Gluttonous: A greedy Algorithm for Steiner Forest

Vihang Agarwal, Aditya Modi, and Shun Zhang

{vihang,admodi,shunzh}@umich.edu

1 Problem Statement and Related Work

In the Steiner Forest problem, we are given a metric space $d(\cdot, \cdot)$, a set of points V and terminal pairs $\{s_i\ ,\ t_i\}_{i=1}^{k}$. The aim is to find the cheapest forest such that all terminal pairs are connected and each pair is contained in the same tree. This can be considered as a generalized version of the Steiner tree problem. Agrawal et al. (1995) gave a primal-dual method based constant factor approximation method for this problem which was later generalized and simplified by Goemans and Williamson (1995). Finding a greedy algorithm for this problem which also gives a constant factor approximation had been an interesting open problem which was solved by this work by Gupta and Kumar (2015). The paper follows a natural greedy approach called the *gluttonous algorithm* and proves that it gives a constant factor approximation.

We use $I = (M, D)$ to denote a Steiner forest instance. A solution forest to I will be represented by F which is a collection of trees $T \in F$. The cost of each tree $T = (V, E_T)$ is defined as the sum of length of all edges in T: $\texttt{cost}(T) = \sum_{e \in E_T} d(e)$. The cost of a forest F is the sum of cost of all trees T in F: $\texttt{cost}(F) = \sum_{T \in F} \texttt{cost}(T)$.

The Steiner forest problem considers the set of feasible forests F such that each terminal pair $(u, \bar{u})$ is connected and is part of the same tree $T_i \in F$. The problem is to find a feasible solution which has the minimum cost $\texttt{cost}(F)$.

2 The Gluttonous Algorithm

Goemans and Williamson (1995) provided a 2-approximation algorithm for the Steiner tree problem using a primal-dual method. In this report, we consider a greedy approach. The simplest way is a *paired greedy algorithm* (Chen et al., 2010): We iteratively connect unconnected pairs of terminals that have the minimum distance and terminate when all pairs of terminals are connected. However, this algorithm is no better than an $\Omega(\log n)$-approximation.

The gluttonous algorithm (Gupta and Kumar, 2015) that we present in this report has is a constant approximation. The idea is that we still want to greedily connect all terminals, but in each iteration, *we do not have to directly connect terminals that are pairs to each other*. We iteratively add edges (or paths) to connect vertices that are unmatched to their mates and terminate when all pairs of terminals are connected. We want to add an edge or path in each iteration that has a minimum *cost*. So we can hope that the algorithm provides a good approximation to the optimal solution.

Rather than connecting terminals, the gluttonous algorithm considers connecting or merging *clusters* that contain unmatched terminals. We call a cluster of a set of vertices a *supernode*. A supernode is *active* if it contains at least one terminal that is unmatched, and is *inactive* otherwise. Initially, each supernode contains one terminal, which is called the trivial clustering. Clearly, all supernodes are active initially. For any such clustering $C \equiv \{S_1, S_2, \ldots, S_q\}$ with S_i as supernodes, we define the C*-punctured* distance between two supernodes as follows:

1. Contract all vertices in a cluster to a supernode: the length of edges between nodes within a cluster are set to 0. For two nodes in different clusters, keep the original distance $d(u, v)$.

2. Considering G_C as the graph over clusters, define the distance between two clusters as $d_{M \setminus C}(S_1, S_2)$ as the shortest path distance between two supernodes in this graph: $\min_{u \in S_1, v \in S_2} d_{M \setminus C}(u, v)$.

The algorithm is described below. The cluster in iteration i is denoted by $C^{(i)}$. For convenience of analysis, we denote by C^g the clustering of vertices when the algorithm terminates. It is easy to see that each step takes polynomial time and the number of merges can at most be $O(k)$. (If the algorithm terminates after n iterations, then $C^g = C^{(n)}$.) Let F^g be the forest returned by the algorithm and F^* be the optimal solution.

Algorithm 1 The Gluttonous Algorithm

initialize $C^{(0)}$ to be the trivial clustering; the set of edges to add $E' \leftarrow \emptyset$; iteration index $i \leftarrow 0$
while there exist active supernodes in $C^{(i)}$ **do**
    find active supernodes $S_1, S_2 \in C^{(i)}$ with the minimum $C^{(i)}$-punctured distance.
    $C^{(i+1)} \leftarrow C^{(i)} \setminus \{S_1, S_2\} \cup \{S_1 \cup S_2\}$.
    find the shortest path from any $u \in S_1$ and $v \in S_2$ under metric $M \setminus C^{(i)}$; add to E' the edges on this path
    $i \leftarrow i + 1$
end while
return a maximal acyclic subgraph of E'

3 The Analysis for Gluttonous

First, we observe a monotonicity property about the merging distance of the algorithm. The merging distance for merging supernodes S, T in iteration i is the $C^{(i)}$-*punctured* distance $d_{M \setminus C^{(i)}}(S, T)$. Such distances should monotonically increase over iterations.

Claim 3.1. *(Gluttonous Merging Distances are Monotone) If supernodes S, T are merged in iteration i and supernodes S', T' are merged in iteration j, where $i \leq j$, then $d_{M \setminus C^{(i)}}(S, T) \leq d_{M \setminus C^{(j)}}(S', T')$.*

Consider an iteration where S, T are merged. Let supernode $U \neq S$ or T. We want to first show that the distance between U and its closet active supernode does not decrease after the merging: If S and T merge and then become inactive, we can easily see the distance between U and its closet active supernode would monotonically increase. If S and T merge and stay active, we consider the following two cases. 1) If U's closest supernode is either S or T, then the merging would not decrease the shortest distance to either S or T. 2) Otherwise, assume the closest supernode is some other supernode W. After the merging, if the distance to W is changed, it must be that the path uses vertices in S or T (the metric only changes on the edges in these supernodes). If that is the case, the closest supernode should have been S or T, which leads to a contradiction. Since the gluttonous algorithm always picks active supernodes with the minimum (punctured) distance, and any un-merged supernode's distance to its closest active supernode does not decrease, we can easily verify the above claim.

Now we are ready to present the main contribution of the paper:

Theorem 3.2. *The gluttonous algorithm is a constant-factor approximation for Steiner Forest.*

The paper does not bound the gluttonous algorithm solution with the optimal solution directly. Instead, it takes advantage of the idea of faithfulness and bounds the increase in cost against the optimal solution. We define the following two properties for the sketch of the analysis:

Definition 3.3. *A forest F is faithful to a clustering C if each supernode $S \in C$ (all vertices in S) is (are) contained within a single tree in F.*

Definition 3.4. *For a forest F that is a solution to instance I , and for any tree $T \in F$, let $\texttt{width}(T)$ denote the largest tree distance between any pair connected by T: $\texttt{width}(T) \coloneqq \max\{d_T(u, \bar{u}) \mid \{u, \bar{u}\} \subseteq V(T)\}$. Let the width of forest F be the sum of the widths of the trees in F.*

The analysis proves the bound in the following two steps:

- There exists a solution F that is faithful to C^g and $\texttt{cost}(F) \leq 2\texttt{cost}(F^*)$ (Theorem 3.5).
- For any solution F that is faithful to C^g, $\texttt{cost}(F^g) \leq O(1) \cdot \texttt{cost}(F)$ (Theorem 3.6).

3.1 Low-Cost and Faithful

First, we want to show that there are solutions faithful to the algorithm solution which have low costs.

Theorem 3.5. *Let $F^* = \{T_1^*, T_2^*, \ldots, T_p^*\}$ be an optimal solution to the Steiner forest instance $I = (M, D)$. There exists another solution F for instance I such that*

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

(b) F is faithful to the final clustering C^g produced by the gluttonous algorithm.

We can easily find an F that satisfies (a) (for example, F^*), but making sure that such F satisfies (b) is challenging. The plan is that, instead of aiming to find an F that is faithful to the final clustering (C^g), we first find an F that is faithful to the initial clustering (the trivial clustering), and inductively modify it to keep it being faithful to $C^{(i)}$ as the algorithm proceeds.

Satisfying condition (b). Concretely, we initialize $F^{(0)} \leftarrow F^*$. Clearly, $F^{(0)}$ is faithful to the initial clustering (since each terminal is in its own cluster). Inductively, assume we are in the i-th iteration of the algorithm and $F^{(i-1)}$ is faithful to our clustering $C^{(i-1)}$ at the beginning of this iteration. Suppose our algorithm connects two terminals u, v in the iteration. If u, v are in the same tree in $F^{(i-1)}$, then we simply let $F^{(i)} \leftarrow F^{(i-1)}$: We do not need to do anything and the current solution continues to be faithful. Otherwise, u, v are in different trees in $F^{(i-1)}$ but become connected in our clustering. In this case, we want to fix $F^{(i-1)}$ by connecting u, v in $F^{(i-1)}$: Let P be the shortest path between u, v in $M \setminus C^{(i)}$. We let $F^{(i)} \leftarrow F^{(i-1)} \cup E(P)$, where $E(P)$ is the set of edges in P. Let n be the index of the last iteration. By induction, $F^{(n)}$ is faithful to the final clustering after the algorithm terminates.

Satisfying condition (a). We know that the initial $F^{(0)}$ ($= F^*$) satisfies (a) by construction. Does $F^{(n)}$ at the end the process still satisfy (a)? We hope that we did not add edges with too much cost in this process. To show this, we consider how $\texttt{width}(F^{(i)})$ changes over iterations.

Consider an iteration where we add edges of path P to the tree. Suppose we pick $u \in T_1$, $v \in T_2$, where $T_1 \neq T_2$. Because of the greediness of the algorithm, we know $\texttt{cost}(P) \leq \min\{d_{T_1}(u, \bar{u}), d_{T_2}(v, \bar{v})\}$. Otherwise the algorithm would have picked $(u, \bar{u})$ or $(v, \bar{v})$ instead. By the definition of width, we know $\min\{d_{T_1}(u, \bar{u}), d_{T_2}(v, \bar{v})\} \leq \min\{\texttt{width}(T_1), \texttt{width}(T_2)\}$. So we have $\texttt{cost}(P) \leq \min\{\texttt{width}(T_1), \texttt{width}(T_2)\}$. The sum of costs of paths we added over iterations is upper bounded by the sum of widths of all trees in F^*. So we have $\texttt{cost}(F^{(n)}) \leq \texttt{cost}(F^*) + \texttt{width}(F^*)$.

Now we show that such $F^{(n)}$ satisfies both (a) and (b), which completes the proof.

3.2 Cost of gluttonous

In the first part of the analysis, it is shown that there exists a feasible solution to the Steiner forest problem which is faithful to the output of gluttonous. Now, we show that for any such faithful solution F, the cost of the approximate solution is at most 48 $\texttt{cost}(F)$.

Theorem 3.6. *For any solution F faithful to the final clustering C^g, the cost of the algorithm is less than* 48 $\texttt{cost}(F)$.

Using Theorem 3.6, we can get the final approximation factor of 96 by choosing the feasible and faithful solution F in 3.6 to be the forest F in Theorem 3.5.

For proving Theorem 3.6, we now start off with a solution faithful to gluttonous, specifically, the solution promised by Theorem 3.5. We define a Steiner forest instance $I^{(t)}$ on the remaining active supernodes for each step during the run of gluttonous and maintain a candidate solution to this instance. As we keep modifying the instance $I^{(t)}$, we remove a set of edges in the candidate solution at every step and intuitively, we want to show that the cost of edges which gluttonous *buys* can be connected to the cost of removed edges.

Formally, we start with the initial trivial clustering and the instance $I^{(t)}$ is just the given problem. The candidate solution we use is the solution F from Theorem 3.5. If $C^{(t)}$ is the clustering during any step, we construct a Steiner forest instance $I^{(t)}$ over the supernodes in $I^{(t)}$. We connect each active supernode in $I^{(t)}$ with a supernode which contains a for some terminal pair $\{u, \bar{u}\}$, $u \in S_1$ and $\bar{u} \in S_2$. The distance between two supernodes is given using the puncutured metric $d_{M \setminus C}$. The pair of supernodes which get connected from any step t till the end can be found out from this instance $I^{(t)}$. We inductively maintain a forest $F^{(t)}$ over the instance $I^{(t)}$ such that

(I) $F^{(t)}$ is a feasible solution to $I^{(t)}$, and

(II) $F^{(t)}$ maintains connectivity in F: for $\{u, v\}$ in same tree in F, the corresponding supernodes S_u and S_v lie in same tree in $F^{(t)}$.

We start with F as solution $F^{(0)}$ to initial clustering $C^{(1)}$. It is easy to see that both invariants are satisfied. To inductively update $F^{(t)}$ at each step, we need to (i) merge two supernodes getting connected into 1 supernode and (ii) remove edges in $F^{(t)}$ to maintain the tree structure along with the invariants. Note that whenever two supernodes are connected, we connect two terminals in the same tree in F, and therefore, they would always lie in the same tree in $F^{(t)}$ (invariant II). Thus, merging two supernodes in $F^{(t)}$ creates a cycle (might be a multi-edge based self loop), and we remove an edge in the resulting graph to maintain a tree. We further control the *cost* of $F^{(t)}$ by short-cutting degree 2 Steiner vertices (inactive supernodes) in $F^{(t)}$. The pseudo-code to update $F^{(t)}$ is shown in Algorithm 2.

We maintain a *potential* value for edges in $F^{(t)}$ where the initial potential in $F^{(0)}$ is $\psi(e) := d_M(e)$. Whenever, we shortcut an edge, we change the potential of the resulting edge to be the sum of all shorted edges: $\psi(e) = \psi(e') + \psi(e'')$.

Algorithm 2 `UpdateForest`$(C^{(t)}, S_1, S_2)$

1: Merge S_1 and S_2 into a node S in the tree T containing both.
2: If S becomes inactive and $\texttt{degree}(S) = 2$, short incident edges.
3: Delete edge with highest potential in the unique cycle C formed after step 2.
4: While there are inactive supernodes S' with degree 2, short cut incident edges for S'.

Since, we only shortcut Steiner nodes in $I^{(t)}$ (inactive supernodes), the updated forest $F^{(t)}$ is still a feasible solution for $I^{(t)}$. Also, the connectivity structure is always maintained as we only remove edges in a cycle to maintain a tree structure, thereby invariant (II). Also, it is easy to see that the degree of Steiner nodes in $F^{(t)}$ is at least 3 due to steps 2 and 4. Further, there are at most two iterations of step 6 during each update.

Note that each edge in $F^{(t)}$ is obtained by short-cutting several ground edges in M and the potential is the sum of all these edges. The key idea now is to look at each tree T in F and observe that eventually all edges in T would be deleted as we terminate with no active supernode. Also, we either short-cut edges or remove them at each step. So, the total potential of deleted edges and edges in $F^{(t)}$ is always the same as the total potential of the starting tree T. Therefore, $\sum_{e\in \text{del}(\infty)} \psi(e) = \texttt{cost}(T)$ where del(∞) denotes the set of edges deleted in all updates for tree T. The main lemma used in the proof now *charges* the merging cost Δ_t at step t to the potential of some deleted edges in del(∞). Summing up these costs over all trees gives us the final approximation factor.

Specifically, one can show that, if we merge supernodes in $F^{(t)}$ which are in tree T in F, there are at least $N_t/8$ edges in del(∞) which $\psi(e) \geq \Delta_t/6$. Thus, we can find a perfect matching σ in a bipartite graph with merge steps and del(∞) as edge sets, with each iteration connected to deleted edge with $\psi(e) \geq \Delta_t/6$. The total merging cost is then given as $\sum_{T\in F} \Delta_t = \sum_{T\in F} \sum_{e\in \text{del}_T(\infty), e=\sigma(t)} \Delta_t \leq \sum_{T\in F} \sum_{e\in \text{del}_T(\infty)} 48\psi(e) = 48 \sum_{T\in F} \texttt{cost}(T) \leq 96\texttt{cost}(F^*)$.

4 Cost Shares for Steiner Forest

A cost sharing method χ is an algorithm that maps triples Q = $(I, (u, \bar{u}))$ to non negative costs. This can be seen as a charging function which divides the cost of connecting terminal pairs in I among the users. Ideally, the sum of such cost shares should at least be equal to the total cost of the connections (*budget balanced*). In addition, if we have a *cross-monotonic* χ, then it leads to a group-strategy proof incentive compatible mechanism (Könemann et al., 2005). In other words, this charging function forces each agent to reveal their true preferences for their terminal pair. In this paper, a *timed* version of the gluttonous algorithm is used for obtaining χ. The timed version divides the execution into stages where all supernodes with merging distance in $[2^i, 2^{i+1}]$ are merged instead of the nearest active supernode pair. Such methods also provide approximations for stochastic optimization problems for Steiner Forest. Some useful definitions for this are:

Definition 4.1. *A cost-sharing method is called budget balanced if, $\sum_{(u,\bar{u})\in D} \chi(Q) \leq cost(F^*)$ where F^* is an optimal solution to instance I. This is α-approximation if $\sum_{(u,\bar{u})\in D} \chi(Q) \leq \alpha \cdot cost(F^*)$. Further, χ is cross monotonic if the cost-share of each individual triple never decreases as the the set of triples shrink.*

Uni-strict cost sharing schemes had been studied by Gupta et al. (2007), whereas, a strict version had been an open problem. Here we focus on strict cost-shares using the TimedGlut algorithm. χ is defined to be strict w.r.t some algorithm A. Let χ be defined using the TimeGlut algorithm which is γ-approximation (γ = 480). At each stage, we increment the cost-share of each of $(u, \bar{u})$ and $(u', \bar{u}')$ by $\frac{2^{i+1}}{2\gamma}$ (u and u' are active terminals with maximum distance to their mates for a pair of supernodes respectively). If B^i is a collection of supernodes with merging distance in $[2^i, 2^{i+1}]$, TimedGlut algorithm shows that $\sum_i |B^i| \cdot 2^{i+1} \leq \gamma \cdot opt(M, D)$. Thus χ being budget balanced follows. Let A be the timed primal dual algorithm (TimedPD). Also consider a partition $D_1 \cup D_2$ on D and metric M.

Claim 4.2. *χ is β-strict w.r.t algorithm A where β is a constant i.e., If F_1 is the forest returned by $A(D_1)$ and F_2^* is the optimal Steiner Forest on D_2 in metric $M\backslash F_1$ then, $cost(F_2^*) \leq \beta \cdot \sum_{(u,\bar{u})\in D_2} \chi(M, D, (u, \bar{u}))$*

Showing feasibility of the solution obtained by the cost-sharing scheme and constant factor approximation on the cost of edges in F_2 suffices to prove the claim.

Let R be the run of TimedGlut on instance $I = (M, D_1 \cup D_2)$. Run A on D_1 and define i^{th} stage s.t. it lasts for a time interval $[6 \cdot 2^i, 6 \cdot 2^{i+1})$ to get F_1^i. At each stage i, R takes the current clustering C^i and constructs a graph H^i (its nodes are the supernodes in C^i and edges are pairs of active supernodes that had mutual merging distance at most 2^{i+1}). Let P^i be some maximal graph in this H^i and μ^{i+1} be a collection at the end of each stage. For a pair of supernodes if their active terminals both lie in D_1 as well as μ^{i+1}, we say the pair belongs to the same equivalence class. We collapse each equivalence class in P^i

and consider only the pairs that lead to a maximal acyclic set. F_2^i is built by adding edges only corresponding to these pairs ($F_2 = \cup_i F_2^i$ and $F_1 = \cup_i F_1^i$). Define,

1. $P_X^i \leftarrow$ pairs dropped to get acyclic set (dropped edges)
2. $P_G^i \leftarrow$ pairs $(S, S') \in P^i \setminus P_X^i$ s.t. at least one of active terminals of S or S' belongs to D_2 (good edges)
3. $P_B^i \leftarrow$ pairs $(S, S') \in P^i \setminus P_X^i$ s.t. both active terminals of S or S' belong to D_1 (bad edges)

As the forest must remain faithful to the clustering at each stage, for any $S \in C^i$, all terminals in S must lie in the same connected component. It is easy to see that if a pair $(S, S') \in C^i$ belong to the same equivalence class then their active terminals belong to the same collection. As A adds edges between collections when they become tight, they must lie in the same connected component in the final solution. When they belong to different collections, we only add edges when $(S, S') \in P^i \setminus P_X^i$. Thus, for any such pair, the terminals in $S \cup S'$ must lie in the same connected component. Finally in the solution, each terminal pair in D_2 is contained in some supernode. Thus they must be eventually connected in $F_1 \cup F_2$ as well. This shows that the cost-sharing scheme returns a feasible solution.

Theorem 4.3. *The total cost of edges in $\cup_i F_2^i$ is at least 3$\cdot \sum_i |P_B^i| \cdot (2^{i+1} - 2^i)$*

We bound the total cost by the number of bad edges added to F_2. This can be shown by induction. The base case for i = 0, follows trivially. Let the statement be true for i-1. By the primal-dual process no two collections meet during stage i. Also edges that lie in μ^{i+1} do not lie in μ^i. By the length of time interval, distance between any two collections is at least $6 \cdot (2^{i+1} - 2^i)$ (in the punctured metric being considered at this stage). Imagine a ball of radius $3 \cdot (2^{i+1} - 2^i)$ around each collection. Thus, this ball contains edges of length at least $3 \cdot (2^{i+1} - 2^i)$ and are disjoint. An edge connecting two active collections is added to the forest when it becomes tight. So total length of all edges in these balls is bounded by the number of bad edges added, i.e., at least 3$\cdot \sum_i |P_B^i| \cdot (2^{i+1} - 2^i) \geq \frac{3}{2} \cdot \sum_i |P_B^i| \cdot (2^{i+1})$.

Theorem 4.4. *The cost of edges in F_2 is at most $6 \cdot \gamma \cdot \sum_{(u,\bar{u}) \in D_2} \chi(M, D, (u, \bar{u}))$*

By Theorem 4.3 and observing that total cost of bad edges is atmost $\sum_i |P_B^i| \cdot (2^{i+1})$, the cost of good edges is at least ($\frac{1}{3} \cdot cost(F_2)$). For the good edges at stage i, one of the terminals for a considered pair lied in D_2. Thus the terminal pairs in D_2 are charged the cost-share(χ) which is at least $\frac{1}{2 \cdot \gamma}$ as well as the cost for adding a good edge i.e., at least $\frac{1}{6 \cdot \gamma} cost(F_2)$. This completes the proof of the claim.

5 Future Directions

This paper presents two main contributions in the approximation algorithms literature for Steiner forests: a constant factor approximation for a greedy method and the first strict cost share method for Steiner forests. Similarly, one can consider the problem of obtaining a constant factor approximation via a local search based method. Borrowing the intuition from the gluttonous algorithm, we can greedily add *and remove* edges in each iteration. Groß et al. (2017) provide a local search algorithm that is also a constant-factor approximation. They start with any feasible solution. In each step, they find two vertices and add the shortest path between them. Then they drop edges that create a cycle. We could perform local search operations to simply reduce the cost of the forest. However, they show that this could give us local optima with $\Omega(\log n) \cdot OPT$. Instead, they consider locally minimizing a potential function, which is the sum of the cost of the forest *and the width of the forest*.

Looking at the contributions of these two papers, there are many possible future directions:

- **Better approximation factor:** For both the greedy and local search methods, the constant factor obtained are 96 and 69 respectively. Although these are the first papers to show a constant factor approximation, the problem of refining the bound and simplifying the analysis is wide open. Note that the best approximation factor available is $2 - 1/k$ using the primal-dual method from Goemans and Williamson (1995).
- **Dynamic Steiner forest** Here, terminal pairs arrive online and we want to maintain a constant-approximate Steiner Forest while changing the solution by only a few edges in each update. Since, dynamic Steiner tree algorithms have been based on local search, obtaining such a solution for this generalized version can also be obtained through the local search approximation algorithm.
- **Stochastic multi-stage Steiner forest** In the stochastic version we are given a distribution over demands and demand set is revealed in the future. The idea is to use cost-sharing schemes to minimize the total expected cost. It would be interesting to show a better approximation for these problems. (primal-dual methods give an approximation factor of 5).

References

Agrawal, A., Klein, P., and Ravi, R. (1995). When trees collide: An approximation algorithm for the generalized steiner problem on networks. *SIAM Journal on Computing*, 24(3):440–456.

Chen, H.-L., Roughgarden, T., and Valiant, G. (2010). Designing network protocols for good equilibria. *SIAM Journal on Computing*, 39(5):1799–1832.

Goemans, M. X. and Williamson, D. P. (1995). A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24(2):296–317.

Groß, M., Gupta, A., Kumar, A., Matuschke, J., Schmidt, D. R., Schmidt, M., and Verschae, J. (2017). A local-search algorithm for steiner forest. *arXiv preprint arXiv:1707.02753*.

Gupta, A. and Kumar, A. (2015). Greedy algorithms for steiner forest. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 871–878. ACM.

Gupta, A., Kumar, A., Roughgarden, T., et al. (2007). Approximation via cost sharing: Simpler and better approximation algorithms for network design. *Journal of the ACM (JACM)*, 54(3):11.

Könemann, J., Leonardi, S., and Schäfer, G. (2005). A group-strategyproof mechanism for steiner forests. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 612–619. Society for Industrial and Applied Mathematics.