $W \leq L$. To see this claim, suppose that a non-lonely layer $\ell = (C, I)$ intersects $\mathbb{T}$ but is wasted—hence for its inflated image $\ell' = (C', \gamma I)$, we have $V(\mathbb{T}) \subseteq C'$. Since ℓ intersects $\mathbb{T}$, there must be a terminal $t \in g$ such that $t \notin C$. We now claim that during the interval I, the terminal t must have been a part of a lonely cluster. Indeed, suppose not; let t be inside a non-lonely layer $\ell_1 = (C_1, I)$ with some other active terminal $t_1 \notin g$. But then, by Lemma 3, the inflated image $\ell_1' = (C_1', \gamma I)$ of this layer ℓ_1 must contain some vertex of $\mathbb{T}$, and since $V(\mathbb{T}) \subseteq C'$, the layers ℓ' and ℓ_1' have a nonempty intersection. This is possible only if ℓ' and ℓ_1' are the same inflated layer, which contradicts Lemma 2, as the clearly distinct layers ℓ and ℓ_1 would then map to the same layer $\ell' = \ell_1'$. Thus,

$$W \leq L. \tag{4.9}$$

Combining the inequalities (4.6–4.9), we obtain $(\gamma - 2)|\mathbb{T}| \leq 4\gamma L$, thus proving Theorem 4.

References

1. Immorlica, N., Karger, D., Minkoff, M., Mirrokni, V.: 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. (2004)
2. Ravi, R., Sinha, A.: Hedging uncertainty: Approximation algorithms for stochastic optimization problems. In: Proceedings of the 10th International Conference on Integer Programming and Combinatorial Optimization (IPCO). (2004) *GSIA Working Paper 2003-E68*.
3. Gupta, A., Pál, M., Ravi, R., Sinha, A.: Boosted sampling: Approximation algorithms for stochastic optimization. In: Proceedings of the 36th Annual ACM Symposium on Theory of Computing. (2004)
4. Shmoys, D., Swamy, C.: Stochastic optimization is (almost) as easy as deterministic optimization. In: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science. (2004)
5. Gupta, A., Ravi, R., Sinha, A.: An edge in time saves nine: Lp rounding approximation algorithms. In: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science. (2004)
6. Becchetti, L., Könemann, J., Leonardi, S., Pál, M.: Sharing the cost more efficiently: Improved approximation for multicommodity rent-or-buy. In: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms. (2005)
7. Gupta, A., Kumar, A., Pál, M., Roughgarden, T.: 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. (2003) 606–615

8. Kumar, A., Gupta, A., Roughgarden, T.: A constant factor approximation algorithm for the multicommodity rent-or-buy problem. In: Proceedings of the 43rd Annual Symposium on Foundations of Computer Science. (2002)
9. Karger, D.R., Minkoff, M.: Building steiner trees with incomplete global knowledge. In: Proceedings of the 41st Annual Symposium on Foundations of Computer Science. (2000) 613–623
10. Hayrapetyan, A., Swamy, C., Tardos, E.: Network design for information networks. In: ACM-SIAM Symposium on Discrete Algorithms. (2005)
11. Dantzig, G.B.: Linear programming under uncertainty. Management Sci. **1** (1955) 197–206
12. Beale, E.M.L.: On minimizing a convex function subject to linear inequalities. J. Roy. Statist. Soc. Ser. B. **17** (1955) 173–184; discussion, 194–203 (Symposium on linear programming.).
13. Birge, J.R., Louveaux, F.: Introduction to stochastic programming. Springer Series in Operations Research. Springer-Verlag, New York (1997)
14. Kall, P., Wallace, S.W.: Stochastic programming. Wiley-Interscience Series in Systems and Optimization. John Wiley & Sons Ltd., Chichester (1994)
15. Schultz, R., Stougie, L., van der Vlerk, M.H.: Two-stage stochastic integer programming: a survey. Statist. Nederlandica **50** (1996) 404–416
16. Agrawal, A., Klein, P., Ravi, R.: When trees collide: an approximation algorithm for the generalized steiner problem on networks. SIAM J. Comput. **24** (1995) 440–456 (Preliminary version in *23rd STOC*, 1991).
17. Goemans, M.X., Williamson, D.P.: A general approximation technique for constrained forest problems. SIAM J. Comput. **24** (1995) 296–317 (Preliminary version in *5th SODA*, 1994).
18. Könemann, J., Leonardi, S., Schäffer, G.: A group-strategyproof mechanism for steiner forests. In: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms. (2005)

A Additional Proofs

Lemma 2. *The mapping from non-lonely layers of $\mathcal{L}$ to layers of $\mathcal{L}'$ is one to one; that is, distinct layers of $\mathcal{L}$ map to distinct layers of $\mathcal{L}'$.*

Proof (Lemma 2). The lemma follows from the at first surprising fact that *the execution $\mathcal{E}'$ never builds any edges (of non-zero length)*. Indeed, suppose that two layers $\ell_1 = (C_1, I)$ and $\ell_2 = (C_2, I)$ of $\mathcal{L}$ map to the same layer $\ell' = (C', \gamma I)$. But this means that there are two distinct active (super)-nodes of the graph H that are in the same cluster, and hence by the invariant maintained by the algorithm, $\mathcal{E}'$ must have built a path between them.

The fact that two active clusters in $\mathcal{E}'$ never merge can be established by comparing the executions $\mathcal{E}'$ (recall, $\mathcal{E}' = \mathsf{Timed}(H, \mathcal{D}, \gamma T_{-g})$) and $\mathsf{Timed}(G, \mathcal{D}, T_{-g})$ side by side. It can be shown that both executions behave identically in that edges in both get tight at the same time (see [7]).

Hence, suppose that two active clusters in $\mathcal{E}'$ merge, each of them containing an active (super)-terminal w_{C_1} and w_{C_2} respectively (recall that each (super)-node w_C of H corresponds to an equivalence class C of the relation $\mathcal{R}_{-g}$). But this means that the path between a pair of active vertices $u \in C_1$ and $v \in C_2$ got tight in the execution $\mathsf{Timed}(G, \mathcal{D}, \gamma T_{-g})$ as well, implying that u and v should be in the same equivalence class of $\mathcal{R}_{-g}$, and hence C_1 and C_2 must be the same equivalence class, contradicting the assumption that w_{C_1} and w_{C_2} are distinct. $\square$