Greedy Algorithms for Steiner Forest

Anupam Gupta
Carnegie Mellon University
Pittsburgh, PA
anupamg@cs.cmu.edu

Amit Kumar
IIT Delhi
New Delhi, India
amitk@cse.iitd.ac.in

ABSTRACT

In the Steiner Forest problem, we are given terminal pairs $\{s_i, t_i\}$, and need to find the cheapest subgraph which connects each of the terminal pairs together. In 1991, Agrawal, Klein, and Ravi gave a primal-dual constant-factor approximation algorithm for this problem. Until this work, the only constant-factor approximations we know are via linear programming relaxations.

In this paper, we consider the following greedy algorithm: *Given terminal pairs in a metric space, a terminal is active if its distance to its partner is non-zero. Pick the two closest active terminals (say s_i, t_j), set the distance between them to zero, and buy a path connecting them. Recompute the metric, and repeat.* It has long been open to analyze this greedy algorithm. Our main result shows that this algorithm is a constant-factor approximation.

We use this algorithm to give new, simpler constructions of cost-sharing schemes for Steiner forest. In particular, the first "strict" cost-shares for this problem implies a very simple combinatorial sampling-based algorithm for stochastic Steiner forest.

1. INTRODUCTION

In the Steiner forest problem, given a metric space and a set of source-sink pairs $\{s_i, t_i\}_{i=1}^{K}$, a feasible solution is a forest such that each source-sink pair lies in the same tree in this forest. The goal is to minimize the cost, i.e., the total length of edges in the forest. This problem is a generalization of the Steiner tree problem, and hence APX-hard. The constant-factor approximation algorithms currently known for it are all based on linear programming techniques. The first such result was an influential primal-dual 2-approximation due to Agrawal, Klein, and Ravi [1]; this was simplified by Goemans and Williamson [9] and extended to many "constrained forest" network design problems. Other works have since analyzed the integrality gaps of the natural linear program, and for some stronger LPs; see § 1.2.

However, no constant-factor approximations are known based on "purely combinatorial" techniques. Some natural algorithms have been proposed, but these have defied analysis for the most part. The simplest is the *paired greedy algorithm* that repeatedly connects the yet-unconnected s_i-t_i pair at minimum mutual distance; this is no better than $\Omega(\log n)$ (see Chan, Roughgarden, and Valiant [6] or the full version). Even greedier is the so-called *gluttonous algorithm* that connects the closest two yet-unsatisfied terminals regardless of whether they were "mates". The performance of this algorithm has been a long-standing open question. Our main result settles this question.

Theorem 1.1 *The gluttonous algorithm is a constant-factor approximation for Steiner Forest.*

We then apply this result to obtain a simple combinatorial approximation algorithm for the *two-stage stochastic version* of the Steiner forest problem. In this problem, we are given a probability distribution π defined over subsets of demands. In the first stage, we can buy some set E_1 of edges. Then in the second stage, the demand set is revealed (drawn from π), and we can extend the set E_1 to a feasible solution for this demand set. However, these edges now cost $\sigma > 1$ times more than in the first stage. The goal is to minimize the total expected cost. It suffices to specify the set E_1—once the actual demands are known, we can augment using our favorite approximation algorithm for Steiner forest. Our simple algorithm is the following: sample $\lceil \sigma \rceil$ times from the distribution π, and let E_1 be the Steiner forest constructed by (a slight variant of) the gluttonous algorithm on union of these $\lceil \sigma \rceil$ demand sets sampled from π.

Theorem 1.2 *There is a combinatorial (greedy) constant-factor approximation algorithm for the stochastic Steiner forest problem.*

Showing that such a "boosted sampling" algorithm obtained a constant factor approximation had proved elusive for several years now; the only constant-factor approximation for stochastic Steiner forest was a complicated primal-dual algorithm with a large approximation factor [10]. Our result is based on the first cost sharing scheme for the Steiner forest problem which is constant strict with respect to a constant factor approximation algorithm; see § 5 for the formal definition. Such a cost sharing scheme can be used for designing

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
STOC'15, June 14–17, 2015, Portland, Oregon, USA.
Copyright © 2015 ACM 978-1-4503-3536-2/15/06 ...$15.00.
http://dx.doi.org/10.1145/2746539.2746590.

approximation algorithms for several stochastic network design problems for the Steiner forest problem. In particular, we obtain the following results:

- For multi-stage stochastic optimization problem for Steiner forest, our strict-cost sharing scheme along with the fact that it is also cross-monotone implies the first $O(1)^k$-approximation algorithm, where k denotes the number of stages (see [12] for formal definitions and the relation with cost sharing).

- Consider the online stochastic problem, where given a set of source-sink pairs $\mathscr{D}$ in a metric $\mathcal{M}$, and a probability distribution π over subsets of $\mathscr{D}$ (i.e., over $2^{\mathscr{D}}$), an adversary chooses a parameter k, and draws k times independently from π. The on-line algorithm, which can sample from π, needs to maintain a feasible solution over the set of demand pairs produced by the adversary at all time. The goal is to minimize the expected cost of the solution produced by the algorithm, where the expectation is over π and random coin tosses of the algorithm. Our cost sharing framework gives the first constant competitive algorithm for this problem, generalizing the result of Garg et al. [8] which works for the special case when π is a distribution over $\mathscr{D}$ (i.e., singleton subsets of $\mathscr{D}$).

1.1 Ideas and Techniques

We first describe the gluttonous algorithm. Call a terminal *active* if it is not yet connected to its mate. Recall: our algorithm merges the two active terminals that are closest in the current metric (and hence zeroes out their distance). At any point of time, we have a collection of *supernodes*, each supernode corresponding to the set of terminals which have been merged together. A supernode is *active* if it contains at least one active terminal. Hence the algorithm can be alternatively described thus: merge the two active supernodes that are closest (in the current metric) into a new supernode. (A formal description of the algorithm appears in §2.)

The analysis has two conceptual steps. In the first step, we reduce the problem to the special case when the optimal solution can be (morally) assumed to be a single tree (formally, we reduce to the case where the gluttonous' solution is a refinement of the optimal solution). The proof for this part is simple: we take an optimal forest, and show that we can connect two trees in the forest if the gluttonous algorithm connects two terminals lying in these two trees, incurring only a factor-of-two loss.

The second step of the analysis starts with the tree solution T promised by the first step of the analysis. As the gluttonous algorithm proceeds, the analysis alters T to maintain a candidate solution to the current set of supernodes. E.g., if we merge two active supernodes u and v to get a new supernode uv. We want to alter the solution T on the original supernodes to get a new solution T', say by removing an edge from the (unique) u-v path in T, and then short-cutting any degree two inactive supernode in T' (see Figure 1.1 for an example). The hope is to argue that the distance between u and v—which is the cost incurred by gluttonous—is commensurate to the cost of the edge of T which gets removed during this process. This would be easy if there were a long edge on u-v path in the tree T. The problem: this may not hold for every pair of supernodes we merge. Despite this,

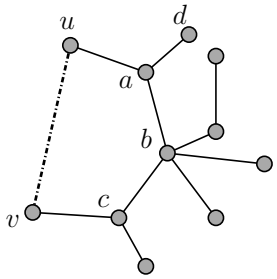

Figure 1.1: Example showing the construction of tree T' from T, which is shown in solid lines. If we merge u and v, we can remove the edge (a, b) to get the tree T'. Assuming a is not active, we can also short-cut the degree 2 vertex a in T' by replacing the edges (u, a) and (a, d) by the edge (u, d).

our analysis shows that the bad cases cannot happen too often, and so we can perform this charging argument in an amortized sense.

Our analysis is flexible and extends to other variants of the gluttonous algorithm. A natural variant is one where, instead of merging the two closest active supernodes, we contract the edges on a shortest path between the two closest active supernodes. Yet another variant is a *timed* version of the algorithm, which is inspired by a timed version of the primal-dual algorithm [16], and is crucial for obtaining the strict cost-shares described next.

Loosely speaking, a cost-sharing method takes an algorithm $\mathscr{A}$ and divides the cost incurred by the algorithm on an instance among the terminals $\mathscr{D}$ in that instance. The "strictness" property ensures that if we partition $\mathscr{D}$ arbitrarily into $\mathscr{D}_1 \cup \mathscr{D}_2$, and build a solution $\mathscr{A}(\mathscr{D}_1)$ on $\mathscr{D}_1$, then the cost-shares of the terminals in $\mathscr{D}_2$ would suffice to augment the solution $\mathscr{A}(\mathscr{D}_1)$ to one for $\mathscr{D}_2$ as well.

A natural candidate for $\mathscr{A}$ is the GW primal-dual algorithm, and the cost-shares are equally natural: we divide up the cost of growing moats among the active terminals in the moat. However, the example in Figure 1.2 shows why this fails when $\mathscr{D}_2$ consists of just the demand pair $\{s, \bar{s}\}$. When run on all the terminals, the primal-dual algorithm stops at time 1, with all terminals getting a cost-share of 1. On the other hand, if we run $\mathscr{A}$ on $\mathscr{D}_1$, it finds a solution which has N connected components, each connecting s_i and $\bar{s}_i$ for $i = 1, \ldots, N$. Then connecting s and $\bar{s}$ costs $2N$, which is much more than their total cost share.

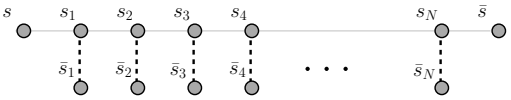

Figure 1.2: Distances $d(s_i, \bar{s}_i)$ are 2 for all i. Further, $d(s_i, s_{i+1}) = d(\bar{s}_i, \bar{s}_{i+1}) = 2$ for $i = 1, \ldots, N-1$. The distances $d(s, s_1)$ and $d(s_N, \bar{s})$ are slightly larger than 2. The dotted lines indicate the forest returned by the GW primal-dual algorithm when run on the demand set $\{(s_i, \bar{s}_i) : i = 1, \ldots, N\}$

To avoid this problem, [11, 7] run the primal-dual algorithm for longer than required, and give results for the case

when $\mathscr{D}_2$ contains a single demand pair. However, the arguments become much more involved than those in the analysis of GW algorithm [9]—the main reason is the presence of "dead" moats which cause some edges to become tight, and the cost shares of active terminals cannot account for such edges. In our case, the combinatorial (greedy) nature of our algorithm/analysis means we do not face such issues. As a result, we can obtain such strict cost sharing methods (when $\mathscr{D}_2$ is a singleton set) with much simpler analysis, albeit with worse constants than those in [11, 7]). We refer to this special case of strictness property as *uni-strictness*.

Our analysis for the general case where $\mathscr{D}_2$ contains multiple demand pairs requires considerably more work; but note that these are the first known strict cost shares for this case, the previous primal-dual techniques could not handle the complexity of this general case. Here, we want $\mathscr{A}$ to build as many edges as possible, and the cost share χ to be as large as possible. Since the gluttonous algorithm tends to build fewer edges than primal-dual (the dead moats causing extra connections and more edges), we end up using the primal-dual algorithm as the algorithm $\mathscr{A}$. However, to define the cost-shares, we use the (timed) gluttonous algorithm in order to avoid the issues with dead moats. The analysis then proceeds via showing a close correspondence between the primal-dual and gluttonous algorithms. Although this is not involved, it needs to carefully match the two runs.

1.1.1 Outline of Paper

We first describe some related work in § 1.2, and give some important definitions in § 1.3. Then we describe the gluttonous algorithm formally in § 2, and then analyze this algorithm in § 3. We briefly describe the timed version in § 4, which gets used in the subsequent section on cost sharing. The cost-sharing scheme and its properties are described in § 5. The details of the analysis and the simpler analysis in the uni-strict case are deferred to the full version. We also analyse the variant of the gluttonous which relies on contract shortest path between supernodes in the full version.

1.2 Related Work

The first constant-factor approximation for the Steiner forest problem was due to Agrawal, Klein, and Ravi [1] using a primal-dual approach; it was refined and generalized by Goemans and Williamson [9] to a wider class of network design problems. The primal-dual analysis also bounds integrality gap of the the natural LP relaxation (based on covering cuts) by a factor of 2. Different approximation algorithms for Steiner forest based off the same LP, and achieving the same factor of 2, are obtained using the iterative rounding technique of Jain [14], or the integer decomposition techniques of Chekuri and Shepherd [5]. A stronger LP relaxation was proposed by Könemann, Leonardi, and Schäfer [15], but it also has an integrality gap of 2 [16].

The Steiner forest problem has also been studied online, with elegant algorithms and analyses. Awerbuch, Azar, and Bartal [2] showed that the greedy algorithm is $O(\log^2 k)$-competitive, where k is the number of terminal pairs. Berman and Coulston [3] gave an improved algorithm (inspired by the primal-dual approach) which is $O(\log k)$-competitive.

The special case of Steiner forest is the *Steiner tree* problem, where all the demands share a common (source) terminal, has been well-studied in the network design community. There is a simple 2-approximation algorithm for this problem: iteratively find the closest terminal to the source vertex, and merge these two terminals. There have been several changes to this simple greedy algorithm leading to improved approximation ratios (see e.g. [18]). Byrka et al. [4] improved these results to a $\ln 4 + \varepsilon \approx 1.39$-approximation algorithm, which is based on rounding a stronger LP relaxation for this problem.

The stochastic Steiner tree/forest problem was defined by Immorlica, Karger, Minkoff, and Mirrokni [13], and further studied by [12], who proposed the boosted-sampling framework of algorithms. The analysis of these algorithms is via "strict" cost sharing methods, which were studied by [11, 7]. A constant-factor approximation algorithm (with a large constant) was given for stochastic Steiner forest by [10] based on primal-dual techniques; it is much more complicated than the algorithm and analysis based on the greedy techniques in this paper.

1.3 Preliminaries

Let $\mathcal{M} = (V, d)$ be a metric space on n points; assume all distances are either 0 or at least 1. Let the *demands* $\mathscr{D} \subseteq \binom{V}{2}$ be a collection of source-sink pairs that need to be connected. By splitting vertices, we may assume that the pairs in $\mathscr{D}$ are disjoint. A node is a *terminal* if it belongs to some pair in $\mathscr{D}$. Let K denote the number of terminals pairs, and hence there are $2K$ terminals. For a terminal u, let $\bar{u}$ be the unique vertex such that $\{u, \bar{u}\} \in \mathscr{D}$; we call $\bar{u}$ the *mate* of u.

For a Steiner forest instance $\mathscr{I} = (\mathcal{M}, \mathscr{D})$, a solution $\mathscr{F}$ to the instance $\mathscr{I}$ is a forest such that each pair $\{u, \bar{u}\} \in \mathscr{D}$ is contained within the vertex set $V(T)$ for some tree $T \in \mathscr{F}$. For a tree $T = (V, E_T)$, let $\mathsf{cost}(T) := \sum_{e \in E_T} d(e)$ be the sum of lengths of edges in T. Let $\mathsf{cost}(\mathscr{F}) := \sum_{T \in \mathscr{F}} \mathsf{cost}(T)$ be the cost of the forest $\mathscr{F}$. Our goal is to find a solution of minimum cost.

2. THE GLUTTONOUS ALGORITHM

We begin with some definitions. Given a Steiner forest instance $\mathscr{I} = (\mathcal{M}, \mathscr{D})$, a *supernode* is a subset of terminals. A *clustering* $\mathscr{C} = \{S_1, S_2, \ldots, S_q\}$ is a partition of the terminal set into supernodes. The *trivial clustering* places each terminal in its own singleton supernode. Our algorithm maintains a clustering at all points in time. Given a clustering, a terminal u is *active* if it belongs to a supernode S that does not contain its mate $\bar{u}$. A supernode S is *active* if it contains some active terminal. In the trivial clustering, all the terminals and supernodes are active.

Given a clustering $\mathscr{C} = (S_1, S_2, \ldots, S_q)$, define a new metric $\mathcal{M}/\mathscr{C}$ called the *$\mathscr{C}$-puncturing* of metric $\mathcal{M}$. To get this, take a complete graph on V; for an edge $\{u, v\}$, set its length to be $d(u, v)$ if (i) at least one of u, v is a non-terminal node, or (ii) both u and v are terminals but lie in different supernodes in $\mathscr{C}$; and to zero if both u, v lie in the same supernode in $\mathscr{C}$. Call this graph $G_{\mathscr{C}}$, and defined the $\mathscr{C}$-punctured distance to be the shortest-path distance in this graph, denoted by $d_{\mathcal{M}/\mathscr{C}}(\cdot, \cdot)$. One can think of this as modifying the metric $\mathcal{M}$ by collapsing the terminals in each of the supernodes in $\mathscr{C}$ to a single node. Given clustering $\mathscr{C}$ and two supernodes S_1 and S_2, the distance between them is naturally defined as

$$d_{\mathcal{M}/\mathscr{C}}(S_1, S_2) = \min_{u \in S_1, v \in S_2} d_{\mathcal{M}/\mathscr{C}}(u, v).$$

The gluttonous algorithm is as follows:

> Start with $\mathscr{C}$ being the trivial clustering, and E' being the empty set. While there exist active supernodes in $\mathscr{C}$, do the following:
>
> (i) Find active supernodes S_1, S_2 in $\mathscr{C}$ with minimum $\mathscr{C}$-punctured distance. (Break ties arbitrarily but consistently, say choosing the lexicographically smallest pair.)
>
> (ii) Update the clustering to
>
> $$\mathscr{C} \leftarrow (\mathscr{C} \setminus \{S_1, S_2\}) \cup \{S_1 \cup S_2\},$$
>
> (iii) Add to E' the edges corresponding to the inter-supernode edges on the shortest path between S_1, S_2 in the graph $G_{\mathscr{C}}$.
>
> Finally, output a maximal acyclic subgraph F of E'.

Above, we say we *merge* S_1, S_2 to get the new supernode $S_1 \cup S_2$. The *merging distance* for the merge of S_1, S_2 is the $\mathscr{C}$-punctured distance $d_{\mathcal{M}/\mathscr{C}}(S_1, S_2)$, where $\mathscr{C}$ is the clustering just before the merge. Since each active supernode contains an active terminal, if $u \in S_1$ and $v \in S_2$ are both active, then when we talk about merging u, v, we mean merging S_1, S_2.

Note that the length of the edges added in step (iii) is equal to $d_{\mathcal{M}/\mathscr{C}}(S_1, S_2)$. The algorithm maintains the following invariant: if S is a supernode, then the terminals in S lie in the same connected component of F[1] – indeed, it is easy to see that the set of edges in E' satisfies this property, and the same holds for F because both the sets of edges have the same connected components. The algorithm terminates when there are no more active terminals, so each terminal shares a supernode with its mate, and hence the final forest F connects all demand pairs. Since the edges added to E' have total length at most the sum of the merging distances, and we output a maximal sub-forest of E', we get:

Fact 2.1 *The cost of the Steiner forest solution output is at most the sum of all the merging distances.*

We emphasize that the edges added in Step (iii) are often overkill: the metric $\mathcal{M}/E'$ (where the edges in E' have been contracted) has no greater distances than the metric $\mathcal{M}/\mathscr{C}$ that we focus on. The advantage of the latter over the former is that distances in $\mathcal{M}/\mathscr{C}$ are well-controlled (i.e., the distances between the two closest active terminals is non-decreasing over time; see Claim 3.1), whereas those in $\mathcal{M}/E'$ change drastically over time (with the distance between the closest active terminals changing less predictably).

Consider the example in Figure 2.3, where the distances for missing edges are inferred by computing shortest-path distances.

Here, we first merge $\{s_1, \bar{s}_1\}$ to form a supernode, say A, which is inactive. Next we merge s_3 and $\bar{s}_2$ to form another supernode, say B. The active supernodes are B, s_2, and $\bar{s}_3$, so we next merge s_2 with B to form supernode C, and finally merge $\bar{s}_3$ with C. When the algorithm ends, there are two (inactive) supernodes corresponding to the sets $\{s_1, \bar{s}_1\}$ and $\{s_2, s_3, \bar{s}_2, \bar{s}_3\}$. However, the forest produced will have only a single tree, which consists of the set of edges drawn in the figure.

[1]The converse is not necessarily true: if we connect S_1 and S_2 by buying edges connecting them both to some inactive supernode S_3, then F has a tree connecting all three, but the clustering has $S_1 \cup S_2$ separate from S_3. Indeed, inactive supernodes never get merged again, whereas inactive trees may.

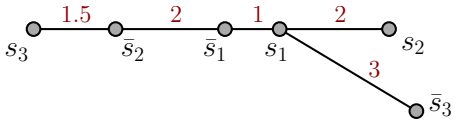

Figure 2.3: Figure for the gluttonous algorithm.

3. THE ANALYSIS FOR GLUTTONOUS

We analyze the algorithm in two steps. One conceptual problem is in controlling what happens when gluttonous connects two nodes in different trees of the optimal forest. To handle this, we show in § 3.1 how to preprocess the optimal forest $\mathscr{F}^\star$ to get a near-optimal forest $\mathscr{F}^{\star\star}$ such that the final clustering of the gluttonous algorithm is a refinement of this near-optimal forest. (I.e., if u and v are in the same supernode in the gluttonous clustering, then they lie in the same tree in $\mathscr{F}^{\star\star}$.) This makes it easier to then account for the total merging distance, which we do in § 3.2. A crucial observation, which makes the analysis much cleaner, is the following.

Claim 3.1 (Merging Distances are Monotone) *If S, T are merged before S', T' in gluttonous, then the merging distance for S, T is no greater than the merging distance for S', T'.*

3.1 A "Well-Behaved" Near-Optimal Solution

Since gluttonous is deterministic and we break ties consistently, given a Steiner forest instance $\mathscr{I} = (\mathcal{M}, \mathscr{D})$ there is a unique final clustering $\mathscr{C}^f$ produced by the algorithm.

Definition 3.2 (Faithful) *A forest $\mathscr{F}$ is* faithful *to a clustering $\mathscr{C}$ if each supernode $S \in \mathscr{C}$ is contained within a single tree in $\mathscr{F}$. (I.e., for all $S \in \mathscr{C}$, there exists $T \in \mathscr{F}$ such that $S \subseteq V(T)$.)*

Note that every forest is faithful to the trivial clustering consisting of singletons. The following theorem, whose proof is deferred to the full version, shows that there exist near-optimal solutions which are faithful to gluttonous' final clustering.

Theorem 3.3 (Low-Cost and Faithful) *Consider an optimal solution $\mathscr{F}^\star = \{T_1^\star, T_2^\star, \ldots, T_p^\star\}$ for the Steiner forest instance $\mathscr{I} = (\mathcal{M}, \mathscr{D})$. There exists another solution $\mathscr{F}^{\star\star}$ for instance $\mathscr{I}$ such that*

(a) $\mathsf{cost}(\mathscr{F}^{\star\star}) \le 2\mathsf{cost}(\mathscr{F}^\star)$, and

(b) $\mathscr{F}^{\star\star}$ is faithful to the final clustering $\mathscr{C}^f$ produced by the gluttonous algorithm.

3.2 Charging to this Near-optimal Solution

Let $\mathscr{F}^\star = \{T_1^\star, \ldots, T_p^\star\}$ be a solution to the Steiner forest instance. The main result of this section is:

Theorem 3.4 *If the forest $\mathscr{F}^\star$ is faithful to the final clustering $\mathscr{C}^f$ of the gluttonous algorithm, then the cost of the gluttonous algorithm is $48 \cdot \mathsf{cost}(\mathscr{F}^\star)$.*

Since by Theorem 3.3 there is a forest $\mathscr{F}^\star$ with cost at most twice the optimum that is faithful to gluttonous' final clustering $\mathscr{C}^f$, applying Theorem 3.4 to this forest proves Theorem 1.1.

We now prove Theorem 3.4. At a high level, the proof proceeds thus: we consider the run of the gluttonous algorithm, and maintain for each iteration t a "candidate" forest $\mathscr{F}_t$ that is a solution to the remaining instance. We show that in an amortized sense, at each step the cost of forest $\mathscr{F}_t$ decreases by an amount which is a constant fraction of the cost incurred by gluttonous. Since the starting cost of this forest is at most a constant times the optimal cost, so is the total merging cost of the gluttonous, proving the result.

For Steiner forest instance $\mathscr{I}$, assume that $\mathscr{C}^f$ is gluttonous' final clustering, and $\mathscr{F}^\star$ is faithful to $\mathscr{C}^f$. Let $\mathscr{C}^{(t)}$ be the gluttonous clustering at the beginning of the iteration t, with $\mathscr{C}^{(t)}_{active}$ being the active supernodes. It will be useful to view this clustering as giving us an induced Steiner forest instance $\mathscr{I}_t$ on the metric whose points are the supernodes in $\mathscr{C}^{(t)}$ and where distances are given by the punctured metric $d_{\mathcal{M}/\mathscr{C}^{(t)}}$, where the terminals in the instance $\mathscr{I}_t$ are supernodes in $\mathscr{C}^{(t)}_{active}$, and where active supernodes $\{S_1, S_2\}$ are mates if there is a pair $\{u, \bar{u}\}$ such that $u \in S_1$ and $\bar{u} \in S_2$. (Supernodes no longer have unique mates, but this property was only used for convenience in Theorem 3.3). For any iteration t, the subsequent run of gluttonous is just a function of this induced instance $\mathscr{I}_t$. Indeed, given the instance $\mathscr{I}_t$, gluttonous outputs a final clustering which is same as $\mathscr{C}^f$ except the inactive supernodes in $\mathscr{C}^{(t)}$ are absent. I.e., the inactive supernodes in $\mathscr{C}^{(t)}$ will not play a role, but all the active supernodes will continue to combine in the same way in $\mathscr{I}_t$ as in $\mathscr{I}$. We now inductively maintain a forest $\mathscr{F}^{(t)}$ such that

(I1) $\mathscr{F}^{(t)}$ is a feasible solution to this Steiner forest instance $\mathscr{I}_t$, and

(I2) $\mathscr{F}^{(t)}$ maintains the connectivity structure of $\mathscr{F}^\star$, i.e., if u and v are two active terminals which are in the same tree in $\mathscr{F}^\star$, then the supernodes containing u and v lie in the same tree in $\mathscr{F}^{(t)}$.

And we will charge the cost of gluttonous to reductions in the cost of this forest $\mathscr{F}^{(t)}$.

3.2.1 The "candidate" forest $\mathscr{F}^{(t)}$

The initial clustering $\mathscr{C}^{(1)}$ is the trivial clustering consisting of singleton terminals; we set $\mathscr{F}^{(1)}$ to $\mathscr{F}^\star$. Since $\mathscr{I}_1$ is the original instance, $\mathscr{F}^{(1)}$ is feasible for it; invariant (I2) is satisfied trivially.

For an iteration t, let $E(\mathscr{F}^{(t)})$ denote the edges in $\mathscr{F}^{(t)}$. Note that an edge $e \in E(\mathscr{F}^{(t)})$ between two supernodes $S_1, S_2 \in \mathscr{C}^{(t)}$ corresponds to an edge between two terminals u, v in the original metric $\mathcal{M}$, where $u \in S_1, v \in S_2$. Define $\mathsf{length}(e)$ as $d_{\mathcal{M}}(u, v)$, the length of the edge e in the original metric. Note that the length of e in the metric $\mathcal{M}/\mathscr{C}^{(t)}$ may be smaller than $\mathsf{length}(e)$. For every edge $e \in \mathscr{F}^{(t)}$, we shall also maintain a *potential* of e, denoted $\psi(e)$. Initially, for $t = 1$, the potential $\psi(e) = \mathsf{length}(e)$ for all $e \in E(\mathscr{F}^{(1)})$. During the course of the algorithm, the potential $\psi(e) \geq \mathsf{length}(e)$; we describe the rule for maintaining potentials below. Intuitively, an edge $e \in \mathscr{F}^{(t)}$ would have been obtained by *short-cutting* several edges of $\mathscr{F}^\star$, and $\psi(e)$ is equal to the total length of these edges.

Suppose we have a clustering $\mathscr{C}^{(t-1)}$ and a forest $\mathscr{F}^{(t-1)}$ which satisfies invariants (I1) and (I2). If we now merge two supernodes $S_1, S_2 \in \mathscr{C}^{(t-1)}$ to get clustering $\mathscr{C}^{(t)}$, we have to update the forest $\mathscr{F}^{(t-1)}$ to get to $\mathscr{F}^{(t)}$ using procedure `UpdateForest` given in Figure 3.2.1. The main idea is simple: when we merge the nodes corresponding to S_1 and S_2 in $\mathscr{F}^{(t-1)}$ into a single node, this creates a cycle. Removing any edge from the cycle maintains the invariants, and reduces the cost of the new forest: we remove the edge with the highest potential from the cycle. We further reduce the cost by getting rid of Steiner vertices, which correspond to inactive supernodes in $\mathscr{F}^{(t)}$ with degree 2. More formally, given two edges $e' = \{u', v\}, e'' = \{u'', v\}$ with a common end-point v, the operation *short-cut* on e', e'' replaces them by a single edge $\{u', u''\}$. Whenever we see a Steiner vertex of degree 2 in $\mathscr{F}^{(t)}$, we shortcut the two incident edges.

Some more comments about the procedure `UpdateForest`. In Step 1, the existence of the tree T follows from the invariant property (I2) and the faithfulness of $\mathscr{F}^\star$ to $\mathscr{C}^f$. Since the terminals in $S_1 \cup S_2$ are in the same tree in $\mathscr{F}^\star$, the invariant means they belong to the same tree in $\mathscr{F}^{(t-1)}$, and the construction ensures they remain in the same tree in $\mathscr{F}^{(t)}$. When we short-cut edges e', e'' to get a new edge e, we define the potential of the new edge e to be $\psi(e) := \psi(e') + \psi(e'')$. It is also easy to check that $\mathscr{F}^{(t)}$ is a feasible solution to the instance $\mathscr{I}_t$. Indeed, the only difference between $\mathscr{I}_{t-1}$ and $\mathscr{I}_t$ is the replacement of S_1, S_2 by S. If S becomes inactive, there is nothing to prove. If S remains active, then the tree containing S must will also have also have the supernodes which were paired with S_1 and S_2 in the instance $\mathscr{I}_{t-1}$. It is also easy to check that the invariant property (I2) continues to hold. The following claim, whose proof is deferred to the full version, proves some more crucial properties of the forest $\mathscr{F}^{(t)}$.

Claim 3.5 *For all iterations t, the Steiner nodes in $\mathscr{F}^{(t)}$ have degree at least 3. Therefore, there are at most 2 iterations of the while loop in Step 6 of the `UpdateForest` algorithm.*

Here's the plan for rest of the analysis. Let's fix a tree $T^\star$ of $\mathscr{F}^\star$, and account for only those merging costs which merge two supernodes with terminals in $T^\star$. (Summing over all trees in $\mathscr{F}^\star$ and using the faithfulness of $\mathscr{F}^\star$ to $\mathscr{C}^f$ will ensure all merging costs are accounted for.) Since $\mathscr{F}^{(t)}$ is obtained by repeatedly contracting nodes and removing unnecessary edges, in each iteration t there is a unique tree $T^{(t)}$ in the forest $\mathscr{F}^{(t)}$ corresponding to the tree $T^\star$, namely the tree containing the active supernodes with terminals belonging to $T^\star$. Call an iteration of the gluttonous algorithm a *relevant iteration* (with respect to $T^\star$) if gluttonous merges two supernodes from the tree $T^{(t)}$ in this iteration. For brevity, we drop the phrase "*w.r.t. $T^\star$*" in the sequel.

Next we show that the total potential of the edges does not change over time. Let $\mathsf{del}(t)$ denote the set of edges which are deleted (from a cycle in Step 5) during the (relevant) iterations among $1, \ldots, t-1$. (Observe that $\mathsf{del}(t)$ does not include edges that are short-cut.)

Lemma 3.6 *For iteration t, the sum of potentials of edges $\mathsf{del}(t)$ and $E(T^{(t)})$ equals $\mathsf{cost}(T^\star)$. Further, $\psi(e) \geq \mathsf{length}(e)$ for all edges $e \in E(T^{(t)}) \cup \mathsf{del}(t)$.*

Algorithm `UpdateForest` $(\mathscr{C}^{(t-1)}, S_1, S_2)$ **:**

1. Let T be the tree in $\mathscr{F}^{(t-1)}$ containing the terminals in S_1 and S_2.
2. Merge S_1 and S_2 to a single node S in the tree T.
3. If the new supernode S becomes inactive, and has degree 2 in the tree T, then short-cut the two edges incident to S.
4. Let C denote the unique cycle formed in the tree T.
5. Delete the edge in the cycle C which has the highest potential.
6. While there is an inactive supernode in T which is a degree-2 vertex, short-cut the two incident edges to this vertex.

Figure 3.4: The procedure for updating $\mathscr{F}^{(t-1)}$ to $\mathscr{F}^{(t)}$.

Eventually $T^{(t)}$ has no active supernodes (for large t) and hence all its edges are deleted. Hence if $\mathsf{del}(\infty)$ denotes the edges deleted during all the relevant iterations in gluttonous, Lemma 3.6 implies $\sum_{e \in \mathsf{del}(\infty)} \psi(e) = \mathsf{cost}(T^\star)$. Let Δ_t denote the merging cost of some relevant iteration t: we now show how to charge this cost to the potential of some deleted edge in $\mathsf{del}(\infty)$. Formally, let N_t denote the number of active supernodes in $T^{(t)}$, at the *beginning* of iteration t.

Theorem 3.7 *If t_0 is relevant, there are at least $N_{t_0}/8$ edges in $\mathsf{del}(\infty)$ of potential at least $\Delta_{t_0}/6$.*

We defer the proof of Theorem 3.7 for the moment, and instead show how to use this to charge the merging costs and to prove Theorem 3.4, which in turn gives the main theorem of the paper.

Proof of Theorem 3.4: Let I^r denote the index set of all relevant iterations during the run of gluttonous. We now define a mapping g from I^r to $\mathsf{del}(\infty)$ such that: (i) for any edge $e \in \mathsf{del}(\infty)$, the pre-image $g^{-1}(e)$ has cardinality at most 8, and (ii) the potential $\psi(g(t)) \geq \Delta_t/6$ for all $t \in I^r$. To get this, consider a bipartite graph on vertices $I^r \cup \mathsf{del}(\infty)$ where a iteration $t \in I^r$ is connected to all edges $e \in \mathsf{del}(\infty)$ for which $\psi(e) \geq \Delta_t/6$. Theorem 3.7 shows this graph satisfies a Hall-type condition for such a mapping to exist; in fact a greedy strategy can be used to construct the mapping (there can be at most N_t relevant iterations after iteration t because each relevant iteration reduces the number of active supernodes by at least one).

Thus, the total merging cost of gluttonous during relevant iterations is at most

$$\sum_{t \in I^r} \Delta_t = \sum_{e \in \mathsf{del}(\infty)} \sum_{t \in g^{-1}(e)} \Delta_t \leq 48 \sum_{e \in \mathsf{del}(\infty)} \psi(e) = 48\, \mathsf{cost}(T^\star),$$

where the last equality follows from Lemma 3.6. By the faithfulness property, each iteration of gluttonous is relevant with respect to one of the trees in $\mathscr{F}^\star$, so summing the above expression over all trees gives the total merging cost to be at most $48\,\mathsf{cost}(\mathscr{F}^\star)$. ■

Combining Theorem 3.4 with Theorem 3.3 gives an approximation factor of 96 for the gluttonous algorithm. While we have not optimized the constants, but it is unlikely that our ideas will lead to constants in the single digits. Obtaining, for instance, a proof that the gluttonous algorithm is a 2-approximation (or some such small constant) remains a fascinating open problem.

3.2.2 *Proof of Theorem 3.7*

In order to prove Theorem 3.7, we need to understand the structure of the trees $T^{(t)}$ for $t \geq t_0$ in more detail. Let $\mathsf{del}_0([t_0 \ldots t))$ denote the edges deleted during the relevant iterations in $t_0, \ldots, t-1$, i.e., $\mathsf{del}([t_0 \ldots t)) := \mathsf{del}(t) \setminus \mathsf{del}(t_0)$. Observe that each edge of $T^{(t)}$ is either in $T^{(t_0)}$ or is obtained by short-cutting some set of edges of $T^{(t_0)}$. Hence we maintain a partition $\mathscr{E}(t)$ of the edge set $E(T^{(t_0)})$, such that there is a correspondence between edges $e \in T^{(t)} \cup \mathsf{del}([t_0 \ldots t))$ and sets $D_t(e) \in \mathscr{E}(t)$, such that $D_t(e)$ is the set of edges in $T^{(t_0)}$ which have been short-cut to form e.

For each set $D_t(e)$, let $\mathsf{head}(D_t(e))$ be the edge $e' \in D_t(e)$ with greatest length. If edge e is removed from $T^{(t)}$ in some relevant iteration t, we have $e \in \mathsf{del}([t_0 \ldots t'))$ for all $t' > t$, and the set $D_{t'}(e) = D_t(e)$ for all future partitions $\mathscr{E}(t')$.

Lemma 3.8 *There are at least $N_{t_0}/2$ edges of length at least $\Delta_{t_0}/6$ in tree $T^{(t_0)}$.*

Proof. Call an edge *long* if its length is at least $\Delta_{t_0}/6$, and let ℓ denote the number of long edges in the tree $T^{(t_0)}$. Deleting these ℓ edges from $T^{(t_0)}$ gives us $\ell + 1$ subtrees $C_1, C_2, \ldots, C_{l+1}$. Let C_i have n_i active supernodes and e_i edges. For each tree C_i where $n_i \geq 2$, take an Eulerian tour X_i and divide it into n_i disjoint segments by breaking the tour at the active supernodes. Each edge appears in two such segments, and each segment has at least six edges (since the distance between active supernodes is at least Δ_{t_0} and none of the edges are long), so $e_i \geq 3n_i$ when $n_i \geq 2$. This means the total number of edges in $T^{(t_0)}$ is at least three times the number of "social" supernodes (supernodes that do not lie in a component C_i with $n_i = 1$, in which they are the only supernode), *plus* those ℓ long edges that were deleted.

And how many such social supernodes are there? If $\ell \geq N_{t_0} + 1$, there may be none, but then we clearly have at least $N_{t_0}/2$ long edges. Else at least $N_{t_0} - \ell$ supernodes are social, so $T^{(t_0)}$ has at least $3(N_{t_0} - \ell) + \ell$ edges. Finally, since every Steiner vertex in $T^{(t_0)}$ has degree at least 3, the number of edges is less than $2N_{t_0}$. Putting these together gives $3\,N_{t_0} - 2\ell \leq 2\,N_{t_0}$ or $\ell \geq N_{t_0}/2$. □

Let L_0 be the set of long edges in $T^{(t_0)}$, and $\mathscr{E}(\infty)$ be the partition at the end of the process. Two cases arise:

- At least $N_{t_0}/8$ edges in L_0 are $\mathsf{head}(D_\infty(e))$ for some set $D_\infty(e) \in \mathscr{E}(\infty)$. Since each set in $\mathscr{E}(\infty)$ has only one head, there are $N_{t_0}/8$ such sets. In any such set

$D_\infty(e)$, $\psi(e) \geq \mathsf{length}(\mathsf{head}(D_\infty(e))) \geq \Delta_{t_0}/6$. Moreover, we must have removed e in some iteration between t_0 and the end, and hence $e \in \mathsf{del}([t_0 \ldots \infty)) \subseteq \mathsf{del}(\infty)$.

- More than than $3N_t/8$ edges in L_0 are not heads of any set in $\mathscr{E}(\infty)$. Take one such edge e_0 — the sets in $\mathscr{E}(t_0)$ are singleton sets and hence e_0 is the head of the set $D_{t_0}(e_0)$. Let t be the first (relevant) iteration such that e_0 is not the head of the set containing it in $\mathscr{E}(t)$, and suppose $e_0 = \mathsf{head}(D_{t-1}(e'))$ for some set $D_{t-1}(e') \in \mathscr{E}(t-1)$. In forming $\mathscr{F}^{(t)}$, we must have short-cut e' and some other edge e'' to form an edge $e \in \mathscr{F}^{(t)}$. Observe that $\mathsf{length}(\mathsf{head}(D(e'')) \geq \mathsf{length}(e_0)$, else e_0 would continue to be the head of $D(e)$. Moreover,

$$\begin{aligned} &\min\left(\psi(e'), \psi(e'')\right) \\ &\quad \geq \min\left(\mathsf{length}(\mathsf{head}(e')), \mathsf{length}(\mathsf{head}(e''))\right) \\ &\quad \geq \Delta_{t_0}/6. \end{aligned} \tag{3.1}$$

  By the discussion in Claim 3.5, one of e' and e'' must lie on the cycle formed when we merged two supernodes in $\mathscr{F}^{(t-1)}$, as in Step 4 of `UpdateForest`. Further, if e_t was the edge removed from this cycle, by the rule in Step 5 we get that the potential $\psi(e_t)$ is the maximum potential of any edge on this cycle, and hence $\psi(e_t) \geq \min(\psi(e'), \psi(e'')) \geq \Delta_{t_0}/6$. Hence we want to "charge" this edge $e_0 \in L_0$ to $e_t \in \mathsf{del}(\infty)$ (which has potential at least $\Delta_{t_0}/6$). However, up to three edges from L_0 may charge to e_t: this is because there can be at most three short-cut operations in any iteration (one from Step 3 and two from Step 6).

In both cases, we've shown the presence of at least $N_{t_0}/8$ edges in $\mathsf{del}(\infty)$ of potential $\Delta_{t_0}/6$, which completes the proof of Theorem 3.7. ■

4. A TIMED GREEDY ALGORITHM

We now give a version of the gluttonous algorithm, which we call `TimedGlut`, where supernodes are deemed active or inactive based on the current time and not whether the terminals in the supernode have paired up with their mates.[2] This version will be useful in getting a strict cost-sharing scheme. In order to obtain strict cost-sharing schemes, we need to compare two runs of the gluttonous algorithm on slightly different inputs: the two inputs will have the same underlying metric space, but the set of demands in one of the inputs will be a subset of that of the other. In general, the behaviour of the gluttonous algorithm can change drastically even if we alter the set of demands slightly. However, in the timed version of the gluttonous algorithm, we shall have more control over how the run of the algorithm changes when we change the demand set slightly.

The algorithm `TimedGlut` is very similar to the gluttonous algorithm except for what constitutes an active supernode. Again, we shall maintain clustering of terminals into supernodes; and at each iteration t, we shall merge the two closest active supernodes in the current metric. This will ensure that the merging distances are monotone(Theorem 3.1). Therefore, we can divide the iterations of the algorithm into *stages*, where stage i denotes the iterations where the merging distance lies in the range $[2^i, 2^{i+1})$.

For a terminal s, define $\mathsf{level}(s)$ as $\lceil \log_2 d_{\mathcal{M}}(s, \bar{s}) \rceil$ (recall that $\mathcal{M}$ denotes the original metric). For a supernode S, define its *leader* as the terminal in S with highest level; in case of ties, choose the terminal with the smallest index among these. A supernode S will be active during stage i of the algorithm if the leader of S has level at least i (otherwise it is declared inactive). Note that a supernode S can remain active even if for every terminal in it, its mate also lies in S. However, we can show that even this algorithm has constant approximation ratio.

Theorem 4.1 *The `TimedGlut` algorithm is a γ_{TG}-approximation algorithm for Steiner forest, where $\gamma_{TG} = 480$.*

5. COST SHARES FOR STEINER FOREST

A *cost-sharing* method is a function χ mapping triples of the form $(\mathcal{M}, \mathscr{D}, (s, \bar{s}))$ to the non-negative reals, where $(\mathcal{M}, \mathscr{D})$ is an instance of the Steiner forest problem, and $(s, \bar{s}) \in \mathscr{D}$. We require the cost-sharing method to be *budget-balanced*: if $\mathscr{F}^\star$ is an optimal solution to the instance $(\mathcal{M}, \mathscr{D})$ then

$$\sum_{(s,\bar{s}) \in \mathscr{D}} \chi(\mathcal{M}, \mathscr{D}, (s, \bar{s})) \leq \mathsf{cost}(\mathscr{F}^\star). \tag{5.2}$$

We now consider the notion of strict cost shares (the details for the construction of the simpler uni-strict cost shares are given in the full version).

Definition 5.1 *A cost-sharing function χ is β-*strict *with respect to an algorithm $\mathscr{A}$ if for all pairs of disjoint terminal sets $\mathscr{D}_1, \mathscr{D}_2$ lying in a metric $\mathcal{M}$, the following condition holds: if $\mathscr{D}$ denotes $\mathscr{D}_1 \cup \mathscr{D}_2$, then $\sum_{(s,\bar{s}) \in \mathscr{D}_2} \chi(\mathcal{M}, \mathscr{D}, (s, \bar{s}))$ is at least $1/\beta$ times the the cost of the optimal Steiner forest on $\mathscr{D}_2$ in the metric $\mathcal{M}/F$, where F is the forest returned by $\mathscr{A}$ on the input $(\mathcal{M}, \mathscr{D}_1)$.*

In addition to the `TimedGlut` algorithm, we will also need a timed primal-dual algorithm for Steiner forest, denoted by `TimedPD`. The input for the `TimedPD` algorithm is a set of terminals, each terminal s being assigned an *activity time* $\mathsf{time}(s)$ such that the terminal is *active* for all times $t \leq \mathsf{time}(s)$. The primal-dual algorithm grows moats around terminals as long as they are active and buys edges that ensure that if two moats meet at some time t, all the terminals in these moats that are active at time t lie in the same tree. One can do this in different ways (see, e.g., [11, 17, 15]); for concreteness we refer to the `KLS` algorithm of Könemann et al. [15] which gives the following guarantee:

Theorem 5.2 *If $\mathsf{time}(s) = \frac{1}{2} d_{\mathcal{M}}(s, \bar{s})$ for all terminals s, then the total cost of edges bought by the timed primal-dual algorithm `KLS` is at most $2 \cdot \mathsf{opt}(\mathscr{I})$.*

[2]Timed versions of the primal-dual algorithm for Steiner forest had been considered previously in [11, 15]; our version will be analogous to that of Könemann et al. [15] which were used to get cross-monotonic cost-shares for Steiner forest.

5.1 Strict Cost Shares

5.1.1 Defining χ and $\mathscr{A}$

To define the cost-shares for the instance $\mathscr{I} = (\mathcal{M}, \mathscr{D})$, run the algorithm `TimedGlut` on $\mathscr{I}$. During stage i, if we merge supernodes S and S' with leaders s and s' respectively, then we increment the cost-share of each of $(s, \bar{s})$ and $(s', \bar{s}')$ by $\frac{2^{i+1}}{2\gamma_{TG}}$. The budget balance property follows directly from the analysis of the `TimedGlut` algorithm.

The algorithm $\mathscr{A}$ on input $\mathscr{I} = (\mathcal{M}, \mathscr{D})$ is simple: set the activity time $\mathsf{time}(s) := 6 \cdot 2^{\mathsf{level}(s)+1}$ for each terminal s, and run the algorithm `TimedPD`. The following claim follows from a simple extension of Theorem 5.2 and the fact that $6 \cdot 2^{\mathsf{level}(s)+1} \leq 48 \cdot \frac{1}{2} d_{\mathcal{M}}(s, \bar{s})$.

Lemma 5.3 *The algorithm $\mathscr{A}$ is a 96-approximation algorithm for Steiner forest.*

5.1.2 Proving Strictness

We now give an outline of the proof of the strictness property, leaving details to the full version. Given a set of demands $\mathscr{D}$ in a metric space $\mathcal{M}$, and a partition into $\mathscr{D}_1 \cup \mathscr{D}_2$, we run the algorithm $\mathscr{A}$ on $\mathscr{D}_1$—let F_1 be the forest returned by this algorithm, and let metric $\mathcal{M}_1$ be obtained by contracting the edges of F_1. To prove the strictness property, we now exhibit a "candidate" Steiner forest F_2 for $\mathscr{D}_2$ in the metric $\mathcal{M}_1$ with cost at most a constant factor times $\sum_{(s,\bar{s}) \in \mathscr{D}_2} \chi(\mathcal{M}, \mathscr{D}, (s, \bar{s}))$, the total cost-share assigned to the terminals in $\mathscr{D}_2$.

To define the forest F_2 connecting $\mathscr{D}_2$, we now imagine running `TimedGlut` on the entire demand set $\mathscr{D}_1 \cup \mathscr{D}_2$ on the original metric $\mathcal{M}$, look at paths added by that algorithm, and choose a carefully chosen subset of these paths to add to F_2. This is the natural thing to do, since such a run of `TimedGlut` was used to define the cost-shares χ in the first place. Ideally, whenever this algorithm connects two supernodes S and S', we will add a path between them (in the forest F_2) unless there are terminals v and v' in S and S' respectively such that both v and v' lie in the same tree of F_1 (note that v, v' belong to $\mathscr{D}_1$). However, analyzing such a scheme directly is quite challenging. Instead, we carefully match the runs of the following algorithms: `TimedGlut` on $\mathscr{D}_1 \cup \mathscr{D}_2$, denoted by $\mathcal{R}$, and $\mathscr{A}$ on $\mathscr{D}_1$. For any parameter t, the run of $\mathscr{A}$, which is a moat growing algorithm, at time t corresponds to the run $\mathcal{R}$ when the merging distance is about t. We now show that if $\mathcal{R}$ merges supernodes S and S', then we will add a subset of the edges on the shortest path between S and S' to the forest F_2 such that these edges correspond to the "dual" raised by the corresponding run of $\mathscr{A}$ (and so would have contributed towards the cost share of a terminal). Details of the analysis can be found in the full version.

Acknowledgments

We thank R. Ravi for suggesting the problem to us, for many discussions about it over the years, and for the algorithm name. We also thank Chandra Chekuri, Jochen Könemann, Stefano Leonardi, and Tim Roughgarden for many useful discussions.

6. REFERENCES

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

[2] B. Awerbuch, Y. Azar, and Y. Bartal. On-line generalized Steiner problem. *Theoret. Comput. Sci.*, 324(2-3):313–324, 2004.

[3] P. Berman and C. Coulston. On-line algorithms for Steiner tree problems. In *STOC*, pages 344–353, 1997.

[4] J. Byrka, F. Grandoni, T. Rothvoss, and L. Sanità. Steiner tree approximation via iterative randomized rounding. *J. ACM*, 60(1):Art. 6, 33, 2013.

[5] C. Chekuri and F. B. Shepherd. Approximate integer decompositions for undirected network design problems. *SIAM J. Discrete Math.*, 23(1):163–177, 2008/09.

[6] H.-L. Chen, T. Roughgarden, and G. Valiant. Designing network protocols for good equilibria. *SIAM J. Comput.*, 39(5):1799–1832, 2010.

[7] L. Fleischer, J. Könemann, S. Leonardi, and G. Schäfer. Strict cost sharing schemes for Steiner forest. *SIAM J. Comput.*, 39(8):3616–3632, 2010.

[8] N. Garg, A. Gupta, S. Leonardi, and P. Sankowski. Stochastic analyses for online combinatorial optimization problems. In *SODA*, pages 942–951, 2008.

[9] M. X. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Comput.*, 24(2):296–317, 1995.

[10] A. Gupta and A. Kumar. A constant-factor approximation for stochastic Steiner forest. In *STOC*, pages 659–668, New York, NY, USA, 2009. ACM.

[11] A. Gupta, A. Kumar, M. Pál, and T. Roughgarden. Approximation via cost sharing: simpler and better approximation algorithms for network design. *J. ACM*, 54(3):Art. 11, 38 pp., 2007.

[12] A. Gupta, M. Pál, R. Ravi, and A. Sinha. Sampling and cost-sharing: approximation algorithms for stochastic optimization problems. *SIAM J. Comput.*, 40(5):1361–1401, 2011.

[13] N. Immorlica, D. Karger, M. Minkoff, and V. Mirrokni. On the costs and benefits of procrastination: Approximation algorithms for stochastic combinatorial optimization problems. In *SODA*, pages 684–693, 2004.

[14] K. Jain. A factor 2 approximation algorithm for the generalized Steiner network problem. *Combinatorica*, 21(1):39–60, 2001. (Preliminary version in *39th FOCS*, pages 448–457, 1998).

[15] J. Könemann, S. Leonardi, and G. Schäfer. A group-strategyproof mechanism for Steiner forests. In *SODA*, pages 612–619, 2005.

[16] J. Könemann, S. Leonardi, G. Schäfer, and S. H. M. van Zwam. A group-strategyproof cost sharing mechanism for the Steiner forest game. *SIAM J. Comput.*, 37(5):1319–1341, 2008.

[17] M. Pál. *Cost Sharing and Approximation*. PhD thesis, Cornell University, 2004.

[18] G. Robins and A. Zelikovsky. Tighter bounds for graph Steiner tree approximation. *SIAM J. Discrete Math.*, 19(1):122–134, 2005.