Greedy Algorithms for Steiner Forest

Anupam Gupta\footnote{Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213, USA. Research partly supported by NSF awards CCF-1016799 and CCF-1319811.} \hspace{1cm} Amit Kumar\footnote{Dept. of Computer Science and Engg., IIT Delhi, India 110016.}

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, and Goemans and Williamson gave primal-dual constant-factor approximation algorithms for this problem; until now 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: 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 [AKR95]; this was simplified by Goemans and Williamson [GW95] 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 [CRV10] or Appendix A). 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 \(\pi\) 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 \(\pi\)), 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 \(\pi\), 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 \(\pi\).

**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 worse approximation factor [GK09]. 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 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 [GPRS11] for formal definitions and the relation with cost sharing).

- Consider the online stochastic problem, where given a set of source-sink pairs \(D\) in a metric \(M\), and a probability distribution \(\pi\) over subsets of \(D\) (i.e., over \(2^D\)), an adversary chooses a parameter \(k\), and draws \(k\) times independently from \(\pi\). The on-line algorithm, which can
sample from \( \pi \), 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 \( \pi \) 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. [GGLS08] which works for the special case when \( \pi \) is a distribution over \( \mathcal{D} \) (i.e., singleton subsets of \( \mathcal{D} \)).

### 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 \( \S 2 \).)

The analysis has two conceptual steps. In the first step, we reduce the problem to the special case when the optimal solution can (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, 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. The first step of the above analysis does not hold any more. However, we show that it is enough to account for the merging cost of supernodes when the active terminals in them lie in the same tree of the optimal solution, and consequently the arguments in the second step of the analysis are sufficient. Yet another variant is a **timed** version of the algorithm, which is inspired by a timed version of the primal-dual algorithm [KLSvZ08], and is crucial for obtaining the strict cost-shares described next.

Loosely speaking, a cost-sharing method takes an algorithm \( \mathcal{A} \) and divides the cost incurred by the
algorithm on an instance among the terminals \( D \) in that instance. The “strictness” property ensures that if we partition \( D \) arbitrarily into \( D_1 \cup D_2 \), and build a solution \( A(D_1) \) on \( D_1 \), then the cost-shares of the terminals in \( D_2 \) would suffice to augment the solution \( A(D_1) \) to one for \( D_2 \) as well.

A natural candidate for \( A \) 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 \( D_2 \) 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 \( A \) on \( D_1 \), 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.

![Distances](image)

To avoid this problem, [GKPR07, FKLS10] run the primal-dual algorithm for longer than required, and give results for the case when \( D_2 \) contains a single demand pair. However, the arguments become much more involved than those in the analysis of GW algorithm [GW95]—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 \( D_2 \) is a singleton set) with much simpler analysis, albeit with worse constants than those in [GKPR07, FKLS10]). We refer to this special case of strictness property as uni-strictness.

Our analysis for the general case where \( D_2 \) 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 \( A \) to build as many edges as possible, and the cost share \( \chi \) 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 \( A \). 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 then show that our analysis is flexible enough to analyze several variants of the gluttonous algorithm. We study the the timed version in §4, which gets used in subsequent sections on cost sharing. We also consider the variant of gluttonous based on path-contraction in the appendix (see Appendix B). The cost-sharing method for the uni-strict case is in §5, and the general case is in §5.2.

### 1.2 Related Work

The first constant-factor approximation algorithm for the Steiner forest problem was due to Agrawal, Klein, and Ravi [AKR95] using a primal-dual approach; it was refined and generalized by Goemans and
Williamson [GW95] to a wider class of network design problems. The primal-dual analysis also bounds the integrality gap of the natural LP relaxation (based on covering cuts) by a factor of 2. Different approximation algorithms for Steiner forest based on the same LP, and achieving the same factor of 2, are obtained using the iterative rounding technique of Jain [Jai01], or the integer decomposition techniques of Chekuri and Shepherd [CS09]. A stronger LP relaxation was proposed by Könemann, Leonardi, and Schäfer [KLS05], but it also has an integrality gap of 2 [KLSvZ08].

The special case of 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 [IKMM04], and further studied by [GPRS11], who proposed the boosted-sampling framework of algorithms. The analysis of these algorithms is via “strict” cost sharing methods, which were studied by [GKPR07, FKLS10]. A constant-factor approximation algorithm (with a large constant) was given for stochastic Steiner forest by [GK09] based on primal-dual techniques; it is much more complicated than the algorithm and analysis based on the greedy techniques in this paper.

The stochastic Steiner tree/forest problem was defined by Immorlica, Karger, Minkoff, and Mirrokni [IKMM04], and further studied by [GPRS11], who proposed the boosted-sampling framework of algorithms. The analysis of these algorithms is via “strict” cost sharing methods, which were studied by [GKPR07, FKLS10]. A constant-factor approximation algorithm (with a large constant) was given for stochastic Steiner forest by [GK09] 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 $D \subseteq \binom{V}{2}$ be a collection of source-sink pairs that need to be connected. By splitting vertices, we may assume that the pairs in $D$ are disjoint. A node is a terminal if it belongs to some pair in $D$. Let $K$ denote the number of terminals, and hence there are $2K$ terminals. For a terminal $u$, let $\bar{u}$ be the unique vertex such that $\{u, \bar{u}\} \in D$; we call $\bar{u}$ the mate of $u$.

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

2 The Gluttonous Algorithm

To describe the gluttonous algorithm, we need some definitions. Given a Steiner forest instance $I = (\mathcal{M}, D)$, a supernode is a subset of terminals. A clustering $C = \{S_1, S_2, \ldots, S_q\}$ 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 $C = \{S_1, S_2, \ldots, S_q\}$, define a new metric $\mathcal{M}/C$ called the $C$-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 $u, v$ lie in different supernodes in $C$, and to zero if $u, v$ lie in the same supernode in $C$. Call this graph $G_C$, and defined the $C$-punctured distance to be the shortest-path distance in this graph, denoted by $d_{\mathcal{M}/C}(\cdot, \cdot)$. One can think of this as modifying the metric $\mathcal{M}$ by collapsing the terminals in each of the supernodes in $C$ to a single node. Given clustering $C$ and two supernodes $S_1$ and $S_2$, the distance between them is naturally defined as

$$d_{\mathcal{M}/C}(S_1, S_2) = \min_{u \in S_1, v \in S_2} d_{\mathcal{M}/C}(u, v).$$
The gluttonous algorithm is as follows:

Start with $\mathcal{C}$ being the trivial clustering, and $E'$ being the empty set. While there exist active supernodes in $\mathcal{C}$, do the following:

(i) Find active supernodes $S_1, S_2$ in $\mathcal{C}$ with minimum $\mathcal{C}$-punctured distance. (Break ties arbitrarily but consistently, say choosing the lexicographically smallest pair.)

(ii) Update the clustering to

$$\mathcal{C} \leftarrow (\mathcal{C} \setminus \{S_1, S_2\}) \cup \{S_1 \cup S_2\},$$

(iii) Add to $E'$ the edges corresponding to the inter-supernode edges on the shortest path between $S_1, S_2$ in the graph $G_\mathcal{C}$.

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 $\mathcal{C}$-punctured distance $d_{\mathcal{M}/\mathcal{C}}(S_1, S_2)$, where $\mathcal{C}$ 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 $d_{\mathcal{M}/\mathcal{C}}(S_1, S_2)$. The algorithm maintains the following invariant: if $S$ is a supernode, then the terminals in $S$ lie in the same connected component of $F$.

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 $\mathcal{M}/\mathcal{C}$ that we focus on. The advantage of the latter over the former is that distances in $\mathcal{M}/\mathcal{C}$ are well-controlled (and distances between active terminals only increase over time), whereas those in $\mathcal{M}/E'$ change drastically over time (with distances between active terminals changing unpredictably).

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

Here, we first merge $\{s_1, s_1\}$ 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 $B, s_2, \bar{s}_2$, 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 $\{s_2, s_3, s_2, \bar{s}_2, \bar{s}_3\}$. However, the forest produced will have only a single tree, which consists of the set of edges drawn in the figure.

---

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.
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 §3.2 how to preprocess the optimal forest \( \mathcal{F}^* \) to get a near-optimal forest \( \mathcal{F}^{**} \) 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 \( \mathcal{F}^{**} \).) This makes it easier to then account for the total merging distance, which we do in §3.3.

3.1 Monotonicity Properties

To begin, some simple claims about monotonicity. The first one is by definition.

**Fact 3.1 (Distance Functions are Monotone)** Let the clustering \( C \) be in the same supernode in the gluttonous clustering, then they lie in the same tree in \( F \) of the claim.

3.2 A Near-Optimal Solution with Good Properties

Claim 3.2 Consider clustering \( C \) and let any two active supernodes \( S, T \) be merged, resulting in clustering \( C' \). Then for any active \( U \in C \) that is not \( S \) or \( T \), the distance to its closest active supernode does not decrease. Also, if \( S \cup T \) is active in \( C' \) then the distance to its closest supernode in \( C' \) is at least as large as the minimum of \( S \) and \( T \)'s distances to their closest supernodes in \( C \).

**Proof.** First observe that when two active supernodes are merged, they may stay active or become inactive. An inactive supernode never merges with any other supernode, and hence, it cannot become active later.

For active supernode \( U \neq S, T \), suppose its closest supernode in \( C \) was \( W \) at \( C \)-punctured distance \( L \), and in \( C' \) it is \( W' \) at \( C' \)-punctured distance \( L' \). If \( L' < L \), there must now be a path through supernode \( S \cup T \) that is of length \( L' \). But this means the \( C \)-punctured distance of \( U \) from either \( S \) or \( T \) was at most \( L' \), and both were active in \( C \)—a contradiction. This proves the first part of the claim.

Now, suppose supernode \( U := S \cup T \) is active in \( C' \). Observe that for any other supernode \( W \in C' \), the punctured distance \( d_{C'/\mathcal{E}'}(U, W) = \min\{d_{C'/\mathcal{E}}(S, W), d_{C'/\mathcal{E}}(T, W)\} \). This proves the second part of the claim.

Claim 3.3 (Gluttonous 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 \( S', T' \).

**Proof.** Gluttonous merges two active supernodes with the smallest current distance. By Claim 3.2 distances between the remaining active supernodes do not decrease. This proves this claim.

3.2 A Near-Optimal Solution with Good Properties

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

**Definition 3.4 (Faithful)** A forest \( \mathcal{F} \) is faithful to a clustering \( C \) if each supernode \( S \in C \) is contained within a single tree in \( \mathcal{F} \). (I.e., for all \( S \in C \), there exists \( T \in \mathcal{F} \) such that \( S \subseteq V(T) \).)

Note that every forest is faithful to the trivial clustering consisting of singletons.

**Definition 3.5 (Width)** For a forest \( \mathcal{F} \) that is a solution to instance \( \mathcal{I} \), and for any tree \( T \in \mathcal{F} \), let \( \text{width}(T) \) denote the largest tree distance between any pair connected by \( T \). Let the width of forest \( \mathcal{F} \) be the sum of the widths of the trees in \( \mathcal{F} \). I.e.,

\[
\text{width}(T) := \max\{d_T(u, \bar{u}) \mid \{u, \bar{u}\} \in \mathcal{D}, \{u, \bar{u}\} \subseteq V(T)\},
\]

(3.1)
width(\mathcal{F}) := \sum_{T \in \mathcal{F}} \text{width}(T),
(3.2)

where \text{width}(T) refers to the tree metric induced by T.

We now show there exist near-optimal solutions which are faithful to gluttonous’ final clustering.

**Theorem 3.6 (Low-Cost and Faithful)** Let \( \mathcal{F}^* = \{T^*_1, T^*_2, \ldots, T^*_p\} \) be an optimal solution to the Steiner forest instance \( \mathcal{I} = (\mathcal{M}, \mathcal{G}) \). There exists another solution \( \mathcal{F}^{**} \) for instance \( \mathcal{I} \) such that

(a) \( \text{cost}(\mathcal{F}^{**}) \leq \text{cost}(\mathcal{F}^*) + \text{width}(\mathcal{F}^*) \leq 2\text{cost}(\mathcal{F}^*) \), and

(b) \( \mathcal{F}^{**} \) is faithful to the final clustering \( \mathcal{C}' \) produced by the gluttonous algorithm.

**Proof.** Start with \( \mathcal{F}^{**} = \mathcal{F}^* \) which clearly satisfies the first (cost) guarantee but perhaps not the second (faithfulness) one. To fix this, run the gluttonous algorithm on \( \mathcal{I} \), and whenever it connects two terminals \((u, v)\) that violate the condition (b), connect up some trees in the current \( \mathcal{F}^{**} \) to prevent this violation. In particular, we show how to do this while maintaining two invariants:

(A) The cost of edges in \( \mathcal{F}^{**} \setminus \mathcal{F}^* \) is at most \( \text{width}(\mathcal{F}^*) - \text{width}(\mathcal{F}^{**}) \), and

(B) at any point in time, the forest \( \mathcal{F}^{**} \) is faithful to the current clustering \( \mathcal{C} \) (during the run of the gluttonous algorithm).

At the beginning, the clustering \( \mathcal{C} \) is the trivial clustering consisting of singleton sets containing terminals, and \( \mathcal{F}^{**} = \mathcal{F}^* \); both invariants (A) and (B) are vacuously true.

Now consider some step of gluttonous which starts with the clustering \( \mathcal{C} \) and connects two active supernodes \( S \) and \( S' \) which are closest to each other to get the clustering \( \mathcal{C}' \). By the invariant (B), we know all terminals in supernode \( S \) lie within the same tree in \( \mathcal{F}^{**} \), and the same for terminals in \( S' \). Let \( u \in S \) and \( v \in S' \) be some active terminals within these supernodes; hence \( \bar{u} \notin S \) and \( \bar{v} \notin S' \). Two cases arise:

- **Case I:** \( u \) and \( v \) belong to the same tree in \( \mathcal{F}^{**} \): Clearly, \( \mathcal{F}^{**} \) satisfies the invariant (B) with respect to \( \mathcal{C}' \) as well. Hence, we keep \( \mathcal{F}^{**} \) unchanged and it satisfies invariant (A) trivially.

- **Case II:** \( u \) and \( v \) belong to different trees \( T_1^{**}, T_2^{**} \in \mathcal{F}^{**} \): Suppose the shortest path between \( u \) and \( v \) in \( \mathcal{M}/\mathcal{C} \) is

\[
P = \{u = x_0, x'_0, x_1, x'_1, x_2, x'_2, x_3, \ldots, x'_{k-1}, x_k, x'_k = v\}
\]

such that each \( x_i, x'_i \) belong to the same supernode \( S_i \) in \( \mathcal{C} \) (see e.g., figure 3.4). By the greedy behavior of gluttonous, \( \text{cost}(P) \) is at most the cost to connect \( u \) to \( \bar{u} \), or to connect \( v \) to \( \bar{v} \) in \( \mathcal{M}/\mathcal{C} \). In fact, we can bound these costs by the cost of the edges between \( u, \bar{u} \) in \( T_1^{**} \), etc. Hence,

\[
\text{cost}(P) \leq \min \{d_{T_1^{**}}(u, \bar{u}), d_{T_2^{**}}(v, \bar{v})\} \leq \min \{\text{width}(T_1^{**}), \text{width}(T_2^{**})\}. \quad (3.3)
\]

Since each of the supernodes \( S_i \) is contained within some tree in \( \mathcal{F}^{**} \) (by invariant (B) applied to clustering \( \mathcal{C} \)), we need only add (a subset of edges from) the path \( P \) to the forest \( \mathcal{F}^{**} \) in order to merge \( T_1^{**} \) and \( T_2^{**} \) (and perhaps other trees in \( \mathcal{F}^{**} \)) into one single tree—thus ensuring invariant (B) for the new clustering \( \mathcal{C}' \).

How does the width of the trees in \( \mathcal{F}^{**} \) change? Each tree that we merge is inactive (since it is a coarsening of the original solution \( \mathcal{F}^* \)). Connecting up \( T_1^{**}, \ldots, T_k^{**} \) causes the width of the resulting tree to be \( \max\{\text{width}(T_1^{**}), \ldots, \text{width}(T_k^{**})\} \). The decrease in \( \text{width}(\mathcal{F}^{**}) \) due to the merge is at least \( \min\{\text{width}(T_1^{**}), \text{width}(T_2^{**})\} \), which ensures invariant (A) (using inequality (3.3)).
Hence, at the end of the run of gluttonous, both invariants hold. Since the initial potential is \( \text{width}(\mathcal{F}^*) \leq \text{cost}(\mathcal{F}^*) \), and the final potential is non-negative, the total cost of edges in \( \mathcal{F}^{**} \setminus \mathcal{F}^* \) is at most \( \text{cost}(\mathcal{F}^*) \). This completes the proof.

3.3 Charging to this Near-optimal Solution

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

**Theorem 3.7** If the forest \( \mathcal{F}^* \) is faithful to the final clustering \( \mathcal{C}^f \) of the gluttonous algorithm, then the cost of the gluttonous algorithm is \( O(1) \cdot \text{cost}(\mathcal{F}^*) \).

Since by Theorem 3.6 there is a forest \( \mathcal{F}^* \) with cost at most twice the optimum that is faithful to gluttonous’ final clustering \( \mathcal{C}^f \), applying Theorem 3.7 to this forest proves Theorem 1.1.

We now prove Theorem 3.7. 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 \( \mathcal{F}_t \) that is a solution to the remaining instance. We show that in an amortized sense, at each step the cost of forest \( \mathcal{F}_t \) 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 \( \mathcal{I} \), assume that \( \mathcal{C}^f \) is gluttonous’ final clustering, and \( \mathcal{F}^* \) is faithful to \( \mathcal{C}^f \). Let \( \mathcal{C}^{(t)} \) be the gluttonous clustering at the beginning of the iteration \( t \), with \( \mathcal{C}^{(t)}_{\text{active}} \) being the active supernodes. It will be useful to view this clustering as giving us an induced Steiner forest instance \( \mathcal{I}^{(t)} \) on the metric whose points are the supernodes in \( \mathcal{C}^{(t)} \) and where distances are given by the punctured metric \( d_{M/\mathcal{C}^{(t)}} \), where the terminals in the instance \( \mathcal{I}_t \) are supernodes in \( \mathcal{C}^{(t)}_{\text{active}} \), 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 3.6). For any iteration \( t \), the subsequent run of gluttonous is just a function of this induced instance \( \mathcal{I}_t \). Indeed, given the instance \( \mathcal{I}_t \), gluttonous outputs a final clustering which is same as \( \mathcal{C}^f \) except the inactive supernodes in \( \mathcal{C}^{(t)} \) are absent. i.e., the inactive supernodes in \( \mathcal{C}^{(t)} \) will not play a role, but all the active supernodes will continue to combine in the same way in \( \mathcal{I}_t \) as in \( \mathcal{I} \). We now inductively maintain a forest \( \mathcal{F}^{(t)} \) such that

\[
\begin{align*}
(\text{I1}) \ & \ \mathcal{F}^{(t)} \text{ is a feasible solution to this Steiner forest instance } \mathcal{I}_t, \\
(\text{I2}) \ & \ \mathcal{F}^{(t)} \text{ maintains the connectivity structure of } \mathcal{F}^*, \text{ i.e., if } u \text{ and } v \text{ are two active terminals which are in the same tree in } \mathcal{F}^*, \text{ then the supernodes containing } u \text{ and } v \text{ lie in the same tree in } \mathcal{F}^{(t)}.
\end{align*}
\]

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

**The “candidate” forest** \( \mathcal{F}^{(t)} \). The initial clustering \( \mathcal{C}^{(1)} \) is the trivial clustering consisting of singleton terminals; we set \( \mathcal{F}^{(1)} \) to \( \mathcal{F}^* \). Since \( \mathcal{I}_1 \) is the original instance, \( \mathcal{F}^{(1)} \) is feasible for it; invariant (I2) is satisfied trivially.

\[\text{Figure 3.4: Case II of the proof of Theorem 3.6. The grey blobs are supernodes in } \mathcal{C}, \text{ the solid lines denote the forest } \mathcal{F}^{**}, \text{ the dotted lines are the path } P. \text{ Observe we do not need to add the second edge of } P \text{ as it will create a cycle.}\]
For an iteration \( t \), let \( E(\mathcal{F}^{(t)}) \) denote the edges in \( \mathcal{F}^{(t)} \). Note that an edge \( e \in E(\mathcal{F}^{(t)}) \) between two supernodes \( S_1, S_2 \in \mathcal{C}^{(t)} \) corresponds to an edge between two terminals \( u, v \) in the original metric \( M \), where \( u \in S_1, v \in S_2 \). Define \( \text{length}(e) \) as \( d_M(u, v) \), the length of the edge \( e \) in the original metric. Note that the length of \( e \) in the metric \( M/\mathcal{C}^{(t)} \) may be smaller than \( \text{length}(e) \). For every edge \( e \in \mathcal{F}^{(t)} \), we shall also maintain a potential of \( e \), denoted \( \psi(e) \). Initially, for \( t = 1 \), the potential \( \psi(e) = \text{length}(e) \) for all \( e \in E(\mathcal{F}^{(1)}) \). During the course of the algorithm, the potential \( \psi(e) \geq \text{length}(e) \); we describe the rule for maintaining potentials below. Intuitively, an edge \( e \in \mathcal{F}^{(t)} \) would have been obtained by short-cutting several edges of \( \mathcal{F}^* \), and \( \psi(e) \) is equal to the total length of these edges.

Suppose we have a clustering \( \mathcal{C}^{(t-1)} \) and a forest \( \mathcal{F}^{(t-1)} \) which satisfies invariants (I1) and (I2). If we now merge two supernodes \( S_1, S_2 \in \mathcal{C}^{(t-1)} \) to get clustering \( \mathcal{C}^{(t)} \), we have to update the forest \( \mathcal{F}^{(t-1)} \) to get to \( \mathcal{F}^{(t)} \) using procedure \textbf{UpdateForest} given in Figure 3.3. The main idea is simple: when we merge the nodes corresponding to \( S_1 \) and \( S_2 \) in \( \mathcal{F}^{(t-1)} \) 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 \( \mathcal{F}^{(t)} \) 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 \( \mathcal{F}^{(t)} \), we short-cut the two incident edges.

\begin{algorithm}
\textbf{Algorithm \textbf{UpdateForest} (} \mathcal{C}^{(t-1)}, S_1, S_2 \text{)} :
\begin{enumerate}
\item Let \( T \) be the tree in \( \mathcal{F}^{(t-1)} \) containing the terminals in \( S_1 \) and \( S_2 \).
\item Merge \( S_1 \) and \( S_2 \) to a single node \( S \) in the tree \( T \).
\item If the new supernode \( S \) becomes inactive, and has degree 2 in the tree \( T \), then short-cut the two edges incident to \( S \).
\item Let \( C \) denote the unique cycle formed in the tree \( T \).
\item Delete the edge in the cycle \( C \) which has the highest potential.
\item While there is an inactive supernode in \( T \) which is a degree-2 vertex, short-cut the two incident edges to this vertex.
\end{enumerate}
\end{algorithm}

Figure 3.5: The procedure for updating \( \mathcal{F}^{(t-1)} \) to \( \mathcal{F}^{(t)} \).

Some more comments about the procedure \textbf{UpdateForest}. In Step 1, the existence of the tree \( T \) follows from the invariant property (I2) and the faithfulness of \( \mathcal{F}^* \) to \( \mathcal{C}^t \). Since the terminals in \( S_1 \cup S_2 \) are in the same tree in \( \mathcal{F}^* \), the invariant means they belong to the same tree in \( \mathcal{F}^{(t-1)} \), and the construction ensures they remain in the same tree in \( \mathcal{F}^{(t)} \). 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 \( \mathcal{F}^{(t)} \) is a feasible solution to the instance \( \mathcal{I}_t \). Indeed, the only difference between \( \mathcal{I}_{t-1} \) and \( \mathcal{I}_t \) 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 \( \mathcal{I}_{t-1} \). It is also easy to check that the invariant property (I2) continues to hold. The following claim proves some more crucial properties of the forest \( \mathcal{F}^{(t)} \).

\textbf{Claim 3.8} For all iterations \( t \), the Steiner nodes in \( \mathcal{F}^{(t)} \) have degree at least 3. Therefore, there are at most 2 iterations of the while loop in Step 6 of the \textbf{UpdateForest} algorithm.

\textbf{Proof.} We prove the first statement of the lemma by induction on \( t \). For \( t = 1 \), it holds by construction: we can assume that \( \mathcal{F}^* \) has no Steiner vertex of degree at most 2: any leaf Steiner node can be deleted, and a degree 2 can be removed by short-cutting the incident edges. Suppose this property is true for \( \mathcal{F}^{(t-1)} \). We merge \( S' \) and \( S'' \), and if the new supernode \( S \) becomes an inactive supernode,
then its degree will be at least 2 (both $S'$ and $S''$ must have had degree at least 1). If the degree is equal to 2, we remove this vertex in Step 3.

When we remove an edge in Step 5, the two end-points could have been Steiner vertices. By the induction hypothesis, their degree will be at least 2 (after the edge removal). If their degree is 2, we will again remove them by short-cutting edges. Note that this will not affect the degree of other nodes in the forest. This also shows that Step 6 will be carried out at most twice.

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

Next we show that the total potential of the edges does not change over time. Let $\text{del}(t)$ 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 $\text{del}(t)$ does not include edges that are short-cut.)

**Lemma 3.9** For iteration $t$, the sum of potentials of edges $\text{del}(t)$ and $E(T^t)$ equals $\text{cost}(T^*)$. Further, $\psi(e) \geq \text{length}(e)$ for all edges $e \in E(T^t) \cup \text{del}(t)$.

**Proof.** By induction on $t$. The base case $t = 1$ follows by construction. For the IH, assume that $t$ is a relevant iteration (else $T^t = T^{t-1}$): if we remove edge $e$ from $T^{t-1}$ during Step 5, we do not change $\psi(e)$. If we short-cut two edges $e', e''$ to an edge $e$, $\psi(e) = \psi(e') + \psi(e'')$. Therefore the total potential of the edges in the tree plus that of the edges in $\text{del}(t)$ does not change. Further, $\text{length}(e) \leq \text{length}(e') + \text{length}(e'') \leq \psi(e') + \psi(e'') = \psi(e)$.

Eventually $T^t$ has no active supernodes (for large $t$) and hence all its edges are deleted. Hence if $\text{del}(\infty)$ denotes the edges deleted during all the relevant iterations in gluttonous, Lemma 3.9 implies $\sum_{e \in \text{del}(\infty)} \psi(e) = \text{cost}(T^*)$. Let $\Delta_t$ 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 $\text{del}(\infty)$. Formally, let $N_t$ denote the number of active supernodes in $T^t$, at the beginning of iteration $t$.

**Theorem 3.10** If $t_0$ is relevant, there are at least $N_{t_0}/8$ edges in $\text{del}(\infty)$ of potential at least $\Delta_{t_0}/6$.

We defer the proof of Theorem 3.10 for the moment, and instead show how to use this to charge the merging costs and to prove Theorem 3.7, which in turn gives the main theorem of the paper.

**Proof of Theorem 3.7:** Let $I'$ denote the index set of all relevant iterations during the run of gluttonous. We now define a mapping $g$ from $I'$ to $\text{del}(\infty)$ such that: (i) for any edge $e \in \text{del}(\infty)$, the pre-image $g^{-1}(e)$ has cardinality at most 8, and (ii) the potential $\psi(g(t)) \geq \Delta_t/6$ for all $t \in I'$. To get this, consider a bipartite graph on vertices $I' \cup \text{del}(\infty)$ where a iteration $t \in I'$ is connected to all edges $e \in \text{del}(\infty)$ for which $\psi(e) \geq \Delta_t/6$. Theorem 3.10 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

$$\sum_{t \in I'} \Delta_t = \sum_{e \in \text{del}(\infty)} \sum_{t \in g^{-1}(e)} \Delta_t \leq 48 \sum_{e \in \text{del}(\infty)} \psi(e) = 48 \text{cost}(T^*),$$

10
where the last equality follows from Lemma 3.9. By the faithfulness property, each iteration of 
gluttonous is relevant with respect to one of the trees in \( \mathcal{F}^* \), so summing the above expression over all trees gives the total merging cost to be at most \( 48 \text{cost}(\mathcal{F}^*) \).

Combining Theorem 3.7 with Theorem 3.6 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.

3.3.1 Proof of Theorem 3.10

In order to prove Theorem 3.10, we need to understand the structure of the trees \( T(t) \) for \( t \geq t_0 \) in more detail. Let \( \text{del}_0([t_0 \ldots t]) \) denote the edges deleted during the relevant iterations in \( t_0, \ldots, t - 1 \), i.e., \( \text{del}([t_0 \ldots t]) := \text{del}(t) \setminus \text{del}(t_0) \). 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 \( \mathcal{E}(t) \) of the edge set \( E(T(t_0)) \), such that there is a correspondence between edges \( e \in T(t_0) \cup \text{del}([t_0 \ldots t]) \) and sets \( D_t(e) \in \mathcal{E}(t) \), 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 \( \text{head}(D_t(e)) \) 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 \( e \in \text{del}([t_0 \ldots t]) \) for all \( t' > t \), and the set \( D_{t'}(e) = D_t(e) \) for all future partitions \( \mathcal{E}(t') \).

Lemma 3.11 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 \( \ell \) denote the number of long edges in the tree \( T(t_0) \). Deleting these edges from \( T(t_0) \) gives \( \ell + 1 \) subtrees \( C_1, C_2, \ldots, C_{\ell+1} \). 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 \( \Delta_{t_0} \) 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 \( \ell \) 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 \( 3N_{t_0} - 2\ell \leq 2N_{t_0} \) or \( \ell \geq N_{t_0}/2 \).

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

- At least \( N_{t_0}/8 \) edges in \( L_0 \) are \( \text{head}(D_{\infty}(e)) \) for some set \( D_{\infty}(e) \in \mathcal{E}(\infty) \). Since each set in \( \mathcal{E}(\infty) \) has only one head, there are \( N_{t_0}/8 \) such sets. In any such set \( D_{\infty}(e) \), \( \psi(e) \geq \text{length}(\text{head}(D_{\infty}(e))) \geq \Delta_{t_0}/6 \). Moreover, we must have removed \( e \) in some iteration between \( t_0 \) and the end, and hence \( e \in \text{del}([t_0 \ldots \infty]) \subseteq \text{del}(\infty) \).

- More than \( 3N_\ell/8 \) edges in \( L_0 \) are not heads of any set in \( \mathcal{E}(\infty) \). Take one such edge \( e_0 \) — the sets in \( \mathcal{E}(t_0) \) 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 \( \mathcal{E}(t) \), and suppose \( e_0 = \text{head}(D_{t-1}(e')) \) for some set \( D_{t-1}(e') \in \mathcal{E}(t - 1) \). In forming \( \mathcal{F}(t) \), we must have short-cut \( e' \) and some other edge \( e'' \) to form an edge \( e \in \mathcal{F}(t) \). Observe that \( \text{length}(\text{head}(D(e''))) \geq \text{length}(e_0) \).
else \( e_0 \) would continue to be the head of \( D(e) \). Moreover,

\[
\min (\psi(e'), \psi(e'')) \geq \min (\text{length}(\text{head}(e')), \text{length}(\text{head}(e''))) \geq \Delta_{t_0}/6.
\]

By the discussion in Claim 3.8, one of \( e' \) and \( e'' \) must lie on the cycle formed when we merged two supernodes in \( F(t-1) \), as in Step 4 of \texttt{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 \( e_t \in \text{del}(\infty) \) (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 \( \text{del}(\infty) \) of potential \( \Delta_{t_0}/6 \), which completes the proof of Theorem 3.10.

\[ \square \]

### 3.4 An Extension of the Analysis in Section 3

Let us now abstract out some properties used in the above analysis, so that we can generalize the analysis to a broader class of algorithms for Steiner forest. This abstraction is used to show that variants of the above algorithm, which are presented in Section 4 and in Appendix B, are also \( O(1) \)-approximations.

Consider an algorithm \( \mathcal{A} \) which maintains a set of supernodes, where a supernode corresponds to a set of terminals, and two different supernodes correspond to disjoint terminals. Initially, we have one supernode for each terminal. Further, a supernode could be active or inactive. Once a supernode becomes inactive, it stays inactive. Now, at each iteration, the algorithm picks two active supernodes, and replaces them by a new supernode which is the union of the terminals in these two supernodes (the new supernode could be active or inactive). Note that the iteration when a supernode becomes inactive is arbitrary (depending on the algorithm \( \mathcal{A} \)).

As in the case of gluttonous algorithm, let \( C_f \) be the final clustering produced by the algorithm \( \mathcal{A} \), and \( T^* \) be a tree solution to a Steiner forest instance \((\mathcal{I}, \mathcal{M})\). Let \( C(t) \) be the set of supernodes at the beginning of iteration \( t \) of \( \mathcal{A} \). For an iteration \( t \), let \( \delta_t \) be the minimum distance (in the metric \( \mathcal{M}/C(t) \)) between any two active supernodes in \( C(t) \). Claim 3.2 gives the following fact.

**Fact 3.12** The quantity \( \delta_t \) forms an ascending sequence with respect to \( t \).

Now Theorem 3.7 generalizes to the following stronger result.

**Corollary 3.13** For any tree solution \( T^* \) to an instance \( \mathcal{I} \), \( \sum_t \delta_t \leq 48 \cdot \text{cost}(T^*) \).

An important remark: this corollary is not making any claim about the merging cost of \( \mathcal{A} \); at any iteration \( \mathcal{A} \) could be connecting two active supernodes which are much farther apart than \( \delta_t \).

### 4 A Timed Greedy Algorithm

We now give a version of the gluttonous algorithm \texttt{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.

\[^2\]Timed versions of the primal-dual algorithm for Steiner forest had been considered previously in [GKPR07, KLS05]; our version will be analogous to that of K"onemann et al. [KLS05] which were used to get cross-monotonic cost-shares for Steiner forest.
The algorithm TimedGlut is very similar to the gluttonous algorithm except for what constitutes an active supernode.

We will again maintain a clustering of terminals (into supernodes) -- let \( C^{(t)} \) be the clustering at the beginning of iteration \( t \). Initially, at iteration \( t = 1 \), \( C^{(1)} \) is the trivial clustering (consisting of singleton sets of terminals). We maintain a set of edges \( E' \) will be the set of edges bought by the algorithm. Initially, \( E' = \emptyset \).

We shall use \( \Delta_t \) to denote the closest distance (in the metric \( M/\mathcal{C}^{(t)} \)) between two active supernodes in \( \mathcal{C}^{(t)} \). Our algorithm will only merge active supernodes, and an inactive supernode will not become active in future iterations. It follows that \( \Delta_t \) cannot decrease with \( t \) (Fact 3.1). This allows us to divide the execution of the algorithm into stages. Stage \( i \) consists of those iterations \( t \) for which \( \Delta_t \) lies in the range \([2^i, 2^{i+1})\) (the initial stage belongs to stage 0, because we can assume w.l.o.g. that the minimum distance between the terminals is 1).

For a terminal \( s \), define

\[
\text{level}(s) := \lceil \log_2 d_M(s, \bar{s}) \rceil. \tag{4.4}
\]

Note that distances in this definition are measured in the original metric \( M \). For a supernode \( S \), define its leader as the terminal in \( S \) whose distance to its mate is the largest (and hence has the largest level); in case of ties, choose the terminal with the smallest index among these.

We shall use \( \mathcal{C}^i \) to denote the clustering at the beginning of stage \( i \) (note the change in notation with respect to \( \mathcal{C}^{(t)} \) which will be denoted by \( \mathcal{C}(t) \)). So, if \( t_i \) denotes the first iteration of stage \( i \), then \( \mathcal{C}^i \) is same as \( \mathcal{C}^{(t_i)} \). Now we specify when a supernode becomes inactive. A terminal \( s \) is active at the beginning of stage \( i \) if \( \text{level}(s) \geq i \). A supernode \( S \) will be active at the beginning of a stage \( i \) if \( \text{level}(\text{leader}(S)) \geq i \). Observe that supernodes do not become inactive during a stage -- if a terminal is active at the beginning of a stage, it remains active during each of the iterations in this stage.

By the definition of a stage, the algorithm will satisfy the invariant that the distance between any two active supernodes in \( \mathcal{C}^i \) (in the metric \( M/\mathcal{C}^i \)) is at least \( 2^i \). During stage \( i \), the algorithm repeatedly performs the following steps in each iteration \( t \): pick any two arbitrary pair of active supernodes \( S', S'' \) which are at most \( 2^{i+1} \) apart (in the metric \( M/\mathcal{C}^{(t)} \)). Further, we take any such \( S' - S'' \) path of length at most \( 2^{i+1} \) (in the graph induced by the metric \( M/\mathcal{C}^{(t)} \) on the vertex set \( \mathcal{C}^{(t)} \)) and add the edges (which go between supernodes) to \( E' \).

Stage \( i \) ends when the merging distance between all remaining active supernodes is at least \( 2^{i+1} \). Observe that when the algorithm stops, we have a feasible solution—indeed, each terminal \( s \) will merge with its mate \( \bar{s} \) by the end of stage \( \text{level}(s) \). At the end, output a maximal acyclic subgraph of \( E' \).

The analysis of TimedGlut goes along the same lines as that of the gluttonous algorithm. The analog of Theorem 3.6 is as follows:

**Theorem 4.1** Let \( \mathcal{F}^* = \{T_1^*, T_2^*, \ldots, T_p^*\} \) be an optimal solution to the Steiner forest instance \( I = (M, D) \). Let clustering \( \mathcal{C}^I \) be produced by some run of the TimedGlut algorithm. There exists another solution \( \mathcal{F}^{**} \) for instance \( I \) such that

(a) \( \text{cost}(\mathcal{F}^{**}) \leq \text{cost}(\mathcal{F}^*) + 4 \cdot \text{width}(\mathcal{F}^*) \leq 5 \cdot \text{cost}(\mathcal{F}^*) \), and

(b) \( \mathcal{F}^{**} \) is faithful to the clustering \( \mathcal{C}^I \).

**Proof.** The proof is very similar to that of Theorem 3.6, where we look over the run of TimedGlut again to alter \( \mathcal{F}^* \) into \( \mathcal{F}^{**} \). Since TimedGlut makes some arbitrary choices, we make the same choices consistently in this proof. We ensure very similar invariants:
(A) The cost of edges in $\mathcal{F}^{**} \setminus \mathcal{F}^*$ is at most $4(\text{width}(\mathcal{F}^*) - \text{width}(\mathcal{F}^{**}))$, and

(B) at any point in time, the forest $\mathcal{F}^{**}$ is faithful to the current clustering $\mathcal{C}$.

Observe the extra factor of 4 in invariant (A). Again, let two active supernodes $S'$ and $S''$ be merged in some stage $i$, and let $u$ and $v$ be the leaders of these supernodes respectively. The argument in Case I remains unchanged. In Case II, let $T_1^{**}$ and $T_2^{**}$ be the trees containing $u$ and $v$ respectively. Being in stage $i$, we know that $\text{level}(u), \text{level}(v) \geq i$, since they are both still active, and that the distance between $S'$ and $S''$ in the current metric is at most $2^{i+1}$, since all merging costs in stage $i$ lie between $2^i$ and $2^{i+1}$. So the cost of connecting $T_1^{**}$ and $T_2^{**}$ is at most

$$2^{i+1} \leq \min(2^{\text{level}(u)+1}, 2^{\text{level}(v)+1}) \leq 4 \cdot \min(d_{T_1^{**}}(u, \bar{u}), d_{T_2^{**}}(v, \bar{v})) \leq 4 \cdot \min(\text{width}(T_1^{**}), \text{width}(T_2^{**})).$$

The rest of the argument remains unchanged. ■

**Theorem 4.2** The TimedGlut algorithm is a $\gamma_{\text{TGlut}}$-approximation algorithm for Steiner forest, where $\gamma_{\text{TGlut}} = 96 \times 5 = 480$.

**Proof.** Consider a solution $\mathcal{F}^{**}$ which is faithful with respect to the final clustering produced by the TimedGlut algorithm. Suppose there are $m_i$ iterations during stage $i$. Then the total merging cost of the algorithm is at most $\sum_i 2^{i+1} \cdot m_i$.

We would like to use Corollary 3.13. Let $\mathcal{F}^{**}$ consist of the trees $\mathcal{T}^{**}_1, \ldots, \mathcal{T}^{**}_k$. For a tree $\mathcal{T}^{**}_i$, and a stage $i$, let $I_{i,r}$ denote the iterations when we merge two supernodes with terminals belonging to the tree $V(\mathcal{T}^{**}_i)$ (note that the faithfulness property implies that there will be such a tree for each iteration of the algorithm). Let $m_{i,r}$ denote the cardinality of $I_{i,r}$. Clearly, $\sum_r m_{i,r} = m_i$. For an iteration $t$ and index $r$, let $\mathcal{C}^{(t)}$ denote the supernodes in $\mathcal{C}^{(t)}$ with terminals belonging to $V(T_r^*)$. Define $\delta_{t,r}$ as the closest distance (in the metric $\mathcal{M}/\mathcal{C}^{(t)}$) between any two active supernodes with terminals belonging to $V(T_r^*)$. If the iteration belongs to stage $i$, then $\delta_{t,r} \geq 2^i$. Using Corollary 3.13, we get

$$\sum_i 2^{i+1} \cdot m_i \leq 2 \cdot \sum_r \sum_i 2^i \cdot m_{i,r} \leq 98 \cdot \sum_r \text{cost}(T_r^*).$$

The result now follows from Theorem 4.1. ■

**4.1 An Equivalent Description of TimedGlut**

An essentially equivalent way to state the TimedGlut algorithm is as follows. For a stage $i$, let $\mathcal{M}_i$ denote the metric $\mathcal{M}/\mathcal{C}^i$ corresponding to the clustering at the beginning of stage $i$. Construct an auxiliary graph $H^{(i)}$ with vertex set being the set of supernodes in $\mathcal{C}^i$, and edges between two vertices if the two corresponding supernodes are active and the distance between them is at most $2^{i+1}$ in the metric $\mathcal{M}_i$. Pick a maximal acyclic set of edges $\mathcal{P}^{(i)}$ in this auxiliary graph $H^{(i)}$.

- For each edge $(S_1, S_2) \in \mathcal{P}^{(i)}$, merge the supernodes $S_1, S_2$. Hence the clustering $\mathcal{C}^{i+1}$ at the end of stage $i$ is obtained by merging together all the supernodes that fall within a connected component of the subgraph $(H^{(i)}, \mathcal{P}^{(i)})$.
- For each edge $(S_1, S_2) \in \mathcal{P}^{(i)}$, add edges corresponding to a path of length $< 2^{i+1}$ in $\mathcal{M}_i$ to a set of edges $E_i$.

Finally, output a maximal sub-forest of the edges $\cup_i E_i$ added during this process.

One can now check that this algorithm is equivalent to the TimedGlut algorithm as described above; the key observation is that because of the definition of the timed algorithm, an active terminal in stage $i$ stays active throughout the stage, and does not become inactive partway through it. More formally, we have the following observation.
Fact 4.3 Consider an execution of the TimedGlut algorithm on an input $\mathcal{I}$. Then one can define graphs $H^{(i)}$ and $P^{(i)}$ for each stage $i$ such that the set of supernodes at the beginning of stage $i$ in the above algorithm is same as that of the TimedGlut algorithm. Further, the two algorithms pick the same set of edges in each stage.

5 Cost Shares for Steiner Forest

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

$$\sum_{(s, \bar{s}) \in \mathcal{D}} \chi(\mathcal{M}, \mathcal{D}, (s, \bar{s})) \leq \text{cost}(F^*). \quad (5.5)$$

We will consider strict cost-shares; these are useful for several problems in network design (see details in the introduction). There are two versions of strictness: uni-strictness, and strictness. Uni-strict cost-shares for Steiner forest were given by [GKPR07, FKLS10], whereas strict cost shares for Steiner forest have remained an open problem. We show how to get both using the TimedGlut algorithm.

5.1 Uni-strict Cost Shares for Steiner Forest

Definition 5.1 Given an $\alpha$-approximation algorithm $\mathcal{A}$ for the Steiner forest problem, a cost sharing $\chi$ is called $\beta$-uni-strict with respect to $\mathcal{A}$ if for all demand pair $(s, \bar{s})$, the cost share $\chi(\mathcal{M}, \mathcal{D}, (s, \bar{s}))$ is at least $1/\beta$ times the distance between $s$ and $\bar{s}$ in the graph $G/F$, where $F$ is the forest returned by algorithm $\mathcal{A}$ on the input $(\mathcal{M}, \mathcal{D} - \{s, \bar{s}\})$.

Our objective is to find an algorithm $\mathcal{A}$ and the associated cost share $\chi$ such that the parameters $\alpha$ and $\beta$ are both constants.

5.1.1 Defining $\chi$ and $\mathcal{A}$

Let the constant $\gamma_{\text{TG}}$ denote the approximation ratio of the algorithm TimedGlut. The cost-sharing method is simple: for a terminal $s$, let $\ell_s$ be the largest value such that $s$ is a leader in stage $\ell_s$ and its supernode is merged with some other supernode during this stage (note that a supernode can go from being active in the beginning of a stage to becoming inactive in the next stage without merging with any supernode; this can happen because all terminals in it become inactive in the next stage). Then

$$\chi(\mathcal{M}, \mathcal{D}, (s, \bar{s})) := \frac{2^{\ell_s} + 2^{\ell_s}}{2\gamma_{\text{TG}}}. \quad (5.6)$$

The algorithm $\mathcal{A}$ is a slight variant on the TimedGlut algorithm. Given an instance $(\mathcal{M}, \mathcal{D})$, run the algorithm TimedGlut on this instance to get forest $F$. Now merge some of the trees in $F$ as follows. Recall that the width of each tree $T$ in $F$ is defined to be $\text{width}(T) := \max_{(s, \bar{s}) \in T} d_T(s, \bar{s})$, where $d_T(s, \bar{s})$ denotes the distance between $s$ and $\bar{s}$ in the tree $T$. While there are trees $T_1, T_2 \in F$ such that $d_{M/F}(T_1, T_2) \leq 5 \min(\text{width}(T_1), \text{width}(T_2))$, connect $T_1, T_2$ by a path of length $d_{M/F}(T_1, T_2)$ to get a tree $T$, and update $F \leftarrow (F \setminus \{T_1, T_2\}) \cup \{T\}$. Here, $d_{M/F}(T_1, T_2)$ denotes the minimum over all pairs $u \in T_1, v \in T_2$, of $d_{M/F}(u, v)$.
5.1.2 Analysis

We now prove that the cost sharing method $\chi$ is $\beta$-uni-strict with respect to $A$, where $\beta$ is a constant. Recall that $F$ denotes the forest returned by the algorithm $A$.

To begin, observe that the algorithm $A$ is also a constant-factor approximation.

**Lemma 5.2** The algorithm $A$ is an $6\gamma_{TG}$-approximation for Steiner forest.

**Proof.** Let $F'$ be the forest returned by the TimedGlut algorithm (called by $A$). Consider the potential $\sum_{T \in F'} (c(T) + 5 \text{width}(T))$. Since the width of each tree is at most the cost of its edges, and since TimedGlut was a $\gamma_{TG}$-approximation, this potential is at most $6\gamma_{TG}$ times the optimal cost. Now, observe that whenever $A$ merges two trees of this forest, the potential of the new forest does not increase. Therefore, the potential of the forest $F$ is also at most $6\gamma_{TG}$ times the optimal cost. ■

**Lemma 5.3** The function $\chi$ is a budget-balanced cost sharing method.

**Proof.** We need to prove the inequality (5.5). To do this, let us run TimedGlut and “charge” the cost of merging two active supernodes to the leaders of the respective clusters—charge half of the distance between these two supernodes to the leaders of each of these supernodes. Clearly, the total charge assigned to the terminals is equal to the total cost paid by the algorithm TimedGlut, which at most $\gamma_{TG} \text{cost}(\mathcal{F}^*)$. Finally, we make the observation that each terminal $s$ is charged at least $\gamma_{TG}$ times the cost share (since it is charged at least $2^\ell_s / 2$ in stage $\ell_s$) to complete the proof. ■

To prove the uni-strictness property, fix a terminal pair $(s, \bar{s})$, and consider two instances: $\mathcal{I} = (\mathcal{M}, \mathcal{D})$ and $\mathcal{I}' = (\mathcal{M}, \mathcal{D} - \{(s, \bar{s})\})$. For instance $\mathcal{I}$, let $\mathcal{C}^i$ denote the set of supernodes at the beginning of stage $i$; let $\mathcal{C}'^i$ be the corresponding set for $\mathcal{I}'$. Let $\mathcal{M}_i$ and $\mathcal{M}'_i$ denote the metrics $\mathcal{M}/\mathcal{C}^i$ and $\mathcal{M}/\mathcal{C}'^i$ respectively. Recall that level$(s) = \lceil \log_2 d_M(s, \bar{s}) \rceil$.

The following claim will be convenient to understand the behavior of TimedGlut.

**Lemma 5.4** Consider stage $i$ in the execution TimedGlut on the instance $\mathcal{I}$. Define a graph $\mathcal{G}_i$ on the vertex set $\mathcal{C}^i$, with an edge between two active supernodes $C_1, C_2 \in \mathcal{C}^i$ if there is a path of length at most $2^{i+1}$ between them in $\mathcal{M}_i$ that does not contain any other active supernode as an internal node. If the connected components of $\mathcal{G}_i$ are $H_1, \ldots, H_q$, then $\mathcal{C}^{i+1}$ has $q$ supernodes—one supernode for each $H_j$ (formed by merging the supernodes in $H_j$).

**Proof.** The statement is essentially the same as Fact 4.3 except that in the graph $H^{(i)}$ (defined in Section 4.1), we join two active supernodes $C_1, C_2 \in \mathcal{C}^i$ by an edge if the distance between them in the metric $\mathcal{M}_i$ is at most $2^{i+1}$, whereas here in the graph $\mathcal{G}_i$, we wish to have a path of length at most $2^{i+1}$ with no internal vertex being an active supernode. We claim that the connected components in the two graphs are the same, and hence, the statement in the lemma follows.

Clearly, an edge $e \in \mathcal{G}_i$ is present in $H^{(i)}$ as well. Now, consider an edge $(C_1, C_2)$ in $H^{(i)}$. Let $P$ be the shortest path of length at most $2^{i+1}$ between $S_1$ and $S_2$ in the metric $\mathcal{M}_i$. Let the active supernodes on this path be $C_1 = C_{i_1}, C_{i_2}, \ldots, C_{i_p} = C_2$ (in this order). Then $\mathcal{G}_i$ has edges $(C_{i_r}, C_{i_{r+1}})$ for $r = 1, \ldots, p - 1$. Therefore $C_1$ and $C_2$ are in the same connected component of $\mathcal{G}_i$. This proves the desired claim. ■

**Theorem 5.5 (Nesting)** For $i \leq \text{level}(s)$, let $C_s$ and $C_{\bar{s}}$ be the supernodes in $\mathcal{C}^i$ containing $s$ and $\bar{s}$ respectively. The following hold:

1. If $C_s \neq C_{\bar{s}}$, we can arrange the supernodes in $\mathcal{C}^i$ as $C'_1, \ldots, C'_p$ such that $C_s = \{s\} \cup C'_1 \cup \ldots \cup C'_a$, $C_{{\bar{s}}} = \{\bar{s}\} \cup C'_{a+1} \cup \ldots \cup C'_b$ for some $0 \leq a \leq b \leq p$. Moreover, $\mathcal{C}^i - \{C_s, C_{\bar{s}}\} = \{C'_{b+1}, \ldots, C'_p\}$. If $C_s = C_{\bar{s}}$, we can arrange the supernodes in $\mathcal{C}^i$ as $C'_1, \ldots, C'_p$ such that $C_s = \{s, \bar{s}\} \cup C'_1 \cup \ldots \cup C'_b$, for some $0 \leq b \leq p$. Also, $\mathcal{C}^i - \{C_s\} = \{C'_{b+1}, \ldots, C'_p\}$.
(b) Suppose $C_s$ and $C_{\bar{s}}$ are distinct supernodes. Then for any terminal $v \in C_s$, $d_{M'_i}(s,v) \leq 2 \cdot 2^i$. Similarly, for any $v \in C_{\bar{s}}$, $d_{M'_i}(s,v) \leq 2 \cdot 2^i$.

(c) Suppose $C_s = C_{\bar{s}}$. Then, $d_{M'_i}(s,\bar{s}) \leq 4 \cdot 2^i$.

**Proof.** We induct on $i$. At the beginning, all clusters are singletons, so the base case is easy. For the inductive step, suppose the statement of the theorem is true for some $i < \text{level}(s)$. Assume that $C_s \neq C_{\bar{s}}$, the other case is similar. Apply Lemma 5.4 to stage $i$ in both $\mathcal{I}$ and $\mathcal{I}'$, and let $\mathcal{G}_i$ and $\mathcal{G}'_i$ be the corresponding graphs on the vertex sets $C$ and $\mathcal{C}^i$ (as defined in Lemma 5.4). We know that the supernodes of $\mathcal{C}^{i+1}$ and $\mathcal{C}'^{i+1}$ correspond to the connected components of these graphs; we now use this information to prove the induction step.

By the induction hypothesis, the supernodes in $\mathcal{C}^i$ can be labeled $C_s, C_{\bar{s}}, C'_{b+1}, \ldots, C'_{p}$; moreover, we can define a map $\phi : V(\mathcal{G}'_i) \rightarrow V(\mathcal{G}_i)$ as follows:

$$
\phi(C'_j) := \begin{cases} 
C_s & 0 \leq j \leq a \\
C_{\bar{s}} & a + 1 \leq j \leq b \\
C'_j & b + 1 \leq j \leq p 
\end{cases} \tag{5.7}
$$

Suppose there is an edge between $C'_j$ and $C'_k$ in $\mathcal{G}'_i$. By the definition of $\mathcal{G}_i$, both are active supernodes, and the length of the shortest path between them in the metric $M'_i$ is at most $2^{i+1}$. This path has no greater length in the metric $M_i$, since the supernodes in $\mathcal{C}^i$ are unions of supernodes in $\mathcal{C}^{i+1}$. This means there is a path between $\phi(C'_j)$ and $\phi(C'_k)$ in $\mathcal{G}_i$, i.e., the clustering $\mathcal{C}'^{i+1}$ is a refinement of $\mathcal{C}^{i+1}$ (because these clusterings are determined by the connected components of the corresponding graphs).

Now consider an edge $e$ in $\mathcal{G}_i$. For the first part of the theorem, suppose $e = (C'_j, C'_k)$ where both $j, k \geq b + 1$. If $P$ is the corresponding path in $M_i$ between these two supernodes, then $P$ cannot contain $C_s$ or $C_{\bar{s}}$ as an internal node (because it does not contain any active supernodes as internal nodes, and both $C_s, C_{\bar{s}}$ are active). But then the length of $P$ remains unchanged in $M'_i$, and we have the corresponding edge $(C'_j, C'_k)$ in $\mathcal{G}'_i$ as well. This means that all the connected components of $\mathcal{G}_i$ not containing $C_s$ or $C_{\bar{s}}$ also form connected components in $\mathcal{G}'_i$. Combined with the fact that $\mathcal{C}'^{i+1}$ is a refinement of $\mathcal{C}^{i+1}$, this proves the part (a) of the theorem for the case $C_s \neq C_{\bar{s}}$. (The proof for the other case is similar.)

For part (b), let $H_1$ be the connected component of $\mathcal{G}_i$ which contains the supernode $C_s$ (as a vertex). So all the supernodes in $H_1$ will merge to form a single supernode of $\mathcal{C}^{i+1}$. As argued in the paragraph above, any edge in $H_1$ which is not incident with $C_s$ is also present in $\mathcal{G}'_i$ (recall that we are assuming $C_s$ is not one of the vertices in $H_1$). Let $v$ be a terminal in a supernode $B$ in $H_1$. Let the path from $C_s$ to $B$ in $H_1$ be $C_s = A_0, A_1, \ldots, A_r = B$. Since the edges $(A_1, A_2), \ldots, (A_{r-1}, A_r)$ belong to $\mathcal{G}_i$ as well, $A_1, \ldots, A_r$ will lie in the same supernode in $\mathcal{C}^{i+1}$. Therefore, $d_{M_{i+1}}(s,v) \leq 2^{i+1} + 2 \cdot 2^i = 2 \cdot 2^{i+1}$, where the term $2 \cdot 2^i$ is present to account for the distance between $s$ and the terminal in $C_s$ which is closest to $A_1$ — this distance can be at most $2 \cdot 2^i$ by the induction hypothesis.

For part (c), consider the last stage $i$ such that the supernodes $C_s$ and $C_{\bar{s}}$ are distinct (so the result in part (b) applies to this stage). The same argument as above applies except that when we consider the path from $C_s$ to $C_{\bar{s}}$ in the component of $\mathcal{G}_i$ containing them, we will have to account for the first and the last edges in this path.

We are now ready to prove the uni-strictness of the cost-shares. We run TimedClut on the instance $\mathcal{I}$ to get the cost shares, and let the cost share for $(s, \bar{s})$ be as in (5.6). Now let $F'$ be the forest returned by the algorithm $\mathcal{A}$ on the instance $\mathcal{I}'$. Recall that $M/F'$ denotes the metric $M$ with the connected components in $F'$ contracted to single points.
Lemma 5.6 The distance between $s$ and $\bar{s}$ in $M/F'$ is at most $4 \cdot (2^{\ell_s+1} + 2^{\ell_{\bar{s}}+1})$.

Proof. Let $j := \max(\ell_s, \ell_{\bar{s}}) + 1$. Suppose $j \geq \text{level}(s)$, then the claim is trivial because $d_M(s, \bar{s}) \leq 2^{\text{level}(s)} \leq 2^j \leq 2 \cdot (2^{\ell_s+1} + 2^{\ell_{\bar{s}}+1})$; hence consider the case where $j \leq \text{level}(s) - 1$.

Let $C_s$ and $C_{\bar{s}}$ denote the supernodes containing $s$ and $\bar{s}$ in clustering $\mathcal{C}^j$ respectively. There are two cases. The first case is when $C_s$ is same as $C_{\bar{s}}$. In this case, part (c) of Theorem 5.5 implies that $d_{M_j}(s, \bar{s}) \leq 4 \cdot 2^j \leq 4 \cdot (2^{\ell_s+1} + 2^{\ell_{\bar{s}}+1})$. The distance in metric $d_{M/F'}$ can only be smaller.

The other case is when $C_s$ and $C_{\bar{s}}$ are different. Note that $C_s$ will be merged with another supernode in some stage during or after stage $j$ (eventually the $s$ and $\bar{s}$ will end up in the same supernode). Since $j > \ell_s$, it follows from the definition of $\ell_s$ that $s$ is not the leader of $C_s$. Similarly, $\bar{s}$ is not the leader of $C_{\bar{s}}$. Let the leaders of $C_s$ and $C_{\bar{s}}$ be $v_1$ and $v_2$ respectively. By Theorem 5.5(b), we know that $d_{M_j}(s, v_1) \leq 2 \cdot 2^j$ and $d_{M_j}(\bar{s}, v_2) \leq 2 \cdot 2^j$. Consequently,

$$d_{M_j}(v_1, v_2) \leq d_{M_j}(v_1, s) + d_{M_j}(s, \bar{s}) + d_{M_j}(\bar{s}, v_2) \leq 4 \cdot 2^j + d_M(s, \bar{s}) \leq 5 d_M(s, \bar{s}),$$

where the last inequality follows because $d_M(s, \bar{s}) \geq d_{M_j}(C_s, C_{\bar{s}}) \geq 2^j$.

Let $F''$ be the final forest produced by $\text{TimedGlut}$ on the instance $\mathcal{J}'$; recall that $F'$ is obtained from $F''$ by merging together some of these trees. Let $T_1$ and $T_2$ be the trees in $F''$ which contain $v_1$ and $v_2$ respectively. Since the distance between $v_1$ and $v_2$ is already at most $5 d_M(s, \bar{s})$ at the beginning of stage $j$, we know that $d_{M/F'}(T_1, T_2) \leq 5 d_M(s, \bar{s})$, where $M/F'$ denotes the metric $M$ with the trees in $F''$ contracted.

Since $s$ lost its leadership to $v_1$, it must be the case that $d(s, \bar{s}) \leq d(v_1, v_1)$; thus $\text{width}(T_1) \geq d(s, \bar{s})$; a similar argument shows $\text{width}(T_2) \geq d(s, \bar{s})$. Since $d_{M/F'}(T_1, T_2) \leq 5 \min(\text{width}(T_1), \text{width}(T_2))$, the algorithm $\mathcal{A}$ would have merged $T_1$ and $T_2$ into one tree. This makes the distance $d_{M/F'}(v_1, v_2) = 0$ and hence

$$d_{M/F'}(s, \bar{s}) \leq d_{M/F'}(s, v_1) + d_{M/F'}(v_2, \bar{s}) \leq d_{M/F}(s, v_1) + d_{M/F}(\bar{s}, v_2) \leq 4 \cdot 2^j,$$

proving the claim. \hfill \blacksquare

This shows that the cost of connecting $(s, \bar{s})$ in $M/F'$ is at most $\beta := 16 \gamma_{\mathcal{C}}$ times the cost share of $(s, \bar{s})$, which proves the uni-strictness property.

5.2 Strict Cost Shares

We now extend the previous cost sharing scheme to the more general strict cost sharing scheme. Let $\chi$ be a budget-balanced cost sharing function for the Steiner forest problem. As before, let $\mathcal{A}$ be an $\alpha$-approximation algorithm for the Steiner forest problem.

Definition 5.7 A cost-sharing function $\chi$ is $\beta$-strict with respect to an algorithm $\mathcal{A}$ if for all pairs of disjoint terminal sets $\mathcal{D}_1, \mathcal{D}_2$ lying in a metric $M$, the following condition holds: if $\mathcal{D}$ denotes $\mathcal{D}_1 \cup \mathcal{D}_2$, then $\sum_{(s, \bar{s}) \in \mathcal{D}_2} \chi(M, \mathcal{D}, (s, \bar{s}))$ is at least $1/\beta$ times the the cost of the optimal Steiner forest on $\mathcal{D}_2$ in the metric $M/F$, where $F$ is the forest returned by $\mathcal{A}$ on the input $(M, \mathcal{D}_1)$.

In addition to the $\text{TimedGlut}$ algorithm, we will also need a timed primal-dual algorithm for Steiner forest, denoted by $\text{TimedPD}$. The input for the $\text{TimedPD}$ algorithm is a set of terminals, each terminal $s$ being assigned an activity time $\text{time}(s)$ such that the terminal is active for all times $t \leq \text{time}(s)$. 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., [GKPR07, Pál04, KLS05]); for concreteness we refer to the KLS algorithm of K"onemann et al. [KLS05] which gives the following guarantee:
Theorem 5.8 If time\(s) = \frac{1}{2}d_M(s, \bar{s})\) for all terminals \(s\), then the total cost of edges bought by the timed primal-dual algorithm KLS is at most \(2 \cdot \text{opt}(\mathcal{F})\).

The following property can be shown for the KLS algorithm:

Lemma 5.9 Multiplying the activity times by a factor of \(K \geq 1\) to \(\frac{K}{2} \cdot d_M(s, \bar{s})\) causes the KLS algorithm to output another feasible solution of total cost at most \(2K \cdot \text{opt}(\mathcal{F})\).

5.2.1 Defining \(\chi\) and \(\mathcal{A}\)

Defining \(\chi\) and \(\mathcal{A}\): To define the cost-shares for the instance \(\mathcal{I} = (M, D)\), run the algorithm TimedGlut on \(\mathcal{I}\). Recall the description of the algorithm as given in § 4.1: in each stage \(i\), we choose a collection \(P^{(i)}\) of pairs of supernodes whose mutual distance (in the metric \(M/\mathcal{E}^i\)) lies in the range \([2^i, 2^{i+1})\), merge each such pair of supernodes to get the new clustering (and add edges in the underlying graph of at most as much length). For pair \((S, S') \in P^{(i)}\), if \(s, s'\) are the leaders of \(S, S'\) respectively, increment the cost-share of each of \((s, \bar{s})\) and \((s', \bar{s}')\) by \(\frac{2^{i+1}}{2^\tau_{TG}}\). Since the analysis of the TimedGlut algorithm proceeds by showing that the quantity \(\sum_i |P^{(i)}| \cdot 2^{i+1} \leq \gamma_{TG} \text{opt}(\mathcal{I})\), the budget-balance property follows.

The algorithm \(\mathcal{A}\) on input \(\mathcal{I} = (M, D)\) is simple: set the activity time \(time(s) := 6 \cdot 2^{\text{level}(s)} + 1\) for each terminal \(s\), and run the algorithm TimedPD. The following claim immediately follows from Lemma 5.9 and Theorem 5.8, and the fact that \(6 \cdot 2^{\text{level}(s)} + 1 \leq 12 \cdot 2^2 \cdot \frac{1}{2}d_M(s, \bar{s})\).

Lemma 5.10 The algorithm \(\mathcal{A}\) is a 96-approximation algorithm for Steiner forest.

5.2.2 Proving Strictness

Given a set of demands \(D\) in a metric space \(M\), and a partition into \(D_1 \cup D_2\), we run the algorithm \(\mathcal{A}\) on \(D_1\) — let \(F_1\) be the forest returned by this algorithm, and let metric \(M_1\) be obtained by contracting the edges of \(F_1\). To prove the strictness property, we now exhibit a “candidate” Steiner forest \(F_2\) for \(D_2\) in the metric \(M_1\) with cost at most a constant factor times \(\sum_{(s, \bar{s}) \in D_2} \chi(M, D, (s, \bar{s}))\), the total cost-share assigned to the terminals in \(D_2\).

Recall that the algorithm \(\mathcal{A}\) on \(D_1\) is just the TimedPD algorithm. We divide this algorithm’s run into stages, where the \(i^{th}\) stage lasts for the time interval \([6 \cdot 2^i, 6 \cdot 2^{i+1})\); the \(0^{th}\) stage lasts for \([0, 6 \cdot 2)\). Let \(F_1^{(i)}\) be the edges of the output forest \(F_1\) which become tight during stage \(i\) of this run, and \(\mathcal{M}^{(i)}\) be the set of moats at the beginning of stage \(i\). These moats are defined in the original metric \(M\).

Defining a Candidate Forest \(F_2\): To define the forest \(F_2\) connecting \(D_2\), we now imagine running TimedGlut on the entire demand set \(D_1 \cup D_2\) on the original metric \(M\), 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 \(\chi\) in the first place. Recall the description of TimedGlut from § 4.1, and let \(\mathcal{R}\) denote this run of TimedGlut on \(\mathcal{I} = (M, D_1 \cup D_2)\).

We examine the run \(\mathcal{R}\) stage by stage: at the beginning of stage \(i\), the run \(\mathcal{R}\) took the current clustering \(\mathcal{E}^i\), built an auxiliary graph \(H^{(i)}\) whose nodes were the supernodes in \(\mathcal{E}^i\) and edges were pairs of active supernodes that had mutual merging distance at most \(2^{i+1}\), picked some maximal forest \(P^{(i)}\) in this graph, merged these supernode pairs in \(P^{(i)}\), and bought edges in the underlying metric corresponding to paths connecting these supernode pairs. We show how to choose some subset of these underlying edges to add to our candidate forest—we denote these edges by \(F_2^{(i)}\).

In the following, we will talk about edges \((S, S') \in P^{(i)}\) (which are edges of the auxiliary graph \(H^{(i)}\)) and edges in the metric \(M\). To avoid confusion, we refer to \((S, S')\) as pairs and those in the metric as edges.
What edges should we add to $F_2^{(i)}$? For that, look at the end of stage $i$ of the run of $\mathcal{A}$ on $\mathcal{D}_1$; the primal-dual algorithm has formed a set of moats $\mathcal{M}_1^{(i+1)}$ at this point. We define an equivalence relation on the supernodes in $\mathcal{G}^i$ as follows. If the leaders of two supernodes $S, S' \in \mathcal{G}^i$ both lie in $\mathcal{D}_1$, and also in the same moat of $\mathcal{M}_1^{(i+1)}$, we put $S$ and $S'$ in the same equivalence class. Now if we collapse each equivalence class by identifying all the supernodes in that class, the pairs in $\mathcal{P}^{(i)}$ may no longer be acyclic in the collapsed version of $H^{(i)}$, they may contain cycles and self-loops. Consider a maximal acyclic set of pairs in $\mathcal{P}^{(i)}$, and denote the dropped pairs by $\mathcal{P}^{(i)}_X$. The set $\mathcal{P}^{(i)}$ is now classified into three parts (see Figure 5.6 for an example):

- Let $\mathcal{P}^{(i)}_G$ be pairs $(S, S') \in \mathcal{P}^{(i)} \setminus \mathcal{P}^{(i)}_X$ for which at least one of $\text{leader}(S), \text{leader}(S')$ belongs to $\mathcal{D}_2$.
- Let $\mathcal{P}^{(i)}_B$ be pairs $(S, S') \in \mathcal{P}^{(i)} \setminus \mathcal{P}^{(i)}_X$ for which both of $\text{leader}(S)$ and $\text{leader}(S')$ belongs to $\mathcal{D}_1$.
- Of course, $\mathcal{P}^{(i)}_X$ is the set of pairs $(S, S') \in \mathcal{P}^{(i)}$ dropped to get an acyclic set.

Given this classification, the edges we add to $F_2^{(i)}$ are as follows. Recall that in the run $\mathcal{R}$, for each pair $(S, S') \in \mathcal{P}^{(i)}$, we had added edges connecting those two supernodes of total length at most $2^{i+1}$. For each pair in $\mathcal{P}^{(i)} \setminus \mathcal{P}^{(i)}_X$, we now add the same edges to $F_2^{(i)}$. We further classify these edges based on their provenance: the edges added due to a pair $(S, S') \in \mathcal{P}^{(i)}_G$ we call good edges, and those added due to a pair in $\mathcal{P}^{(i)}_B$ we call bad edges. Observe that we add no edges for pairs in $\mathcal{P}^{(i)}_X$.

![Figure 5.6](image_url)

Figure 5.6: The solid edges denote the pairs $\mathcal{P}^{(i)}$, and the grey regions denote the equivalence classes. Let $\mathcal{P}^{(i)}_X$ be the pairs $\{(S_2, S_6), (S_4, S_{10}), (S_3, S_{10}), (S_3, S_6)\}$. Assume that leaders of $S_1$ and $S_2$ belong to $\mathcal{D}_2$ (the leaders of the rest of the supernodes must be in $\mathcal{D}_1$ because the equivalence classes corresponding to these supernodes have cardinality larger than 1). So, $\mathcal{P}^{(i)}_G$ consists of pairs $\{(S_6, S_{12}), (S_{11}, S_9)\}$ and $\mathcal{P}^{(i)}_B$ consists of $\{(S_1, S_7), (S_2, S_8), (S_5, S_{10})\}$.

This completes the construction of the set $F_2^{(i)}$. The forest $F_2$ is obtained by taking the union of $\cup_i F_2^{(i)}$. The task now is to show (a) feasibility, that the edges in $F_2$ form a Steiner forest connecting up the demands of $\mathcal{D}_2$ in metric $\mathcal{M}_1$, or equivalently that $F_1 \cup F_2$ is a Steiner forest on the set $\mathcal{D}_1 \cup \mathcal{D}_2$, and (b) strictness, that the cost of $F_2$ is comparable to the cost shares assigned to the demands in $\mathcal{D}_2$.

**Feasibility** First, we show feasibility, i.e., that $F_1 \cup F_2$ connects all pairs in $\mathcal{D}_2$. Observe that we took the run $\mathcal{R}$, and added to $F_2$ some of the edges added in $\mathcal{R}$. Had we added all the edges, we would trivially get feasibility (but not the strictness), but we omitted edges corresponding to pairs in $\mathcal{P}^{(i)}_X$. The idea of the proof is that such supernodes will get connected due to the other connections, and to the fact that we inflated the activity times in the TimedPD algorithm. Let’s give the formal proof, which proceeds by induction over time.

For integer $i$, define $F_1^{(\leq i)} := \cup_{j \leq i} F_1^{(j)}$, and define $F_2^{(\leq i)}$ similarly. The first claim relates the stages in the run $\mathcal{R}$ of TimedGlut ($\mathcal{D}_1 \cup \mathcal{D}_2$) to the stages in the run of TimedPD.

**Claim 5.11** If terminal $s \in \mathcal{D}_1$ is active at the beginning of stage $i$ in the run $\mathcal{R}$, then the moat containing $s$ remains active during stage $i$ of the run of TimedPD on $\mathcal{D}_1$.  

20
Proof. Since \( s \in \mathcal{D}_1 \) is active in stage \( i \), \textit{level}(s) \geq i. Hence its activity time \textit{time}(s) \geq 4 \cdot 2^{i+1}. Since stage \( i \) for the timed primal-dual algorithm ends at time \( 4 \cdot 2^{i+1}, \) the moat containing \( s \) in TimedPD will be active at least until the end of stage \( i \).

Lemma 5.12 Let \( \mathcal{C}^i \) be the clustering at the beginning of stage \( i \) in the run \( \mathcal{R} \). Then,

(a) For any \( S \in \mathcal{C}^i \), all terminals in \( S \) lie in the same connected component of \( F_1^{(\leq i-1)} \cup F_2^{(\leq i-1)} \).

(b) For every \((S,S') \in \mathcal{P}^{(i)} \setminus \mathcal{P}^{(i)}_X \), the terminals in \( S \cup S' \) lie in the same connected component of \( F_1^{(\leq i-1)} \cup F_2^{(\leq i)} \).

Proof. We first show that if the statement (a) is true for some stage \( i \), then the corresponding statement (b) is also true (for this stage). Consider a pair \((S,S') \in \mathcal{P}^{(i)} \). If \((S,S') \in \mathcal{P}^{(i)}_G \cup \mathcal{P}^{(i)}_B \), the edges we add to \( F_2^{(i)} \) would connect the terminals in \( S \cup S' \), as long as all the supernodes in \( \mathcal{C}^i \) formed connected components. But by the assumption, we know that edges in \( F_1^{(\leq i-1)} \cup F_2^{(\leq i-1)} \) connect up each supernode in \( \mathcal{C}^i \). Consequently, terminals in \( S \cup S' \) lie in the same connected component of \( F_1^{(\leq i-1)} \cup F_2^{(\leq i)} \). This proves statement (b).

We now prove statement (a) by induction on \( i \). At the beginning of stage \( i = 0 \), each supernode \( S \) is a singleton and hence the statement is true.

Now to prove the induction step for (a). It suffices to show that if \((S,S') \in \mathcal{P}^{(i)}_X \) then \( S \cup S' \) is contained in the same component in \( F_1^{(\leq i)} \cup F_2^{(\leq i)} \). We distinguish two cases. The first case is when \( S \) and \( S' \) both lie in the same equivalence class that was used to construct \( \mathcal{P}^{(i)}_X \). Then \( s = \text{leader}(S) \) and \( s' = \text{leader}(S') \) belong to \( \mathcal{D}_1 \) and also to the same moat in \( \mathcal{M}^{(i+1)} \), at the end of stage \( i \). Since \( S, S' \) are active in stage \( i \) of TimedGlut (\( \mathcal{D}_1 \)), both \( s, s' \) have level at least \( i \). By Claim 5.11 they remain active throughout stage \( i \) of TimedPD. Moreover, the end of that stage they share the same moat. Hence, \( s, s' \) belong to the same moat when active—but recall that the TimedPD algorithm ensures that whenever two active terminals belong to the same moat they lie in the same connected component. Hence \( s, s' \) must lie in the same connected component of \( F_1^{(\leq i)} \). By the induction hypothesis, the rest of \( S, S' \) are connected to their leaders in \( F_1^{(\leq i-1)} \cup F_2^{(\leq i-1)} \). Hence \( S, S' \) are connected in \( F_1^{(\leq i)} \cup F_2^{(\leq i-1)} \); indeed for each equivalence class, the supernodes that belong to it are connected using those edges.

The second case is when for pair \((S,S') \in \mathcal{P}^{(i)}_X \), the supernodes \( S \) and \( S' \) do not fall in the same equivalence class, but the adding pair \((S,S') \) to \( \mathcal{P}^{(i)} \setminus \mathcal{P}^{(i)}_X \) would form a cycle when equivalence classes are collapsed. The argument here is similar: if the leaders are again \( s, s' \), then the above arguments applied to each pair on the cycle, and to each equivalence class imply that \( s \) and \( s' \) must be connected in \( F_1^{(\leq i)} \cup F_2^{(\leq i-1)} \)—and therefore so must \( S \cup S' \).

Since each pair \( \{s,s'\} \in \mathcal{D}_2 \) is contained in some supernode at the end of the run \( \mathcal{R} \), Lemma 5.12 implies that they are eventually connected in using \( F_1 \cup F_2 \) as well. This completes the proof that \( F_1 \cup F_2 \) is a feasible solution to the demands in \( \mathcal{D}_2 \).

Bounding the Cost of Forest \( F_2 \) Finally, we want to bound the cost of the edges in \( F_2 \) by a constant times \( \sum_{(s,s') \in \mathcal{D}_2} \chi(M, \mathcal{D}_1 \cup \mathcal{D}_2, (s,s)) \). If \( \mathcal{P}^{(i)}_B := |\mathcal{P}^{(i)}_B| \), then the total cost of bad edges is at most

\[
\sum_i \mathcal{P}^{(i)}_B \cdot 2^{i+1}
\]

because the length of edges added for each connection in \( \mathcal{P}^{(i)}_B \) is at most \( 2^{i+1} \).

Lemma 5.13 The total cost of the edges in \( \cup_i F_2^{(i)} \) is at least \( 3 \sum_i \mathcal{P}^{(i)}_B \cdot (2^{i+1} - 2^i) \).
Proof. For this proof, recall that we run the primal-dual process on the metric $\mathcal{M}$, and $\mathcal{M}_1^{(i)}$ are the dual moats at the beginning of stage $i$. Let $E_i$ denote the set of tight edges lying inside the moats in $\mathcal{M}_1^{(i)}$. We prove the following statement by induction on $i$: the total cost of edges in $F_2^{(\leq i)} \cap E_i$ is at least $3 \sum_{j \leq 1} p_B^{(j)} \cdot (2^{j+1} - 2^j)$.

The base case for $i = 0$ follows trivially because the $F_2^{(0)}$ is empty, and $p_B^{(0)}$ is also 0.

Suppose the statement is true for some $i - 1$. Now consider the pairs in $\mathcal{P}_B^{(i)}$—these correspond to pairs of supernodes $(S, S')$ whose leaders lie in $\mathcal{P}_1$. The pairs in $\mathcal{P}_B^{(i)}$ form an acyclic set in the auxiliary graph $H^{(i)}$. Consider the set of supernodes which occur as endpoints of the edges in $\mathcal{P}_B^{(i)}$, and let $L_B$ be the set of terminals that are leaders of these supernodes. Now pick a maximal set of these supernodes subject to the constraint that all of them lie in different moats in $\mathcal{M}_1^{(i+1)}$; i.e., no two of them lie in the same equivalence class. Let $L_B^* \subseteq L_B$ be the set of terminals that are leaders of this maximal set. In the example given in Figure 5.6, $L_B = \{S_1, S_2, S_5, S_7, S_8, S_{10}\}$, and we could define $L_B^*$ as $\{S_1, S_5, S_7, S_8\}$. By the fact that $\mathcal{P}_B^{(i)}$ is an acyclic set, we get $|L_B^*| \geq p_B^{(i)} + 1$.

For any $s \in L_B$, let $M_s$ be the moat in $\mathcal{M}_1^{(i)}$ containing $s$. By construction,

- For each pair $(S, S') \in \mathcal{P}_B^{(i)}$, the leaders of $S$ and $S'$ lie in different moats in $\mathcal{M}_1^{(i+1)}$.
- For all $s \neq s' \in L_B^*$, $s$ and $s'$ belong to different moats in $\mathcal{M}_1^{(i+1)}$.

Also note that terminals in distinct moats of $\mathcal{M}_1^{(i+1)}$ a fortiori lie in distinct moats in $\mathcal{M}_1^{(i)}$. Now contract all the moats in $\mathcal{M}_1^{(i)}$ in the metric $\mathcal{M}$. Observe that all edges in $F_1^{(\leq i-1)}$ were already tight by the end of stage $i - 1$, and hence get contracted by this operation.

By Lemma 5.12(b), for each $s \in L_B^*$ the edges in $F_1^{(\leq i-1)} \cup F_2^{(\leq i)}$ connect moat $M_s$ to some another moat $M_{s'}$ for some $s' \in L_B$ (corresponding to the pair in $\mathcal{P}_B^{(i)}$). Since we contracted the moats in $\mathcal{M}_1^{(i)}$, the edges in $F_2^{(\leq i)}$ connect $M_s$ to $M_{s'}$ in this contracted metric. But any two moats $M_s, M_{s'}$ are at least $6 \cdot (2^{i+1} - 2^i)$ apart in this contracted metric (because these moats do not meet during stage $i$ of the primal dual growing process, else $s, s'$ would share a moat in $\mathcal{M}_1^{(i+1)}$). Therefore, if we draw a ball of radius $3(2^{i+1} - 2^i)$ around the moat $M_s$ in this contracted metric, this ball contains edges from $F_2^{(\leq i)}$ of length at least $3(2^{i+1} - 2^i)$.

Since these balls around the terminals in $L_B^*$ are all disjoint, the total length of all edges in these balls is at least

$$|L_B^*| \cdot 3 \cdot (2^{i+1} - 2^i) \geq 3(p_B^{(i)} + 1) \cdot (2^{i+1} - 2^i).$$

All these edges lie in moats in $\mathcal{M}_1^{(i+1)}$ but not within moats in $\mathcal{M}_1^{(i)}$, so we add these to the bound we get for $F_2^{(\leq i-1)} \cap E_{i-1}$ from the induction hypothesis and complete the inductive step. 

Theorem 5.14 The cost of edges in $F_2$ is at most $6\gamma_{G\mathcal{P}_2} \times \sum_{(s, s') \in \mathcal{P}_2} \chi(\mathcal{M}, \mathcal{P}_1 \cup \mathcal{P}_2, (s, s'))$.

Proof. By Lemma 5.13, the cost of edges in $F_2$ is at least $3 \sum_j p_B^{(j)} \cdot (2^{j+1} - 2^j) \geq \frac{3}{2} \sum_j p_B^{(j)} \cdot 2^{j+1}$, by (5.8). So the cost of the good edges in $F_2$ is at least one third of the cost of all edges in $F_2$.

However, observe that good edges correspond to pairs $(S, S') \in \mathcal{P}_B^{(i)}$ for some $i$, i.e., this pair of supernodes was connected by these good edges in stage $i$ of TimedGlut, and moreover, at least one of the leaders of $S$ and $S'$ belong to a terminal pair in $\mathcal{P}_2$. By the construction of our cost shares, it follows the cost share of terminal pairs in $\mathcal{P}_2$ is at least $\frac{1}{3\gamma_{G\mathcal{P}_2}}$ times the cost of the good edges. Hence the cost of the forest $F_2$ is at most $6\gamma_{G\mathcal{P}_2}$ times the cost share of terminal pairs in $\mathcal{P}_2$, proving the theorem. 

22
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.

References


A The Lower Bound for (Paired) Greedy

The (paired) greedy algorithm picks, at each time the closest yet-unconnected source-sink pair $(s, \bar{s})$ in the current graph, and connects them using a shortest $s-\bar{s}$ path in the current graph. A lower bound of $\Omega(\log n)$ for this algorithm was given by Chen, Roughgarden, and Valiant [CRV10]. We repeat this lower bound here for completeness.

Take a cubic graph $G = (V, E)$ with $n$ nodes and girth at least $c \log n$ for constant $c$; see [Big98] for constructions of such graphs. Fix a spanning tree $T$ of $G$, and let $E' = E \setminus E(T)$ be the non-tree edges. Set the lengths of edges in $E(T)$ to 1, and the lengths of edges in $E'$ to $\frac{c}{2} \log n$.

Let $M$ be a maximal matching in $G' = (V, E')$; since $G$ and hence $G'$ has maximum degree 3, this maximal matching has size at least $\Omega(|E'|) = \Omega(n)$. The demand set $\mathcal{D}$ consists of the matching $M$.

We claim that the paired-greedy algorithm will just buy the direct edges connecting the demand pairs. This will incur cost $|M| \cdot \frac{c}{2} \log n = \Omega(n \log n)$. The optimal solution, on the other hand, is to buy the edges of $T$, which have total cost $n - 1$, which gives the claimed lower bound of $\Omega(\log n)$.

The proof of the claim about the behavior of paired-greedy is by induction. Suppose it is true until some point, and then demand $(u, v)$ is considered. By construction, $(u, v) \in M$. We can model the fact that we bought some previous edges by zeroing out their lengths. Now the observation is that for any path $P$ between $u$ and $v$ that is not the direct edge $(u, v)$, if there are $k$ edges from $E'$ on this path, then at most $\lceil k/2 \rceil$ of them can be zeroed out. Moreover, since the girth is $g$, the number of edges on the path is at least $g - 1$. So the new length of the path $P$ is at least $(g - 1 - k) + \lceil k/2 \rceil \frac{c}{2} \log n \geq g - 2$. This means the direct edge between $u, v$ is still the (unique) shortest path, and this proves the claim and the result.

B A Different Gluttonous Algorithm, and its Analysis

In this section, we consider a slightly different analysis of the gluttonous algorithm which does not rely on the faithfulness property developed in Section 3.2. We then use this analysis to show that a different gluttonous algorithm is also a constant-factor approximation.

As before, let $\mathcal{F}^*$ denote an optimal solution to the Steiner forest instance $\mathcal{I} = (\mathcal{M}, \mathcal{D})$, and let the trees in $\mathcal{F}^*$ be $T_1^*, T_2^*, \ldots, T_p^*$. A supernode $S$ is a subset of terminals – note that we no longer require that the supernodes formed during the gluttonous algorithm should be contained in one of the trees of $\mathcal{F}^*$. Let $\mathcal{C}^{(t)}$ denote the clustering at the beginning of iteration $t$. So $\mathcal{C}^{(1)}$ consists of singleton supernodes.

For each index $r$, $1 \leq r \leq p$, we can think of a new instance $\mathcal{I}_r$ with set of terminals $V(T_r^*)$ and metric $\mathcal{M}_r$ (the metric $\mathcal{M}$ restricted to these terminals). We now define the notion of projected supernodes.

For each tree $T_r^*$, we shall maintain a clustering $\mathcal{C}_r^{(t)}$ corresponding to $\mathcal{C}^{(t)}$ – this should be seen as the restriction of the algorithm $\mathcal{A}$ to the instance $\mathcal{I}_r$. A natural way of defining this would be $S \cap V(T_r^*)$ for every $S \in \mathcal{C}^{(t)}$. But it turns out that we will really need a refinement of the latter clustering. The reason for this is as follows: if the algorithm $\mathcal{A}$ merges two active supernodes in the instance $\mathcal{I}_r$, the intersection of the two supernodes with $V(T_r^*)$ may be inactive supernodes. But we do not want to combine two inactive supernodes in the clustering $\mathcal{C}_r^{(t)}$.

We now define the clustering $\mathcal{C}_r^{(t)}$ formally. For a supernode $S$, let $S_r$ denote $S \cap V(T_r^*)$. We say that $S_r$ is active if there is some demand pair $(u, \bar{u})$ such that $u \in S_r$, but $\bar{u} \notin S_r$. Let alive$(S)$ denote the set of indices $r$ such that $S_r$ is active. For each iteration $t$, we shall maintain the following invariants.

(i) The clustering $\mathcal{C}_r^{(t)}$ will be a refinement of the clustering $\{S_r : S \in \mathcal{C}^{(t)}\}$.

24
(ii) For each (active) supernode $S \in \mathcal{C}^{(t)}$ such that $S_r$ is active, there will be exactly one active supernode $A_r(S) \subseteq S_r$ in the clustering $\mathcal{C}^{(t)}_r$.

Initially, $\mathcal{C}^{(1)}_r$ is the clustering consisting of singleton sets (it is easy to check that it satisfies the two invariants above because $\mathcal{C}^{(1)}$ also consists of singleton sets). Suppose at iteration $t$, the algorithm $\mathcal{A}$ merges supernodes $S'$ and $S''$ to a supernode $S$. If $S'_r$ and $S''_r$ are both active (i.e., $r \in \text{alive}(S') \cap \text{alive}(S'')$), then we replace $A_r(S')$ and $A_r(S'')$ by their union in the clustering $\mathcal{C}^{(t)}_r$ to get $\mathcal{C}^{(t+1)}_r$. Otherwise, $\mathcal{C}^{(t+1)}_r$ is same as $\mathcal{C}^{(t)}_r$. It is easy to check that the above two invariants will still be satisfied. Further, observe that we only merge two active supernodes of $\mathcal{C}^{(t)}_r$, and an inactive supernode never merges with any other supernode.

Let $\delta_{t,r}$ denote the minimum over two active supernodes $S', S'' \in \mathcal{C}^{(t)}_r$ of $d_{M_r/\mathcal{C}^{(t)}_r}(S', S'')$ (in case $\mathcal{C}^{(t)}_r$ has only one active supernode, this quantity is infinity). Fact 3.12 implies that $\delta_{t,r}$ is ascending with $t$. Let $\delta_t$ denote the minimum over all $r$ of $\delta_{t,r}$. Clearly, $\delta_t$ is also ascending with $t$.

Whenever the algorithm merges active supernodes $S'$ and $S''$, we will think of merging the corresponding supernodes $A_r(S')$ and $A_r(S'')$ in the instance $\mathcal{I}_r$ for each value of $r$ (provided $r \in \text{alive}(S') \cap \text{alive}(S'')$). Note that the latter merging cost may not suffice to account for the merging cost of $S'$ and $S''$. For example, suppose $S' = \{a\}$, $S'' = \{b\}$, and $a$ and $b$ happen to be in different trees in $\mathcal{T}^*$, say $T^*_1$ and $T^*_2$ respectively. If $S = \{a, b\}$ is the new supernode, then $A_1(S) = A_1(S') = \{a\}$ and $A_2(S) = A_2(S'') = \{b\}$. So the merging costs in the corresponding instances is 0, but the actual merging cost is positive. The main idea behind the proof is that we will pay for this merging cost later when the algorithm merges the supernode containing $a$ with some other supernode which has non-empty intersection with $T^*_1$ (and similarly for $b$).

In order to keep track of the unpaid merging cost, we associate a charge with each supernode (which gets formed during the gluttonous algorithm $\mathcal{A}$) – let $\text{charge}(S)$ denote this quantity. At the beginning of every every iteration $t$ and supernode $S \in \mathcal{C}^{(t)}$, we will maintain the following invariant:

\begin{equation}
\text{charge}(S) \leq \begin{cases} 
0 & \text{if } S \text{ is inactive at the beginning of iteration } t. \\
(n(S) - 1) \cdot \delta_t & \text{otherwise},
\end{cases}
\end{equation}

where $n(S)$ denotes the number of indices $r$ such that $S_r$ is active, i.e., $n(S) = |\text{alive}(S)|$. Now, we show how the quantity $\text{charge}(S)$ is updated. Initially (at the beginning of iteration 1), we have a supernode $\{u\}$ for each terminal $u$. The charge associated with each of these supernodes is 0. Clearly, the invariant condition above is satisfied.

Suppose the algorithm $\mathcal{A}$ merges supernodes $S'$ and $S''$ in iteration $t$. Let $S$ denote the new supernode. We define $\text{charge}(S)$ as

\begin{equation}
\text{charge}(S) = \text{charge}(S') + \text{charge}(S'') - (2|\text{alive}(S') \cap \text{alive}(S'')| - 1)\delta_t.
\end{equation}

We first prove that the invariant about charge of a supernode is satisfied.

**Claim B.1** The invariant conditions (B.9) are always satisfied.

**Proof.** We prove this by induction on the iteration $t$. Suppose $\mathcal{A}$ merges (active) supernodes $S'$ and $S''$ iteration $t$, and let $S$ be the new supernode. If we consider a supernode $S_1$ other than $S$ in $\mathcal{C}^{(t+1)}$, then the statement follows easily. Note that $\text{charge}(S_1)$ does not change. If $S_1$ were inactive before iteration $t$, it will continue to be inactive at the end of iteration $t$ as well. If $S_1$ were active before iteration $t$, then it will continue to be active be at the beginning of iteration $(t + 1)$ as well. Further,
The algorithm picks the two active supernodes with smallest distance between them, where distances
are just the initial graph \( G(1) \) is just the initial graph \( G \) on the vertices (i.e., terminals and Steiner nodes). In iteration \( t \), the algorithm picks the two active supernodes with smallest distance between them, where distances are measured with respect to the edge lengths in \( G(t) \). Let \( P_t \) be the corresponding shortest path (all internal vertices of \( P_t \) must be inactive supernodes). We contract all the edges of \( P_t \) to get the graph \( G(t+1) \). The clustering \( \mathcal{C}(t+1) \) will be the same as \( \mathcal{C}(t) \) except for the fact that the supernodes corresponding to vertices of \( P_t \) will get replaced by a new supernode which will be the union of all these supernodes.

For the above algorithm, the proof of Theorem 3.6 does not hold. More specifically, consider case I in
the proof where \( u \) and \( v \) happen to be in the same tree in \( \mathcal{T}^* \). Now, the algorithm may contract a
path with end-points containing \( u \) and \( v \) respectively. Further, the internal vertices of this path could correspond to inactive supernodes, and so, the new supernode will contain terminals outside this tree. But we cannot pay for the cost of this path. However, we claim that the analysis of this section still applies.

**Lemma B.4** The above algorithm is a 96-approximation algorithm for the Steiner forest problem.

**Proof.** Let \( \mathcal{B} \) denote the version of the gluttonous algorithm described above. It is no longer true that an inactive supernode does not merge with any other supernode. In each iteration, the algorithm may choose to merge two active supernodes along with several inactive supernodes. However, the clusterings \( \mathcal{C}_{(t)}^{(r)} \) will respect the former property.

The clusterings \( \mathcal{C}_{(t)}^{(r)} \) will respect the invariants (i) and (ii) described above. When the algorithm \( \mathcal{B} \) merges active supernodes \( S' \) and \( S'' \) (along with possibly other inactive supernodes), and the index \( r \in \text{alive}(S') \cap \text{alive}(S'') \), we merge \( A_r(S') \) and \( A_r(S'') \) in the clustering \( \mathcal{C}_{(t)}^{(r)} \) to get \( \mathcal{C}_{(t+1)}^{(r)} \). It is easy to check that these two invariants are satisfied. The rest of the proof proceeds without any change. The key property that holds for this algorithm as well is that the merging cost (i.e., length of the shortest path) in each iteration is at most \( \delta_t \) (defined in the proof above). \( \blacksquare \)