SCS Senior Thesis
Simpler Approximations for the Network Steiner-Tree Problem

Vaidehi Srinivas

May 1, 2020

Research Advisor: Anupam Gupta

Abstract

The Network Steiner Tree problem is a well-known problem in combinatorial optimization. The input is an undirected graph $G_{\text{full}} = (V_{\text{full}}, E_{\text{full}})$ with non-negative edge weights, and a set of terminals, $V \subseteq V_{\text{full}}$. The goal is to output a minimum cost tree $T \subseteq G_{\text{full}}$, that spans all of V. In general, this problem is known to be NP-complete [Kar72], so we study polynomial-time approximation algorithms. One polynomial-time approximation result for this problem is an 11/6-approximation, found by Zelikovsky in 1993 [Zel93]. In 2005, Robins and Zelikovsky found an algorithm that achieves a $1 + \frac{\ln 3}{2} \approx 1.55$-approximation [RZ05]. These algorithms use greedy strategies to make a set of incremental improvements to the result. We work out the details of an idea due to Deeparnab Chakrabarty [Cha18], and investigate whether the algorithms can be simplified by reducing to submodular function optimization under knapsack constraints, and applying Sviridenko's 2004 cost-benefit greedy algorithm, which is known to give a $1 - 1/e$ approximation [Svi04].

Contents

1 Introduction **3**

2 The Network Steiner Tree Problem **4**

2.1 Introductory Definitions 4

2.2 A Simple 2-Approximation 5

2.3 An 11/6-Approximation 6

2.4 A $1+\frac{\ln 3}{2}$-Approximation 8

3 Submodular Function Maximization **10**

3.1 Approximation for Submodular Maximization Under a Cardinality Constraint 11

3.2 Approximation for Maximization Under a Knapsack Constraint 13

4 Steiner-Tree Approximations Using Submodular Maximization **18**

4.1 A Helpful Lemma about MSTs and Submodularity 18

4.2 A Simplified $(1+\ln 2)$-Approximation 20

4.3 A Simplified $(1+\frac{\ln 3}{2})$-Approximation 23

4.3.1 Relationship to Robins and Zelikovsky's Algorithm 27

5 Future Work **28**

1 Introduction

The Network Steiner Tree problem is a well-known problem in combinatorial optimization. An instance of this problem is an undirected weighted graph $G_{\text{full}} = (V_{\text{full}}, E_{\text{full}})$ with non-negative edge weights, and a set of terminals $V \subseteq V_{\text{full}}$. A Steiner Tree is a tree $T \subseteq G_{\text{full}}$ that spans all of V, and is allowed to use any subset of other vertices in V_{full}. The other vertices that it uses are called *Steiner points*. In the Network Steiner Tree problem, given a graph G_{full} and set of terminals V, we want to find the minimum cost Steiner Tree. In general, the Network Steiner problem is known to be NP-complete [Kar72], so we look for polynomial-time approximation algorithms.

Approximation	Citation
2	Minimum Spanning Tree
11/6	Zelikovsky, 1993, [Zel93]
$1 + \frac{\ln 3}{2}$	Robins and Zelikovsky, 2005, [RZ05]
$\ln(4)$	Byrka, Grandoni, Rothvoß, and Sanità, 2010, [BGRS10]

Table 1: Approximation ratios of previous polynomial-time Steiner Tree approximation algorithms

In Table 1, we summarize known approximation algorithms for the Steiner Tree problem. These algorithms are discussed in Section 2.

In this thesis, we work out the details of an idea due to Deeparnab Chakrabarty, who noted that the algorithms in [Zel93] and [RZ05] can be viewed as greedy maximization of submodular functions [Cha18]. We do not look at the results in [BGRS10] in detail, but we discuss them briefly in the future work section (Section 5).

Submodularity is a property of functions over subsets of a *ground set* V. We denote the power set of V (set of all subsets of V) as $\mathcal{P}(V)$. A function $f : \mathcal{P}(V) \to \mathbb{R}$ is submodular, if for all sets A, B such that $A \subseteq B \subseteq V$, and elements $e \in V \setminus B$, we have that $f(A \cup \{e\}) - f(A) \geq f(B \cup \{e\}) - f(B)$. The problem of finding a set S to maximize $f(S)$, subject to constraints on S, has been well studied for various types of constraints. Specifically, for knapsack constraints, which are relevant for Steiner Tree minimization, optimal polynomial-time approximations are known. These results are dicussed in Section 3.

In this thesis, we present simpler algorithms based on the work [Zel93] and [RZ05], using algorithms for approximate maximization of submodular functions. Our simpler algorithm based on [Zel93] achieves an approximation ratio of $1 + \ln 2 \approx 1.693$, which is better than the original of $11/6 = 1.833$. Our simpler approach based on [RZ05] meets their approximation ratio of $1 + \frac{\ln 3}{2} \approx 1.549$. We also show that our simplification actually results in a simpler presentation of the same algorithm that Robins and Zelikovsky present. These results are discussed in Section 4.

2 The Network Steiner Tree Problem

2.1 Introductory Definitions

We refer to the graph in which we are finding a Steiner Tree as $G_{\text{full}} = (V_{\text{full}}, E_{\text{full}})$. We say that the terminals are a subset $V \in V_{\text{full}}$, and refer to an associated graph $G = (V, E)$, where the distance of each edge (u, v) in G is the shortest path distance between the terminals u and v in G_{full}. We assume that the weight of the edge (u, v) in G_{full} is the shortest path distance between u and v in G_{full}. For graphs that do not obey this, we can run our approximation algorithms on a graph that represents the shortest paths, and transform back into our original graph.

Definition 2.1 (Full Component, k-restriction)**.** *A* full component *in G_{full} is a subtree of G_{full} such that all leaves are terminals, and all internal nodes are non-terminals. A* k-restricted full component *is a full component with at most k terminals.*

Lemma 2.2 (Number of Steiner Points bounded)**.** *Any k-restricted full component has at most $k - 2$ internal nodes.*

Proof. We can assume that any internal node in a full component K has degree at least 3. Suppose that there was an internal node r in K of degree 2, adjacent to nodes u and v. Since the edge (u, v) in G_{full} has distance equal to the shortest path between u and v, we know that $\text{cost}(u, v) \leq \text{cost}(u, r) + \text{cost}(r, v)$. This means that if we remove r, and connect u and v directly, we get a tree of the same or lower cost.

Since all internal nodes have degree at least 3, the number of internal nodes is upper bound by two less than the number of leaves. This means that any k-restricted full component can have at most $k - 2$ internal nodes. $\square$

We define Z^* to be the set of full components that make up the optimum Steiner Tree, T^*, in G_{full}. Note that any Steiner Tree can be split into full components by splitting at every terminal in the tree.

We define $Z^*_{(k)}$ to be the set of full components containing at most k terminals, that make up the optimum k-restricted Steiner Tree, $T^*_{(k)}$, in G_{full}. A k-restricted Steiner Tree is a Steiner Tree in which every full component is k-restricted.

Definition 2.3 (The cost function)**.** *For any graph, G,* $\text{cost}(G)$ *is the sum of the weights of the edges in G. For a full component z,* $\text{cost}(z)$ *is the cost of the minimum cost tree connecting all of the terminals in z. For a set of full components Z,* $\text{cost}(Z)$ *is the sum of* $\text{cost}(z)$ *for $z \in Z$.*

Definition 2.4 (Notation for Minimum Spanning Trees)**.** *Let* $\text{MST}(G)$ *be the minimum spanning tree of all vertices in G. Then we can define* $\text{mstcost}(G) = \text{cost}(\text{MST}(G))$.

Definition 2.5 (Graph Contraction $G[(e, c)]$)**.** *Let $G = (V, E)$ be a weighted graph, and (e, c) be a pair of an edge $e \in E$, and an associated cost $c \in \mathbb{R}^+ \cup \{0\}$. We say that $G[(e, c)]$ is the resulting graph if we set the weight of edge e in G to the minimum of its current weight and c. For a set $S = \{p_1, \ldots, p_n\}$ of such pairs, $G[S]$ is $G[p_1] \ldots [p_n]$. For a collection of such sets $\mathcal{Z} = \{S_1, \cdot, S_n\}$, $G[\mathcal{Z}]$ is $G[S_1] \ldots [S_n]$.*

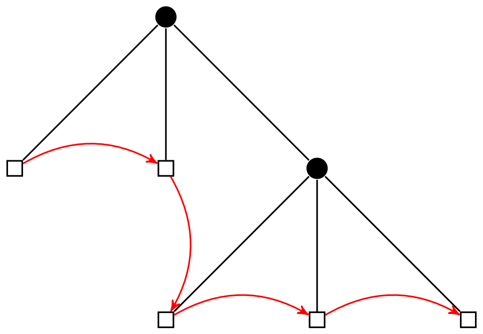


Figure 1: Constructing Spanning Tree in G (red) from T^* (black)

Theorem 2.6 (The effects of k-restriction)**.** *Let ρ_k be the smallest ratio of $\frac{\operatorname{cost}(T^*)}{\operatorname{cost}\left(T^*_{(k)}\right)}$ over all metrics G_{full}. In [BD97], Borchers and Du show that for $k = 2^r + s$, where $0 \leq s < 2^r$,*

$$\rho_k = \frac{r2^r + s}{(r+1)2^r + s}.$$

This also means that for $k = 2^r + s$,

$$\frac{\operatorname{cost}\left(T^*_{(k)}\right)}{\operatorname{cost}\left(T^*\right)} = \frac{(r+1)2^r + s}{r2^r + s} = 1 + \frac{2^r}{r2^r + s} \leq 1 + \frac{1}{r},$$

so setting $k_\varepsilon = 2^{\frac{1}{\varepsilon}}$ ensures that $\rho_{k_\varepsilon} \leq 1 + \varepsilon$.

2.2 A Simple 2-Approximation

One algorithm for approximately finding the minimum cost Steiner Tree in G_{full} is to take the minimum spanning tree of the associated graph on just the terminals, G. This is a 2-approximation.

Theorem 2.7 (MST is a 2-approximation)**.** *For a graph G_{full} and associated graph on the terminals, G,*

$$\operatorname{mstcost}(G) \leq 2 \cdot \operatorname{cost}(T^*).$$

Proof. Fix the tree T^* in G_{full}. We find a spanning tree, T', in G with cost at most $2 \cdot \operatorname{cost}(T^*)$. We do this as illustrated in Fig. 1. Starting at one of the terminals t_0 arbitrarily (shown as squares), we do a depth-first traversal of T^*. Every time we see a new terminal t_i, we add the edge from t_i to the previous terminal seen t_{i-1} (shown in red) to T'. The cost of this edge corresponds to the cost of the shortest path in G_{full} between t_{i-1} and t_i.

If the shortest path between t_{i-1} and t_i in G_{full} was the path in T^*, for every i, then T' would have distance at most equivalent to traversing every edge in T^* twice, once out and once back in the depth first traversal. If the shortest path between the subsequent terminals is not in T^*, then T' will have even smaller cost. Either way,

$$\operatorname{cost}(T') \leq 2 \cdot \operatorname{cost}(T^*). \qquad \square$$

2.3 An 11/6-Approximation

In [Zel93], Zelikovsky presents an algorithm and shows that it is an 11/6 approximation for the Network Steiner Problem. We present an overview of the algorithm and argument.

Definition 2.8 (The save function)**.** *We define the function* $\text{save}_G(Z)$*, where* G *is associated with* G_{full}*, and* Z *is a set of full components in* G_{full}*. For each* $z \in Z$*, let* S_z *be the set of edges between pairs of terminals in* Z*, each associated with the cost 0. Let* S_Z *be the set containing all* S_z*. Now we can define*

$$\text{save}_G(Z) = \text{mstcost}(G) - \text{mstcost}(G[S_Z]).$$

Algorithm 1 Zelikovsky's 11/6 Approximation

```
Input:  $G_{\text{full}}$, $S$
 1: $F \leftarrow$ the graph $G$ induced by taking shortest path distance over the terminals in $G_{\text{full}}$
 2: for every triple (full component on 3 terminals) $z$ do
 3:     find the internal node $v$ that minimizes the cost of $z$
 4:     $v(z) \leftarrow v$
 5:     $\text{cost}(z) \leftarrow$ the cost of the tree with $v$ and $z$
 6: end for
 7: $W \leftarrow \varnothing$
 8: while There exists a triple $z$: $\text{save}_z(F) - \text{cost}(z) > 0$ do
 9:     Find $z^*$ that maximizes $\text{save}_z(F) - \text{cost}(z)$
10:     $W \leftarrow W \cup \{v(z)\}$
11:     $S_z \leftarrow$ a set containing pairs of edges in $z$ with cost 0
12:     $F \leftarrow F[S_z]$
13: end while
14: return The MST in $G_{\text{full}}$ of the set $S \cup W$
```

At a high level, Algorithm 1 considers the set of all triples, or 3-restricted full components (line 2). At each step, it greedily chooses the triple z that maximizes $\text{save}_F(z) + \text{cost}(z)$ (line 9). This is useful as $\text{save}_F(z)$ is how much our MST on the terminals improves if we sink the cost to connect the terminals in z, and $\text{cost}(z)$ is the cost to connect these terminals. This heuristic allows us to best pick the internal nodes to include in our tree. In general, for a set of triples Z, $\text{save}_F(Z) + \text{cost}(Z)$ is not a perfect heuristic, since multiple triples may share edges, so the cost of including z_1 and z_2 may be less than $\text{cost}(z_1) + \text{cost}(z_2)$.

In the analysis, Zelikovsky presents two lemmas that help get the approximation result. The first lemma says that a sequence of greedily chosen triples does at least half as well as any set of triples. That is, for a set of greedily chosen triples H, and any set of triples Z:

$$\text{save}_G(H) + \text{cost}(H) \geq \frac{1}{2}\left(\text{save}_G(Z) + \text{cost}(Z)\right). \quad (1)$$

They make this argument by thinking each of each of the greedily chosen triples $\{a, b, c\}$ as a pair of 0 weight edges $(a, b), (b, c)$. Then they show that for each edge e, either contracting e does not affect how well Z does in the resulting graph, or there is a triple $z \in Z$ such that contracting e causes the rest of Z to do at least as well as contracting z. That is:

$$\text{save}_{F[(a,b)][(b,c)]}\,(Z \setminus \{z_1, z_2\}) + \text{cost}(\{a,b,c\}) \geq \text{save}_{F[z_1][z_2]}\,(Z \setminus \{z_1, z_2\}) + \text{cost}(z_1) + \text{cost}(z_2).$$

Note that F is the state of the contracted graph when we consider this triple.

This allows them to conclude that when they choose a triple at line 9, it decreases the potential gain of the optimum set of triples by at most two triples. Since they chose the triple that maximizes save + cost, they will decrease the potential gain by at most 2 times what they increased the actual gain by. This inductively shows Eq. (1).

The details of the proof rely on arguments about the MSTs in the various contracted graphs, and can be found in the paper [Zel93].

The second lemma says there exists a set of triples that gets us relatively close to the optimum tree. More specifically, there is a set of triples Z such that

$$3\,(\text{mstcost}(G) - \text{save}_G(Z) + \text{cost}(Z)) \leq 5 \cdot \text{cost}(T^*). \tag{2}$$

They make this argument by considering T^* to be a full binary tree, with all terminals at the leaves. They then associate a triple of leaves with each internal node in the tree, and bound the cost of that triple by the cost to connect the triple within the tree. They then bound the sum of the save values of all such triples, and show that the triples can be partitioned into 3 sets such that the triples do not affect each other's save values within a single set. This allows them to argue that there is at least one set that satisfies Eq. (2). The proof in [Zel93] is much more complete, and goes into the details of how the triples are defined, and their save values can be bounded.

Finally, using Eq. (1) and Eq. (2), and Theorem 2.7, we can conclude that contracting a greedy set of triples H will give us an 11/6 approximation. Let Z^* be the best set of triples, and T' be the tree that we output.

$$\begin{aligned}
\text{cost}(T') &\leq \text{mstcost}(G) - \text{save}_G(H) + \text{cost}(H) \\
&\leq \text{mstcost}(G) - \frac{1}{2}\,(\text{save}_G(Z^*) + \text{cost}(Z^*)) && \text{by Eq. (1)} \\
&\leq \frac{1}{2} \cdot \text{mstcost}(G) + \frac{5}{6} \cdot \text{cost}(T^*) && \text{by Eq. (2)} \\
&\leq \frac{11}{6} \cdot \text{cost}(T^*). && \text{by Theorem 2.7}
\end{aligned}$$

This proof relies on many technical arguments about the structure of the minimum spanning trees and the optimum Steiner Tree. In Section 4.2, we show that the function save is submodular. This means that we can get a simpler approximation algorithm based on the save function by applying an algorithm for submodular function maximization, and avoid many of the technical details of Zelikovsky's proof.

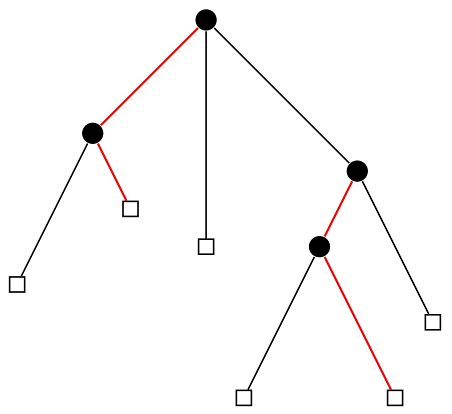


Figure 2: One way to connect Steiner points to terminals

2.4 A $1 + \frac{\ln 3}{2}$-Approximation

In [RZ05], Robins and Zelikovsky present an algorithm and show that it is a $1 + \frac{\ln 3}{2}$-Approximation for the Network Steiner Problem. We present an overview of the algorithm and the argument.

Definition 2.9 (The loss function)**.** *Given a full component K, and the minimum cost tree T_K connecting K, $\mathrm{Loss}(K)$ is defined to be the minimum cost subset of edges from T_K such that all internal nodes are connected to terminals.*

We also define the shorthand $\mathrm{loss}(K) = \mathrm{cost}(\mathrm{Loss}(K))$ for convenience.

Definition 2.10 (The imp function)**.** *We define the function $\mathrm{imp}_G(Z)$ (for improvement), where G is associated with G_{full} and Z is a set of full components in G_{full}.*

For each $z \in Z$, let T_z be the optimal tree connecting z in G_{full}. Let T'_z be T_z where all edges in $\mathrm{Loss}(K)$ are set to have distance 0. Now define S_z to be the set edges (u, v) between terminals of z, associated with the distance between u and v in T'_z.

We define S_Z to be the set of S_z for each $z \in Z$. Now we can define

$$\mathrm{imp}_G(Z) = \mathrm{mstcost}(G) - \mathrm{mstcost}(G[S_Z]).$$

Lemma 2.11 (Loss is at most $\frac{1}{2}$ cost)**.** *For a full component K, $\mathrm{Loss}(K) \leq \frac{1}{2} \cdot \mathrm{cost}(K)$.*

Proof. Let T_K be the optimum Steiner tree connecting the terminals of K. If T_K has no Steiner points, then $\mathrm{loss}(K) = 0$, and the lemma holds trivially.

Otherwise, we show that there is a way to connect the Steiner points of T_K to terminals, as shown in Fig. 2. Root T_K arbitrarily at a Steiner point. We construct a set of edges S (shown in red) that connects all Steiner points in T_K to terminals (shown as circles). Note that every leaf is a terminal. For each Steiner point in T_K, include the shortest edge to one of its children in S. Inductively, this must connect every Steiner point to at least one terminal.

By Lemma 2.2, we know that every Steiner point in T_K must have degree at least 3. This means that once we root the tree, each Steiner point has at least two children. Every time we include an edge in S, we must be leaving at least one edge that is at least as long out of S for the other children. This means that $\text{cost}(S) \leq \frac{1}{2} \cdot \text{cost}(T_K)$. Since $\text{Loss}(K)$ is the minimum cost way to connect the Steiner points in K to terminals of K, this gives us that $\text{loss}(K) \leq \frac{1}{2} \cdot \text{cost}(K)$. □

Algorithm 2 Robins and Zelikovsky's $1 + \frac{\ln 3}{2}$ Approximation

```
Input: $G_{\text{full}}$, $S$
1: $T \leftarrow \text{MST}(G)$ for $G = (V, E)$ associated with $G_{\text{full}}$
2: $H \leftarrow V$
3: while There is a $k$-restricted full component $K$ with at least 3 terminals such that $\text{imp}_T(K) - \text{loss}(K)/\text{loss}(K) > 0$ do
4:     Find the component $K$ with at least 3 terminals that maximizes $\text{imp}_T(K) - \text{loss}(K)/\text{loss}(K)$
5:     $H \leftarrow H \cup$ the vertices in $K$
6:     $K_S \leftarrow$ the subgraph of $G$ connecting the terminals of $K$ with shortest path distance in $G_{\text{full}}$ if edges in $\text{Loss}(K)$ are set to weight 0
7:     $T \leftarrow \text{MST}(T \cup K_S)$
8: end while
9: return $\text{MST}(H)$
```

At a high level, Algorithm 2 considers the set of all k-restricted full components. At each step, it greedily chooses the component K that maximizes $\text{imp}_T(K)/\text{loss}(K)$, such that K satisfies $\text{imp}_T(K) - \text{loss}(K) > 0$. It then updates T by recalculating the shortest path distances between terminals as though all connections in $\text{Loss}(K)$ have cost 0. That is, it assumes that we have sunk the cost to add $\text{Loss}(K)$ to our tree, and calculates the additional cost of each connection.

In the analysis in [RZ05], they argue that when the loop guard at 3 returns false, $\text{cost}(T) \leq \text{cost}\left(T^*_{(k)}\right)$. This is a lower bound on the successive imp of the greedily chosen components. They also show that the total loss of the greedily chosen components can be upper bounded by $\text{loss}\left(T^*_{(k)}\right) \cdot \ln\left(1 + \frac{\text{mstcost}(G) - \text{cost}\left(T^*_{(k)}\right)}{\text{loss}\left(T^*_{(k)}\right)}\right)$. These facts and Lemma 2.11 allow us to get that the tree returned by Algorithm 2 is a

$$\frac{\text{cost}\left(T^*_{(k)}\right)}{\text{cost}\left(T^*\right)} \cdot \left(1 + \frac{1}{2} \cdot \ln\left(\frac{4}{\text{cost}\left(T^*_{(k)}\right) / \text{cost}\left(T^*\right)} - 1\right)\right)$$

approximation. By Theorem 2.6, we know that as $k \to \infty$, $\text{cost}\left(T^*_{(k)}\right) / \text{cost}\left(T^*\right) \to 1$, and the approximation ratio approaches $1 + \frac{\ln 3}{2}$.

The details of the analysis are technical, and we do not go over them here. However, in Section 4.3, we give a simpler algorithm and analysis achieving the same approximation ratio. In our simplified approach, we show that imp_G is a submodular function, and we apply an algorithm to approximately maximize a submodular function with respect to a knapsack constraint, where we are maximizing the function imp_G over the set of k-restricted full components, with the cost function loss over the components. We then argue that the simplified algorithm is actually identical to the original.

3 Submodular Function Maximization

Submodularity is a property of functions that go from subsets to reals. That is, we are looking at functions of the form $f : \mathcal{P}(V) \to \mathbb{R}$ for a fixed finite *ground set* V, where $\mathcal{P}(V)$ is the *power set* of V (the set containing all subsets of V).

Definition 3.1 (Discrete Derivative)**.** *For a function $f : \mathcal{P}(V) \to \mathbb{R}$, a set $S \subseteq V$, and an element $e \in V$, the* discrete derivative *of f at S with respect to e is defined as*

$$\Delta_f(e|S) = f(S \cup \{e\}) - f(S).$$

We can extend a similar definition to sets. For a set $S \subseteq V$ and $A \subseteq V$, we define

$$f_S(A) = f(S \cup A) - f(S).$$

Definition 3.2 (Submodularity)**.** *A function $f : \mathcal{P}(V) \to \mathbb{R}$ is* submodular*, if for all sets A, B where $A \subseteq B \subseteq V$, and $e \in V \setminus B$*

$$\Delta_f(e|A) \geq \Delta_f(e|B). \tag{3}$$

Equivalently we can say that f is submodular *if for every $A \subseteq V$ and $B \subseteq V$,*

$$f(A \cap B) + f(A \cup B) \leq f(A) + f(B). \tag{4}$$

Proof. We show that definitions Eq. (3) and Eq. (4) are equivalent.

First assume that definition Eq. (3) holds for $f : \mathcal{P}(V) \to \mathbb{R}$. Now fix sets A and B, $A, B \subseteq V$. Let $b_1, \ldots, b_k$ be the elements of $B \setminus A$, and B_i be the set containing $b_1, \ldots, b_i$.

$$\begin{aligned} &f(A \cap B) + f(A \cup B) \\ &= f(A \cap B) + f(A \cap B) + f_{A \cap B}(A \setminus B) + f_A(B \setminus A) \\ &= f(A \cap B) + f_{A \cap B}(A \setminus B) + f(A \cap B) + \sum_{i=1}^{k} \Delta_f\left(b_i | A \cup B_i\right). \end{aligned} \tag{5}$$

We get these two equivalences by thinking of the marginal value added of each of these subsets of $A \cup B$. Now, from Eq. (5), we get that

$$\begin{aligned} &\leq f(A \cup B) + f_{A \cap B}(A \setminus B) + f(A \cap B) + \sum_{i=1}^{k} \Delta_f\left(b_i | (A \cap B) \cup B_i\right) && \text{by Eq. (3)} \\ &= f(A) + f(B). \end{aligned}$$

This shows that definition Eq. (3) implies definition Eq. (4).

Now, assume that definition Eq. (4) holds for $f : \mathcal{P}(V) \to \mathbb{R}$. Fix sets $A \subseteq B \subseteq V$, and $e \in V \setminus B$. By

Eq. (4) we know that

$$\begin{aligned} f\big((A \cup \{e\}) \cap B\big) + f\big((A \cup \{e\}) \cup B\big) &\le f(A \cup \{e\}) + f(B) \\ f(A) + f(B \cup \{e\}) &\le f(A \cup \{e\}) + f(B) && \text{since } A \subseteq B,\, e \notin B \\ f(e|B) &\le f(e|A). \end{aligned}$$

This shows that Eq. (3) implies Eq. (4). □

Definition 3.3 (Monotone)**.** *A function $f : \mathcal{P}(V) \to \mathbb{R}$ is* monotone *if for all sets A, B such that $A \subseteq B \subseteq V$, we have $f(A) \le f(B)$.*

Note that for a monotone function f, we do not need to require that $e \in V \setminus B$ in definition Eq. (3) of submodularity, as for $e \in B$, we have that

$$\Delta_f(e|B) = f(B \cup \{e\}) - f(B) = 0,$$

and by monotonicity we have that

$$\Delta_f(e|A) = f(A \cup \{e\}) - f(A) \ge 0,$$

so we trivially get that $\Delta_f(e|A) \ge \Delta_f(e|B)$ in this case.

In submodular function maximization, we are interested in algorithms that given a monotone submodular function $f : \mathcal{P}(V) \to \mathbb{R}$, and a group set V, finds a set S that maximizes $f(S)$, such that S meets certain constraints. Some examples of constraints that we can consider are a constraint on the cardinality of S, or a knapsack constraint on S (explained in more detail in Section 3.2). Since it can be computationally infeasible to solve these problems exactly, we look at algorithms that find approximate solutions in polynomial time and a polynomial number of calls to a black-box oracle for f.

3.1 Approximation for Submodular Maximization Under a Cardinality Constraint

In this problem, given a monotone submodular function $f : \mathcal{P}(V) \to \mathbb{R}$, we want an algorithm that returns a set $S \subseteq V$ that approximately maximizes $f(S)$, such that S is subject to a cardinality constraint. That is, we require that $|S| \le K$ for some fixed K.

We describe the polynomial-time algorithm due to Nemhauser et al. in [NWF78], following Horel's presentation in [Hor]. When called on a function f, ground set V, and cardinality constraint K, this algorithm returns a set that is a $\left(1 - \frac{1}{e}\right)$-approximation of the best set of size K. We also extend the proof to show that if we relax the cardinality constraint of the approximation, and consider sets of size upto αK, we will return a set with value at least $\left(1 - \frac{1}{e^\alpha}\right)$ times the optimal set of size K. This is useful when you can think of f as a value function, and K (the cardinality constraint) as an associated cost. The α relaxation allows us to tune our cost versus our benefit.

In [Fei98], Feige showed that unless $\mathsf{P} = \mathsf{NP}$, there is no polynomial-time algorithm that can approximate maximum k-cover to a better ratio than $1 - \frac{1}{e}$. Maximum k-cover is a problem in which you are given a collection of subsets over a ground set S, and you must find the k subsets that have the largest union.

Since maximum k-cover is a special case of submodular maximization with a cardinality constraint, we get that unless $\mathsf{P} = \mathsf{NP}$, there is no polynomial time algorithm with approximation ratio greater than $1 - \frac{1}{e}$.

Algorithm 3 The Greedy Cardinality Algorithm

Input: V, K, black-box routine for f

```
1: $S_0 \leftarrow \varnothing$
2: for $i = 1, \ldots, K$ do
3:     $e_i \leftarrow$ the element $e$ in $V \setminus S_{i-1}$ that maximizes $f(S_{i-1} \cup \{e\})$
4:     $S_i = S_{i-1} \cup \{e_i\}$
5: end for
6: return $S_K$
```

Theorem 3.4 (Greedy Cardinality is a $\left(1 - \frac{1}{e^\alpha}\right)$-approximation)**.** *When called on V, K, and f, the Greedy Cardinality Algorithm (Algorithm 3) outputs a set $S \subseteq V$, $|S| = k$ such that*

$$f(S) \geq \left(1 - \frac{1}{e}\right) f(S^*_{(K)}),$$

where $S^*_{(K)}$ is the optimal set of size K that maximizes f.

More generally, when this algorithm is called on V, αK, and f, it outputs a set $S \subseteq V$, $|S| = \alpha K$ such that

$$f(S) \geq \left(1 - \frac{1}{e^\alpha}\right) f\left(S^*_{(K)}\right).$$

Proof. Consider the sets S_i in the algorithm. Since the algorithm adds exactly one element per iteration, it is clear that $|S_{\alpha K}| = \alpha K$. Now, we show that $S_{\alpha K}$ meets the stated approximation ratio by bounding the improvement of each successive S_i in terms of the value of the optimum set $S^*_{(K)}$.

Consider the difference in the value of $S^*_{(K)}$ and S_{i-1}. That is, $f(S^*_{(K)}) - f(S_{i-1})$. This is at most the marginal value of adding the elements of $S^*_{(K)}$ to S_{i-1}. That is,

$$f(S^*_{(K)}) - f(S_{i-1}) \leq f_{S_{i-1}}(S^*_{(K)}) \leq \sum_{e \in S^*_{(K)}} f_{S_{i-1}}(e). \tag{6}$$

We know that $S_i = S_{i-1} + e_i$, where e_i is element that maximizes $f_{S_{i-1}}$. This gives us that

$$\sum_{e \in S^*_{(K)}} f_{S_{i-1}}(e) \leq \sum_{e \in S^*_{(K)}} f_{S_{i-1}}(e_i) = K \cdot f_{S_{i-1}}(e_i) = K \cdot \left(f(S_i) - f(S_{i-1})\right). \tag{7}$$

Adding $(K-1) \cdot f(S^*_{(K)})$ to both sides of the inequality given by Eq. (6) and Eq. (7) gives us

$$\begin{aligned} K \cdot f(S^*_{(K)}) - f(S_{i-1}) &\leq (K-1) \cdot f(S^*_{(K)}) + K \cdot f(S_i) - K \cdot f(S_{i-1}) \\ f(S^*_{(K)}) - f(S_i) &\leq \frac{K-1}{K} \left(f(S^*_{(K)}) - f(S_{i-1})\right). \end{aligned}$$

Since $f(S_0) = 0$, this gives us inductively that

$$f(S^*_{(K)}) - f(S_i) \leq \left(\frac{K-1}{K}\right)^i \cdot f(S^*_{(K)}).$$

Now, if we run our algorithm for $i = \alpha K$ iterations, we get that

$$\begin{aligned} f(S^*_{(K)}) - f(S_{\alpha K}) &\leq \left(\frac{K-1}{K}\right)^{\alpha K} \cdot f(S^*_{(K)}) \\ &\leq \left(\frac{1}{e}\right)^{\alpha} \cdot f(S^*_{(K)}), \end{aligned} \tag{8}$$

where Eq. (8) follows from $\left(\frac{K-1}{K}\right)^K \leq \frac{1}{e}$.

This allows us to conclude that

$$f(S_{\alpha K}) \geq \left(1 - \frac{1}{e^{\alpha}}\right) f(S^*_{(K)}). \qquad \square$$

Theorem 3.5 (Greedy Cardinality runs in polynomial-time)**.** *For α, the Greedy Cardinality Algorithm Algorithm 3 runs in time $\mathcal{O}(\alpha K \cdot |V|)$ with $\mathcal{O}(\alpha K \cdot |V|)$ calls to the black-box routine for f.*

Proof. The Greedy Cardinality Algorithm runs for $\alpha \cdot K$ iterations. On each iteration, it compares at most $|V|$ different calls to f. In total, the algorithm runs in time $\mathcal{O}(\alpha K \cdot |V|)$ with $\mathcal{O}(\alpha K \cdot |V|)$ calls to the black-box routine for f. $\square$

3.2 Approximation for Maximization Under a Knapsack Constraint

In this problem, we are given a monotone submodular function $f : \mathcal{P}(V) \to \mathbb{R}$, and a cost function over V, $c : V \to \mathbb{R}^+$. We want an algorithm that returns a set $S \subseteq V$, that approximately maximizes $f(S)$, such that S is subject to a knapsack constraint. That is, we require that $\sum_{e \in S} c(e) \leq B$, for some fixed budget B.

Let $S^*_{(B)}$ be the best set of cost at most B. We provide a polynomial-time algorithm due to Sviridenko [Svi04], that called on a function f, ground set V, cost function c, and budget B, returns a set that is a $\left(1 - \frac{1}{e}\right)$-approximation of $S^*_{(B)}$. We also extend Sviridenko's proof to show that if we relax the budget of our approximation, and consider sets of cost upto αB, we will return a set with value at least $\left(1 - \frac{1}{e^{\alpha}}\right)$ times the value of $S^*_{(B)}$. This allows us to tune our cost versus our benefit.

For simplicity, we will abuse the notation of the cost function slightly, and say that for a set S, we refer to the cost of S as $c(S) = \sum_{e \in S} c(e)$.

We begin by proving the following lemma about the greedy algorithm.

Lemma 3.6 (One more iteration of Greedy Knapsack is a $\left(1 - \frac{1}{e^{\alpha}}\right)$-approximation)**.** *Suppose that line 5 in the Greedy Knapsack Algorithm (Algorithm 4) evaluates to false for an element x and set S_i. Then*

$$f(S_i + x) \geq \left(1 - \frac{1}{e}\right) f(S^*_{(B)}).$$

More generally, suppose we run the Greedy Knapsack Algorithm with the budget αB. Then, if line 5 in

Algorithm 4 The Greedy Knapsack Algorithm

```
Input: $V$, $c$, $B$, a black-box routine $f$
 1: $S_0 \leftarrow \varnothing$
 2: $i \leftarrow 0$
 3: while $V \neq \varnothing$ do
 4:     $e_i \leftarrow$ the element $e$ in $V \setminus S_{i-1}$ that maximizes $f(S_{i-1} \cup \{e\})$
 5:     if $c(e) + c(S_{i-1}) \leq B$ then
 6:         $i \leftarrow i + 1$
 7:         $S_i \leftarrow S_{i-1} + e_i$
 8:     end if
 9:     $V \leftarrow V - e_i$
10: end while
11: return $S_i$
```

Algorithm 5 The Partial Enumeration Algorithm

```
Input: $V$, $c$, $B$, a black-box routine $f$
 1: $S_1 \leftarrow$ the set $S \subseteq V$, $|S| < d$, $c(S) \leq B$ that maximizes $f(S)$
 2: $S_2 \leftarrow \varnothing$
 3: for $S \subseteq V$, $|S| = d$, $c(S) \leq B$ do
 4:     $V' \leftarrow V \setminus S$
 5:     $S_G \leftarrow$ output of greedy algorithm, run with $V'$, $c$, $B$, and $f$, and setting $S_0 \leftarrow S$ on line 1
 6:     if $f(S_G) \geq f(S_2)$ then
 7:         $S_2 \leftarrow S_G$
 8:     end if
 9: end for
10: return whichever of $S_1, S_2$ has higher value
```

Note that d is a value based on α that will be set in the analysis *the Greedy Knapsack Algorithm evaluates to false for an element x and set S_i, we know that*

$$f(S_i + x) \geq \left(1 - \frac{1}{e^{\alpha}}\right) f(S^*_{(B)}).$$

Proof. Consider $f\left(S^*_{(B)}\right) - f\left(S_{i-1}\right)$. This is at most the marginal value of adding the elements of $S^*_{(B)}$ to S_{i-1}:

$$f\left(S^*_{(B)}\right) - f\left(S_{i-1}\right) \leq f_{S_{i-1}}\left(S^*_{(B)}\right) \leq \sum_{e \in S^*_{(B)}} f_{S_{i-1}}(e). \tag{9}$$

where the last inequality follows from the submodularity of f. We know that $S_i = S_{i-1} + e_i$, where e_i is the element that maximizes $\frac{f_{S_{i-1}}(e)}{c(e)}$. This gives us that

$$\begin{aligned} \sum_{e \in S^*_{(B)}} f_{S_{i-1}}(e) &\leq \sum_{e \in S^*_{(B)}} c(e) \cdot \frac{f_{S_{i-1}}(e)}{c(e)} \\ &\leq \frac{f(S_i) - f(S_{i-1})}{c(e_i)} \cdot \sum_{e \in S^*_{(B)}} c(e) \\ &\leq \frac{B}{c(e_i)} \cdot \left(f(S_i) - f(S_{i-1})\right). \end{aligned} \tag{10}$$

Adding $\left(\frac{B}{c(e_i)} - 1\right) \cdot f\left(S^*_{(B)}\right)$ to both sides of the inequality given by Eq. (9) and Eq. (10) and simplifying, gives us

$$\begin{aligned} f(S^*_{(B)}) - f(S_i) &\leq \left(1 - \frac{c(e_i)}{B}\right)\left(f(S^*_{(B)}) - f(S_{i-1})\right) \\ &\leq e^{-\frac{c(e_i)}{B}}\left(f(S^*_{(B)}) - f(S_{i-1})\right). \end{aligned}$$

Since $f(S_0) = 0$, inductively we get

$$\begin{aligned} f(S^*_{(B)}) - f(S_i) &\leq \left(\prod_{j=1}^{i} e^{-\frac{c(e_j)}{B}}\right) f(S^*_{(B)}) \\ &\leq e^{-\frac{f(S_i)}{B}} \cdot f(S^*_{(B)}). \end{aligned} \tag{11}$$

Now, if we run our algorithm until line 5 returns false, and include that last element, we know that our final S_i has cost more than αB. This means

$$f(S_i) \geq \left(1 - e^{-\frac{\alpha B}{B}}\right) \cdot f(S^*_{(B)}) \geq \left(1 - \frac{1}{e^{\alpha}}\right) \cdot f(S^*_{(B)}). \qquad \square$$

Theorem 3.7 (Partial Enumeration is a $\left(1 - \frac{1}{e^{\alpha}}\right)$-approximation)**.** *When called with N, c, B, and a black-box routine f, the Partial Enumeration Algorithm (Algorithm 5) outputs a set S, $c(S) \leq B$ such* *that*

$$f(S) \geq \left(1 - \frac{1}{e}\right) f(S^*_{(B)}).$$

More generally, when called with N, c, αB and a black-box routine f, the partial enumeration algorithm outputs a set S, $c(S) \leq \alpha B$ such that

$$f(S) \geq \left(1 - \frac{1}{e^\alpha}\right) f(S^*_{(B)}).$$

Proof. First of all, we only ever add an item e to a set that might be output, if e does not cause the cost of the set to exceed $\alpha \cdot B$. This means that any output will respect the knapsack constraint.

If $|S^*_{(B)}| < d$, then we know that we will find a set that does at least as well on line 1 of the Partial Enumeration Algorithm.

Otherwise, $|S^*_{(B)}| \geq d$. Order the elements in $S^*_{(B)}$ as $e^*_1, e^*_2, \ldots, e^*_j$, where e^*_i is the element in $\{e^*_i, \ldots, e^*_j\}$ that maximizes $f_{\{e^*_1,\ldots,e^*_{i-1}\}}$. Let $S^*_d = \{e^*_1, \ldots, e^*_d\}$. Now consider the iteration where we choose $S = S^*_d$ on line 3.

Note that when we run line 1 in the Partial Enumeration Algorithm, we will effectively be running the Greedy Algorithm on $V \setminus S^*_d$, c, $\alpha B - c(S^*)$, and $f_{S^*_d}$. Since $\alpha B - c(S^*) \geq \alpha(B - c(S^*))$, the set that the Greedy Algorithm returns will do at least as well as the set that it would have returned when setting the budget to $\alpha(B - c(S^*))$.

In the iterations of the greedy algorithm, if we only include elements that are in $S^*_{(B)} \setminus S^*_d$, then we will output the optimal $S^*_{(B)}$, and reach our approximation ratio.

Otherwise, let x^* be the first element in $S^*_{(B)} \setminus S^*_d$ that causes line 5 in the Greedy Algorithm to return false. Now by Lemma 3.6, we know that

$$\begin{aligned}
f_{S^*_d}(S_i + x^*) &\geq \left(1 - \frac{1}{e^\alpha}\right) f_{S^*_d}\left(S^*_{(B)} \setminus S^*_d\right) && \text{by Lemma 3.6} \\
f(S^*_d \cup S_i \cup \{x^*\}) - f(S^*_d) &\geq \left(1 - \frac{1}{e^\alpha}\right) \left(f\left(S^*_{(B)}\right) - f(S^*_d)\right) && \text{by definition of } f_{S^*_d} \\
f(S^*_d \cup S_i) + f_{S^*_d \cup S_i}(x^*) &\geq \left(1 - \frac{1}{e^\alpha}\right) f\left(S^*_{(B)}\right) + \frac{1}{e^\alpha} \cdot f(S^*_d). && (12)
\end{aligned}$$

In the last line, we split the marginal value of x^* from the value of $S^*_d \cup S_i$, and we add $f(S^*_d)$ to both sides.

By the ordering of S^*_d, we know that

$$f_{S^*_d \cup S_i}(x^*) \leq f_{S^*_d}(x^*) \leq f_{S^*_{j-1}}(e^*_j) \qquad \text{for } 1 \leq j \leq d$$

which means that

$$f_{S^*_d \cup S_i}(x^*) \leq \frac{1}{d} \cdot f(S^*_d). \tag{13}$$

Eq. (12) and Eq. (13) together give us that

$$f(S_d^* \cup S_i) \geq \left(1 - \frac{1}{e^\alpha}\right) f\left(S_{(B)}^*\right) + \frac{1}{e^\alpha} \cdot f(S_d^*) - \frac{1}{d} \cdot f(S_d^*).$$

Finally, if we set $d \geq e^\alpha$, then we get that

$$f(S_2) \geq \left(1 - \frac{1}{e^\alpha}\right) f\left(S_{(B)}^*\right)$$

which meets our approximation ratio. $\square$

Theorem 3.8 (The Partial Enumeration Algorithm runs in polynomial-time)**.** *For a fixed constant α, the Partial Enumeration Algorithm Algorithm 5 has runtime and calls to f bounded by $\mathcal{O}\left(|V|^{e^\alpha+3}\right)$.*

Proof. In line 1, the algorithm iterates over fewer than $|V|^d$ sets, and makes fewer than $|V|^d$ calls to f.

In line 3, the algorithm also iterates over fewer than $|V|^d$ sets. For each of these sets, it runs the Greedy Knapsack Algorithm Algorithm 4. The Greedy Knapsack Algorithm runs for at most $|V|$ iterations, and for each iteration it compares at most $|V|$ calls to f. This means that this loop will do $\mathcal{O}\left(|V|^d \cdot |V|^2\right)$ work, and make $\mathcal{O}\left(|V|^d \cdot |V|^2\right)$ calls to f.

In all, since d is set to be the smallest integer such that $d > e^\alpha$, this gives bounds the runtime, and the number of calls to f by $\mathcal{O}\left(|V|^{e^\alpha+3}\right)$. $\square$

4 Steiner-Tree Approximations Using Submodular Maximization

4.1 A Helpful Lemma about MSTs and Submodularity

Definition 4.1 (Contraction Function). *Consider a weighted complete graph $G = (V, E)$, and a collection of sets $\mathcal{D}$, such that each $S \in \mathcal{D}$ is a set of edge-cost pairs: $(e, c) \in E \times (\mathbb{R}^+ \cup \{0\})$.*

Let f be a function $f : \mathcal{P}(\mathcal{D}) \to \mathbb{R}$, such that for a collection of sets $\mathcal{S} \subseteq \mathcal{D}$

$$f(\mathcal{S}) = \text{mstcost}(G) - \text{mstcost}(G[\mathcal{S}]).$$

We call any function that can be defined this way for a graph G and set $\mathcal{D}$ a contraction function*.*

Lemma 4.2 (Contraction Functions are Monotone Submodular). *Let f be a contraction function over a graph G and set $\mathcal{D}$. f is monotone submodular.*

Proof. First, we show that f is monotone. Consider sets $\mathcal{A}, \mathcal{B}$, such that $\mathcal{A} \subseteq \mathcal{B} \subseteq \mathcal{D}$. We know that each edge cost in $G[\mathcal{B}]$ is upper bounded by the edge cost $G[\mathcal{A}]$. This means that the MST of $G[\mathcal{A}]$ has an equal or smaller cost in $G[\mathcal{B}]$. Since $\text{mstcost}(G[\mathcal{B}]) \leq \text{mstcost}(G[\mathcal{A}])$, this gives us

$$f(\mathcal{A}) = \text{mstcost}(G) - \text{mstcost}(G[\mathcal{A}]) \leq \text{mstcost}(G) - \text{mstcost}(G[\mathcal{B}]) = f(\mathcal{B})$$

so f is monotone.

Now, we show that f is submodular. We want to show that for any collections $\mathcal{A}, \mathcal{B} \subseteq \mathcal{D}$ such that $\mathcal{A} \subseteq \mathcal{B}$, and set $Z \notin \mathcal{B}$,

$$\Delta_f(Z|\mathcal{A}) \geq \Delta_f(Z|\mathcal{B}).$$

First we prove this claim for the case where Z only has one element. Then we will extend it to larger sets.

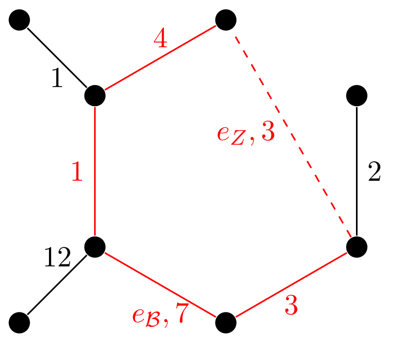
Suppose that the only element of Z is a pair (e_Z, c_Z). We can get the MST in $F[\mathcal{B}][Z]$ by adding a copy of e_Z with cost c_Z to the MST in $F[\mathcal{B}]$, as shown in Fig. 3a. This creates a cycle, C (red in Fig. 3a), and we remove the heaviest edge, $e_\mathcal{B}$, from C. This will result in a tree, T, of cost $\text{mstcost}(F[\mathcal{B}]) - \text{cost}(e_\mathcal{B}) + c_Z$ in $F[\mathcal{B}][z]$, as shown in Fig. 3b.

We start by showing that this is indeed a minimum spanning tree of $F[\mathcal{B}][z]$. Assume for the sake of contradiction that there is a spanning tree T' in $F[\mathcal{B}][z]$ such that

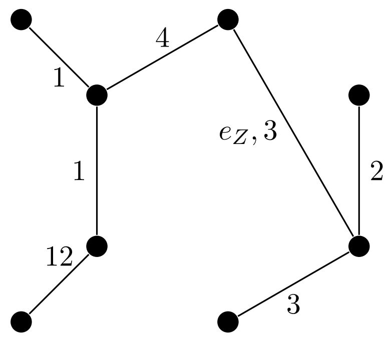
$$\text{cost}(T') < \text{mstcost}(F[\mathcal{B}]) - \text{cost}(e_\mathcal{B}) + \text{cost}(e_Z).$$

We know that T' must include e_Z, otherwise it would not have a lower cost in $F[\mathcal{B}][z]$ than it does in $F[\mathcal{B}]$.

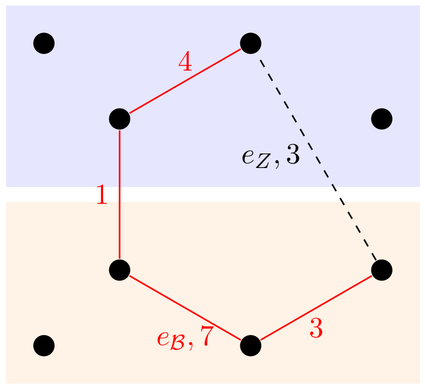
Now suppose we removed e_Z from T', and consider the two resulting connected components. This defines a cut U, V in the graph. Fig. 3c shows a potential such cut in orange and blue. Consider the



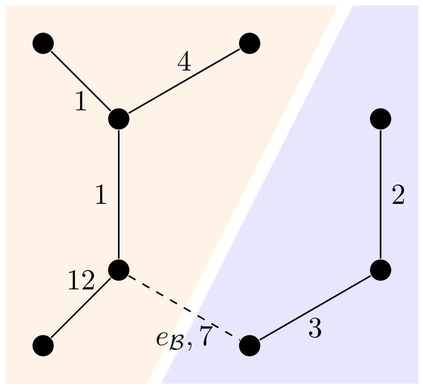
(a) $F[\mathcal{B}]$ (solid) and e_Z (dashed)



(b) $F[\mathcal{B}][(e_Z, 3)]$



(c) The cut in T' induced by e_Z



(d) The cut in $F[\mathcal{B}]$ induced by $e_{\mathcal{B}}$

Figure 3: Submodularity of Contraction Functions

cut U, V in $F[\mathcal{B}]$. We know that at least one edge in C must cross the cut U, V, since C connects the endpoints of e_Z. We add one of these edges to $T' \setminus \{e_Z\}$. Since this edge is in C, it must have cost at most $\text{cost}(e_{\mathcal{B}})$. This means that the resulting spanning tree of $F[\mathcal{B}]$ has cost $< \text{mstcost}(F[\mathcal{B}]) - \text{cost}(e_{\mathcal{B}}) + \text{cost}(e_Z) + \text{cost}(e_{\mathcal{B}}) - \text{cost}(e_Z)$, which is a contradiction. This shows that T is indeed a minimum spanning tree of $F[\mathcal{B}][Z]$, and that $\Delta_f(Z|\mathcal{B}) = \text{cost}(e_{\mathcal{B}}) - \text{cost}(e_Z)$.

Now consider the two connected components you get from removing $e_{\mathcal{B}}$ from $\text{MST}(F[\mathcal{B}])$, illustrated in Fig. 3d. By the MST cut property, $e_{\mathcal{B}}$ must be the minimum cost edge across this cut. Since $\mathcal{A} \subseteq \mathcal{B}$, the edges in $F[\mathcal{A}]$ can only have cost higher than the edges in $F[\mathcal{B}]$, so $\text{cost}(e_{\mathcal{B}})$ is a lower bound on the minimum cost edge across this cut in $F[\mathcal{A}]$ as well. If we were to add e_Z to $\text{MST}(F[\mathcal{A}])$, there would be at least one edge across this cut that we could drop to maintain a spanning tree. Any edge across this cut has cost at least $\text{cost}(e_{\mathcal{B}})$, and we added $\text{cost}(e_Z)$, so this tree in $F[\mathcal{A}][Z]$ will cost at least $\Delta_f(Z|\mathcal{B})$ less than $\text{MST}(F[\mathcal{A}])$. This means that

$$\Delta_f(Z|\mathcal{A}) \geq \Delta_f(Z|\mathcal{B}).$$

This proves our claim when Z only has one element.

Now, to prove the claim for larger sets z, consider sets $\mathcal{A}' \subseteq \mathcal{B}' \subseteq \mathcal{D}$, and $Z' \in \mathcal{D} \setminus \mathcal{B}'$. Let the pairs in Z' be $p_1, p_2, \ldots, p_j$. We can see that

$$\Delta_f(Z|\mathcal{A}') = f_{\mathcal{A}'}(Z)$$

$$\begin{aligned}
&= f_{\mathcal{A}'}(\{p_1, \dots, p_j\}) \\
&= f_{\mathcal{A}'}(\{p_1\}) + f_{\mathcal{A}' \cup \{p_1\}}(\{p_2\}) + \cdots + f_{\mathcal{A}' \cup \{p_1 \dots p_{j-1}\}}(\{p_j\}) \\
&\geq f_{\mathcal{B}'}(\{p_1\}) + f_{\mathcal{B}' \cup \{p_1\}}(\{p_2\}) + \cdots + f_{\mathcal{B}' \cup \{p_1 \dots p_{j-1}\}}(\{p_j\}) \\
&= f_{\mathcal{B}'}(Z) = \Delta_f(Z|\mathcal{B}')
\end{aligned}$$

which extends our claim to finite sets Z of any size. □

Lemma 4.3 (Contraction functions can be evaluated in polynomial-time)**.** *Let f be a contraction function over a graph G and set D. For a set $\mathcal{S} \subseteq \mathcal{D}$, $f(\mathcal{S})$ can be calculated in time $\mathcal{O}(|E| \log |V|)$.*

Proof. $f(\mathcal{S})$ can be found by first calculating $G[\mathcal{S}]$ in time $\mathcal{O}(|E|)$ (we only need to contract the minimum weight version of each edge), and then finding the MSTs of G and $G[\mathcal{S}]$ in time $\mathcal{O}(|E| \log |V|)$. This is a total of $\mathcal{O}(|E| \log |V|)$ time. □

4.2 A Simplified $(1 + \ln 2)$-Approximation

In Section 2.3, we describe Zelikovsky's 11/6 approximation algorithm for the minimum cost Steiner Tree. At a high level, his strategy is to maximize the difference between value of the function save_G (Definition 2.8), and the cost function cost, for a set of greedily chosen 3-restricted full components. We show that save_G is submodular. Using this and Theorem 2.6, which tells us that there is a set of k-restricted full components that will allow us to approximate T^*, we show a simpler approximation algorithm that relies on applying one of our algorithms for submodular function maximization to maximize save over the set of all k-restricted components, with a knapsack constraint based on the cost function. This results in a better approximation ratio, and a simpler proof that does not rely as much on technical details about minimum spanning trees.

Lemma 4.4 (save_G is submodular)**.** *The function* save_G *is monotone submodular, and can be calculated in time $\mathcal{O}(|E| \log |V|)$.*

Proof. For each full k-component q, consider the set of all pairwise edges among terminals of q. Now let S_q be this set of edges, each in a pair associated with the cost 0. Let $\mathcal{D}$ be the collection of S_q for every possible k-component q.

Now consider $\text{save}_G(Z)$ for an arbitrary set of k-components Z. We can define $\text{save}_G(Z) = \text{save}'_G(\mathcal{D}_Z)$, where $\mathcal{D}_Z \subseteq \mathcal{D}$ consists of S_q for $q \in Z$. Note that save' is a contraction function over graph G and base set $\mathcal{D}$.

Since save' is a contraction function, Lemma 4.2 tells us that save' must be monotone submodular, and Lemma 4.3 tells us that save' can evaluated in time $\mathcal{O}(|E| \log |V|)$. Since save and save' have a one-to-one correspondence in input-output pairs, save must also be monotone submodular and computable in $\mathcal{O}(|E| \log |V|)$ time. □

Lemma 4.5 (Tree construction from full components)**.** *Given a set Z of full components from G_{full}, we can find a Steiner tree, T, in G_{full} such that*

$$\text{cost}(T) \leq \text{mstcost}(G) - \text{save}_G(Z) + \text{cost}(Z),$$

in time $\mathcal{O}(|V| + |E| \log |V|)$.

Proof. We first show that there exists a tree T in G_{full} such that $\text{cost}(T) \leq \text{mstcost}(G) - \text{save}_G(Z) + \text{cost}(Z)$ such that T spans all terminals, and all Steiner nodes in Z.

Let $\mathcal{D}_Z$ be the collection of sets of contractions corresponding to Z as described in Lemma 4.4. We know that $\text{mstcost}(G[\mathcal{D}_Z])$ describes a the cost of a spanning tree connecting all terminals in G_{full}, given that the distance between two vertices in the same full component in Z is 0. We also know that with $\text{cost}(Z)$, we can connect all terminals to other terminals that they share full components in Z with, and connect all Steiner nodes in Z. This means that there is a tree of cost at most $\text{mstcost}(G[\mathcal{S}_Z]) + \text{cost}(Z) = \text{mstcost}(G) - \text{save}_G(Z) + \text{cost}(Z)$ in G_{full} that connects all terminals and the Steiner nodes in Z.

Let G' be a subgraph of G_{full} that includes all terminals and the Steiner nodes in Z. Let $T' = \text{MST}(G')$. The above gives us that

$$\text{cost}(T') \leq \text{mstcost}(G) - \text{save}_G(Z) + \text{cost}(Z).$$

Taking the subgraph G' takes time at most $\mathcal{O}(|V|+|E|)$, and finding the MST takes time $\mathcal{O}(|E| \log |V|)$. So in total, we can find T' in time $\mathcal{O}(|V| + |E| \log |V|)$. $\square$

Lemma 4.6 (Contracting a Steiner Tree results in MST cost 0)**.** *Let $S_{Z^*_{(k)}}$ be the set of graph contractions corresponding to $Z^*_{(k)}$ as according to Lemma 4.4.*

$$\text{mstcost}\left(G\left[S_{Z^*_{(k)}}\right]\right) = 0.$$

Proof. $Z^*_{(k)}$ is a set of full components that makes up the optimal k-restricted Steiner Tree in G_{full}. This means that all edges in $T^*_{(k)}$ are in some full component in $Z^*_{(k)}$, so in $G\left[S_{Z^*_{(k)}}\right]$, $T^*_{(k)}$ has cost 0. $\square$

Lemma 4.7 (Result of applying partial enumeration algorithm to save_G)**.** *Given α and k, graphs $G = (V, E)$, $G_{full} = (V_{full}, E_{full})$, and a value $B \geq \text{cost}\left(Z^*_{(k)}\right)$, we can find a set of full components Z from G_{full}, such that $\text{cost}(Z) \leq \alpha \cdot B$, and*

$$\text{save}_G(Z) \geq \left(1 - \frac{1}{e^\alpha}\right) \text{save}_G(Z^*_{(k)}),$$

in time $\mathcal{O}(|V|^{k\cdot(e^\alpha+3)+2} \cdot \log |V| + |V_{full}|^{2k})$.

Proof. We apply the Partial Enumeration Algorithm (Algorithm 5) to approximately maximize a monotone submodular function. We apply it to the function save_G, and set K of all k-components in G_{full}, and the cost function over $v \in V$ $\text{cost}(v)$.

By Theorem 3.8, we know that this makes at most $\mathcal{O}(|K|^{e^\alpha+3})$ calls to save_G, which can be calculated in time $\mathcal{O}(|E| \log |V|)$ (Lemma 4.4).

We also know that there are at most $|V|^k$ possible k-components in G. The optimum way to connect a single k-component takes at most k Steiner nodes, so we can find this by iterating over $|V_{\text{full}}|^k$

possibilities. This means that $|K| \leq |V|^k$, and the time to compute K and all values for the cost function is at most $|V|^k \cdot |V_{\text{full}}|^k$. This bounds our total runtime as $\mathcal{O}(|V|^{k\cdot(e^\alpha+3)} \cdot |E| \log |V| + |V|^k |V_{\text{full}}|^k) \in \mathcal{O}(|V|^{k\cdot(e^\alpha+3)+2} \cdot \log |V| + |V_{\text{full}}|^{2k})$.

Since $B \geq \operatorname{cost}\left(Z^*_{(k)}\right)$, $Z^*_{(k)}$ is a feasible solution. This means that the algorithm will give us a set $\mathcal{S}$ such that $\operatorname{save}_G(\mathcal{S}) \geq \left(1 - \frac{1}{e^\alpha}\right) \operatorname{save}_G\left(Z^*_{(k)}\right)$, and $c(\mathcal{S}) \leq \alpha \cdot B$. $\square$

Lemma 4.8 (Approximation ratio in terms of α and k)**.** *Given α and k, we can find a tree T' such that*

$$\operatorname{cost}(T') \leq \left(\alpha + \frac{2}{e^\alpha} + \alpha \cdot \varepsilon\right) \operatorname{cost}\left(Z^*_{(k)}\right)$$

in time $\mathcal{O}\left(\frac{1}{\varepsilon}\left(|V|^{k\cdot(e^\alpha+3)+2} \cdot \log |V| + |V_{\mathit{full}}|^{2k}\right)\right)$.

Proof. We know by Theorem 2.7 that MST(G) has cost at most 2 times the optimum Steiner tree. This means that

$$\frac{1}{2} \cdot \operatorname{mstcost}(G) \leq \operatorname{cost}\left(Z^*_{(k)}\right) \leq \operatorname{mstcost}(G).$$

This means that among the $\frac{1}{\varepsilon} + 1$ values

$$\frac{1}{2} \cdot \operatorname{mstcost}(G), (1+\varepsilon) \cdot \frac{1}{2} \cdot \operatorname{mstcost}(G), (1+2\varepsilon) \cdot \frac{1}{2} \cdot \operatorname{mstcost}(G), \ldots, \operatorname{mstcost}(G)$$

at least one of them must be a value B such that

$$\operatorname{cost}\left(Z^*_{(k)}\right) \leq B \leq (1+\varepsilon) \operatorname{cost}\left(Z^*_{(k)}\right).$$

Given this B, we could use Lemma 4.7 to find $Z'_{(k)}$ such that $\operatorname{cost}\left(Z'_{(k)}\right) \leq \alpha(1+\varepsilon) \operatorname{cost}\left(Z^*_{(k)}\right)$ and $\operatorname{save}_G\left(Z'_{(k)}\right) \geq \left(1 - \frac{1}{e^\alpha}\right) \operatorname{save}_G\left(Z^*_{(k)}\right)$. Let $\mathcal{S}_{Z^*_{(k)}}$ be the set of contractions corresponding to $Z^*_{(k)}$, as per Lemma 4.4. By Lemma 4.5, we could construct a tree T', such that

$$\begin{aligned}
\operatorname{cost}(T') &\leq \operatorname{mstcost}(G) - \operatorname{save}_G\left(Z'_{(k)}\right) + \operatorname{cost}\left(Z'_{(k)}\right) \\
&\leq \operatorname{mstcost}(G) - \left(1 - \frac{1}{e^\alpha}\right) \operatorname{save}_G\left(Z^*_{(k)}\right) + \alpha(1+\varepsilon) \operatorname{cost}\left(Z^*_{(k)}\right) && \text{by Lemma 4.7} \\
&\leq \frac{1}{e^\alpha} \cdot \operatorname{mstcost}(G) + \left(1 - \frac{1}{e^\alpha}\right) \operatorname{mstcost}\left(G\left[\mathcal{S}_{Z^*_{(k)}}\right]\right) + \alpha(1+\varepsilon) \operatorname{cost}\left(Z^*_{(k)}\right) && \text{by definition of } \operatorname{save}_G \\
&\leq \frac{2}{e^\alpha} \cdot \operatorname{cost}\left(Z^*_{(k)}\right) + 0 + \alpha(1+\varepsilon) \operatorname{cost}\left(Z^*_{(k)}\right) && \text{by Lemma 4.6} \\
&\leq \left(\alpha + \frac{2}{e^\alpha} + \alpha \cdot \varepsilon\right) \operatorname{cost}\left(Z^*_{(k)}\right).
\end{aligned}$$

This means that if we try all $\frac{1}{\varepsilon} + 1$ possibilities for B, and take the minimum cost resulting tree to be T', we will get a T' such that

$$\operatorname{cost}(T') \leq \left(\alpha + \frac{2}{e^\alpha} + \alpha \cdot \varepsilon\right) \operatorname{cost}\left(Z^*_{(k)}\right).$$

Applying Lemma 4.7 and Lemma 4.5 takes time $\mathcal{O}(|V|^{k\cdot(e^\alpha+3)+2} \cdot \log |V| + |V_{\text{full}}|^{2k})$. Since we do this $\frac{1}{\varepsilon}+1$ times, we can find T' in time $\mathcal{O}\left(\frac{1}{\varepsilon}\left(|V|^{k\cdot(e^\alpha+3)+2} \cdot \log |V| + |V_{\text{full}}|^{2k}\right)\right)$. □

Theorem 4.9 (Approximation for save_G based algorithm)**.** *We can find a*

$$1 + \ln 2 + \mathcal{O}(\varepsilon) \approx 1.693$$

approximation to the best Steiner Tree in G in time $\mathcal{O}\left(\frac{1}{\varepsilon}\left(|V|^{2^{\frac{1}{\varepsilon}}\cdot(e^\alpha+3)+2} \cdot \log |V| + |V_{full}|^{2\cdot 2^{\frac{1}{\varepsilon}}}\right)\right)$.

Proof. We know by Theorem 2.6 that for $k = 2^{\frac{1}{\varepsilon}}$, $\frac{\text{cost}\left(T^*_{(k)}\right)}{\text{cost}(T^*)} \leq 1+\varepsilon$.

Now, we can apply Lemma 4.8 with $k = 2^{\frac{1}{\varepsilon}}$. For a setting of α, this will give us a T' such that

$$\begin{aligned} \text{cost}(T') &\leq \left(\alpha + \frac{2}{e^\alpha} + \alpha\cdot\varepsilon\right)\text{cost}\left(Z^*_{(2^{\frac{1}{\varepsilon}})}\right) \\ &\leq \left(\alpha + \frac{2}{e^\alpha} + \alpha\cdot\varepsilon\right)(1+\varepsilon)\,\text{cost}\left(T^*\right) && \text{by Theorem 2.6} \\ &\leq \left(\alpha + \frac{2}{e^\alpha} + 3\alpha\varepsilon\right)\text{cost}\left(T^*\right). \end{aligned}$$

If we set $\alpha = \ln 2$, this gives us that

$$\text{cost}(T') \leq (\ln 2 + 1 + 3\cdot(\ln 2)\cdot\varepsilon)\,\text{cost}(T^*) = (1+\ln 2 + \mathcal{O}(\varepsilon))\,\text{cost}(T^*) \approx 1.693\cdot\text{cost}(T^*).$$

According to the time bound from Lemma 4.8, we can do this in time

$$\mathcal{O}\left(\frac{1}{\varepsilon}\left(|V|^{2^{\frac{1}{\varepsilon}}\cdot(e^\alpha+3)+2} \cdot \log |V| + |V_{\text{full}}|^{2\cdot 2^{\frac{1}{\varepsilon}}}\right)\right).$$ □

This result is hard to compare to Zelikovsky's result in [Zel93], since we optimize over the set of k-components, and Zelikovsky only considered triples. If we actually use this optimizer for triples, our algorithm does not do as well as Zelikovsky's algorithm. However, our analysis is simpler, and it paves the way for the next section, in which we match Robins and Zelikovsky's better bound.

4.3 A Simplified $(1+\frac{\ln 3}{2})$-Approximation

In Section 2.4, we describe Robins and Zelikovsky's $1+\frac{\ln 3}{2} \approx 1.55$ approximation algorithm for the minimum cost Steiner Tree [RZ05]. Their algorithm greedily chooses full components K that maximize $\frac{\text{imp}_T(K)}{\text{loss}(K)}$, where the functions imp and loss are defined in Definition 2.10 and Definition 2.9, and T is the tree that we improve on each iteration.

In this section, we present a simpler algorithm and analysis, by showing that the imp function is submodular, and applying an algorithm to approximately maximize a submodular function. We then show that we get the same approximation ratio of $1+\frac{\ln 3}{2}$, and compare the two algorithms, to show that they perform identically.

Lemma 4.10 (imp_G is submodular)**.** *The function* imp_G *is monotone submodular, and can be calculated in time* $\mathcal{O}(|E|\log|V|)$*.*

Proof. For each full k-component q, let q' be q where all edges in $\mathrm{Loss}(q)$ have been set to weight 0. Now, consider the edge between each pair of terminals u, v in q, and associate it with the shortest path distance between u and v in q'. Call this set of edge-cost pairs S_q. Let $\mathcal{D}$ be the collection of S_q for every possible k-component q.

Now consider $\mathrm{imp}_G(Z)$ for an arbitrary set of k-components Z. We can define $\mathrm{imp}_G(Z) = \mathrm{imp}'_G(\mathcal{D}_Z)$, where $\mathcal{D}_Z \subseteq \mathcal{D}$ consists of S_q for $q \in Z$. Note that imp' is a contraction function over graph G and base set $\mathcal{D}$.

Since imp' is a contraction function, Lemma 4.2 tells us that imp' must be monotone submodular, and Lemma 4.3 tells us that imp' can evaluated in time $\mathcal{O}(|E|\log|V|)$. Since imp and imp' have a one-to-one correspondence in input-output pairs, imp must also be monotone submodular and computable in $\mathcal{O}(|E|\log|V|)$ time. □

Lemma 4.11 (Tree construction from full components)**.** *Given a set Z of full components from G_{full}, we can find a Steiner tree T in G_{full} such that*

$$\mathrm{cost}(T) \leq \mathrm{mstcost}(G) - \mathrm{imp}_G(Z) + \mathrm{loss}(Z)$$

in time $\mathcal{O}(|E_{full}|\log|V_{full}| + |V_{full}|)$*.*

Proof. We convert each component in $z \in Z$ into a set of edge contractions, S_z, as described in Lemma 4.10. Let $\mathcal{D}_Z = \{S_z : z \in Z\}$. $G[\mathcal{D}_Z]$ is the graph G, where the cost of connecting $\mathrm{Loss}(z)$ for every $z \in Z$ is set to 0. This means that if we include all of the nodes in $\mathrm{Loss}(z)$ for each $z \in Z$, there exists a spanning tree of cost at most $\mathrm{mstcost}(G[\mathcal{D}_Z]) + \mathrm{loss}(Z)$ in G_{full}, that spans all of the terminals and all Steiner nodes in Z.

This means if we take the subgraph of G_{full} consisting of all terminals and the Steiner nodes in Z, and take the minimum spanning tree of the resulting graph, we can get a tree T, such that

$$\begin{aligned}\mathrm{cost}(T) &\leq \mathrm{mstcost}(G[\mathcal{D}_Z]) + \mathrm{loss}(Z)\\ &\leq \mathrm{mstcost}(G) - \mathrm{imp}_G(Z) + \mathrm{loss}(Z).\end{aligned}$$

Taking the subgraph of G_{full} takes time at most $|V_{\text{full}}| + |E_{\text{full}}|$, and taking the MST over the resulting graph takes time $\mathcal{O}(|E_{\text{full}}|\log|V_{\text{full}}|)$, for a total of $\mathcal{O}(|E_{\text{full}}|\log|V_{\text{full}}| + |V_{\text{full}}|)$ time. □

Lemma 4.12 (Result of applying partial enumeration algorithm to imp_G)**.** *Given α and k, graphs $G = (V, E)$, $G_{full} = (V_{full}, E_{full})$, and a value $B \geq \mathrm{loss}\left(Z^*_{(k)}\right)$, we can find a set of full components Z from G_{full}, such that* $\mathrm{loss}(Z) \leq \alpha \cdot B$, *and*

$$\mathrm{imp}_G(Z) \geq \left(1 - \frac{1}{e^\alpha}\right)\mathrm{imp}_G(Z^*_{(k)}),$$

in time $\mathcal{O}(|V|^{k\cdot(e^\alpha+3)+2} \cdot \log|V| + |V_{full}|^{2k})$*.*

Proof. The proof of this lemma is identical to Lemma 4.7. We include it here for completeness.

We apply the Partial Enumeration Algorithm (Algorithm 5) to approximately maximize a monotone submodular function. We apply it to the function imp_G, and set K of all k-components in G_{full}, and the cost function over $v \in V$ $\text{loss}(v)$.

By Theorem 3.8, we know that this makes at most $\mathcal{O}(|K|^{e^\alpha+3})$ calls to save_G, which can be calculated in time $\mathcal{O}(|E|\log|V|)$ (Lemma 4.10).

We also know that there are at most $|V|^k$ possible k-components in G. The optimum way to connect a single k-component takes at most k Steiner nodes, so we can find this by iterating over $|V_{\text{full}}|^k$ possibilities. This means that $|K| \leq |V|^k$, and the time to compute K and all values for the cost function is at most $|V|^k \cdot |V_{\text{full}}|^k$. This bounds our total runtime as $\mathcal{O}(|V|^{k\cdot(e^\alpha+3)} \cdot |E|\log|V| + |V|^k|V_{\text{full}}|^k) \in \mathcal{O}(|V|^{k\cdot(e^\alpha+3)+2} \cdot \log|V| + |V_{\text{full}}|^{2k})$.

Since $B \geq \text{cost}\left(Z^*_{(k)}\right)$, $Z^*_{(k)}$ is a feasible solution. This means that the algorithm will give us a set $\mathcal{S}$ such that $\text{imp}_G(\mathcal{S}) \geq \left(1 - \frac{1}{e^\alpha}\right)\text{imp}_G\left(Z^*_{(k)}\right)$, and $c(\mathcal{S}) \leq \alpha \cdot B$. □

Lemma 4.13 (Approximation ratio in terms of α and k)**.** *Given $\alpha > 1$ and k, we can find a tree T' such that*

$$\text{cost}(T') \leq \frac{1}{2}\left(1 + \frac{3}{e^\alpha} + \alpha\cdot\varepsilon + \alpha\right)\text{cost}\left(T^*_{(k)}\right)$$

in time $\mathcal{O}\left(\frac{1}{\varepsilon}\left(|V|^{k\cdot(e^\alpha+3)+2} \cdot \log|V| + |V_{full}|^{2k}\right)\right)$.

Proof. We know from Lemma 2.11 that $\text{loss}\left(Z^*_{(k)}\right) \leq \text{cost}\left(T^*_{(k)}\right)$, and from Theorem 2.7 that $2 \cdot \text{cost}\left(T^*_{(k)}\right) \leq \text{mstcost}(G)$. This gives us that

$$0 \leq \text{loss}\left(Z^*_{(k)}\right) \leq \frac{1}{2}\text{mstcost}(G).$$

This means that among the $\frac{1}{\varepsilon}+1$ values

$$0,\ \frac{\varepsilon}{2}\cdot\text{mstcost}(G),\ \frac{2\varepsilon}{2}\cdot\text{mstcost}(G),\ \ldots,\ \frac{1}{2}\cdot\text{mstcost}(G)$$

at least one of them must be a value B such that

$$\text{loss}\left(Z^*_{(k)}\right) \leq B \leq \text{loss}\left(Z^*_{(k)}\right) + \frac{\varepsilon}{2}\cdot\text{mstcost}(G).$$

Given this B, we can use Lemma 4.12 to find $Z'_{(k)}$ such that $\text{loss}\left(Z'_{(k)}\right) \leq \alpha\cdot\text{loss}\left(Z^*_{(k)}\right) + \alpha\cdot\frac{\varepsilon}{2}\cdot\text{mstcost}(G)$ and $\text{imp}_G\left(Z'_{(k)}\right) \geq \left(1-\frac{1}{e^\alpha}\right)\text{save}_G\left(Z^*_{(k)}\right)$. Then by Lemma 4.11, we would be able to construct a tree T' such that

$$\begin{aligned}\text{cost}(T') &\leq \text{mstcost}(G) - \text{imp}_G\left(Z'_{(k)}\right) + \text{loss}\left(Z'_{(k)}\right)\\ &\leq \text{mstcost}(G) - \left(1-\frac{1}{e^\alpha}\right)\text{imp}_G\left(Z^*_{(k)}\right) + \alpha\cdot\text{loss}\left(Z^*_{(k)}\right) + \alpha\cdot\frac{\varepsilon}{2}\cdot\text{mstcost}(G)\end{aligned}$$

$$\leq \left(\frac{1}{e^\alpha} + \frac{\alpha \cdot \varepsilon}{2}\right) \text{mstcost}(G) + \left(1 - \frac{1}{e^\alpha}\right)\left(\text{mstcost}(G) - \text{imp}_G\left(Z^*_{(k)}\right)\right) + \alpha \cdot \text{loss}\left(Z^*_{(k)}\right). \tag{14}$$

Now, note that $T^*_{(k)}$ consists of Loss(q) for every component $q \in Z^*_{(k)}$, and the MST after all of the Losses in $Z^*_{(k)}$ have been contracted. This is the value $\text{mstcost}(G) - \text{imp}_G\left(Z^*_{(k)}\right)$. So this gives us that

$$\text{cost}\left(T^*_{(k)}\right) = \text{loss}\left(Z^*_{(k)}\right) + \text{mstcost}(G) - \text{imp}_G\left(Z^*_{(k)}\right).$$

Now, we know by Lemma 2.11 that $\text{loss}\left(Z^*_{(k)}\right) \leq \frac{\text{cost}\left(T^*_{(k)}\right)}{2}$. We also know from Theorem 2.7 that $\text{mstcost}(G) \leq 2 \cdot \text{cost}\left(T^*_{(k)}\right)$. Since $\alpha \geq 1$, along with Eq. (14), we get that

$$\begin{aligned}\text{cost}(T') &\leq \left(\frac{2}{e^\alpha} + \alpha \cdot \varepsilon\right) \text{cost}\left(T^*_{(k)}\right) + \left(1 - \frac{1}{e^\alpha}\right)\frac{\text{cost}\left(T^*_{(k)}\right)}{2} + \alpha \frac{\text{cost}\left(T^*_{(k)}\right)}{2} \\ &\leq \frac{1}{2}\left(1 + \frac{3}{e^\alpha} + \alpha \cdot \varepsilon + \alpha\right) \text{cost}\left(T^*_{(k)}\right).\end{aligned}$$

Applying Lemma 4.12 and Lemma 4.11 takes time $\mathcal{O}(|V|^{k\cdot(e^\alpha+3)+2} \cdot \log|V| + |V_{\text{full}}|^{2k})$. Since we do this $\frac{1}{\varepsilon} + 1$ times, we can find T' in time $\mathcal{O}\left(\frac{1}{\varepsilon}\left(|V|^{k\cdot(e^\alpha+3)+2} \cdot \log|V| + |V_{\text{full}}|^{2k}\right)\right)$. □

Theorem 4.14 (Approximation ratio for imp_G based algorithm)**.** *We can find a*

$$1 + \frac{\ln 3}{2} + \mathcal{O}(\varepsilon) \approx 1.549$$

approximation to the best Steiner Tree in G_{full} in time $\mathcal{O}\left(\frac{1}{\varepsilon}\left(|V|^{2^{\frac{1}{\varepsilon}}\cdot(e^\alpha+3)+2} \cdot \log|V| + |V_{full}|^{2\cdot 2^{\frac{1}{\varepsilon}}}\right)\right)$.

Proof. Theorem 2.6 gives us that $2^{\frac{1}{\varepsilon}}$, $\text{cost}\left(T^*_{(2^{\frac{1}{\varepsilon}})}\right) / \text{cost}\left(T^*\right) \leq 1+\varepsilon$. This means that if we set $k = 2^{\frac{1}{\varepsilon}}$, Lemma 4.13 gives us

$$\text{cost}(T') \leq \frac{1}{2}\left(1 + \frac{3}{e^\alpha} + \alpha \cdot \varepsilon + \alpha\right)(1+\varepsilon)\,\text{cost}\left(T^*\right).$$

Setting $\alpha = \ln(3)$ gives us that

$$\begin{aligned}\text{cost}(T') &\leq \left(1 + \frac{\ln 3}{2} + \frac{\varepsilon \cdot \ln 3}{2}\right)(1+\varepsilon)\,\text{cost}(T^*) \\ &\leq \left(1 + \frac{\ln 3}{2} + 4\varepsilon\right)\text{cost}(T^*) \\ &\approx 1.549 \text{ cost}(T^*).\end{aligned}$$

The lemma also tells us that we can do this in time

$$\mathcal{O}\left(\frac{1}{\varepsilon}\left(|V|^{2^{\frac{1}{\varepsilon}}\cdot(e^\alpha+3)+2} \cdot \log|V| + |V_{\text{full}}|^{2\cdot 2^{\frac{1}{\varepsilon}}}\right)\right).$$ □

4.3.1 Relationship to Robins and Zelikovsky's Algorithm

Theorem 4.14 shows a simpler approach to acheive the same approximation ratio as described in [RZ05] (Section 2.4). The approach of Robins and Zelikovsky in [RZ05] may at first seem quite different from ours, as their analysis relies more on technical arguments about the structures of minimum spanning trees, and the imp function. However, the two algorithms and analyses are actually almost identical.

In Robins and Zelikovsky's algorithm (Algorithm 2), at each step they greedily choose the component K that maximizes $\operatorname{imp}_T(K)/\operatorname{loss}(K)$ where T is the MST on the successively contracted graph, as long as $\operatorname{imp}_T(K) - \operatorname{loss}(K) \geq 0$. When we apply the Partial Enumeration Algorithm (Algorithm 5) with objective function imp_G and cost function $\operatorname{loss}(K)$, either we brute force find the correct solution, or we also greedily choose the component K that maximizes $\operatorname{imp}_T(K)/\operatorname{loss}(K)$ at each step, until the total loss exceeds $\ln(3) \cdot \operatorname{loss}\left(T^*_{(k)}\right)$. The only difference between the greedy parts of the two algorithms is the condition to end the loop.

But a key part of Robins and Zelikovsky's analysis is where they show that the loss of their greedily chosen components is upper bounded by $\operatorname{loss}\left(T^*_{(k)}\right) \cdot \ln\left(1 + \frac{\operatorname{mstcost}(G) - \operatorname{cost}\left(T^*_{(k)}\right)}{\operatorname{loss}\left(T^*_{(k)}\right)}\right)$. This bound can be simplified to

$$\begin{aligned}
\operatorname{loss}\left(T^*_{(k)}\right) \cdot \ln\left(1 + \frac{\operatorname{mstcost}(G) - \operatorname{cost}\left(T^*_{(k)}\right)}{\operatorname{loss}\left(T^*_{(k)}\right)}\right) &\leq \operatorname{loss}\left(T^*_{(k)}\right) \cdot \ln\left(1 + \frac{\operatorname{cost}\left(T^*_{(k)}\right)}{\operatorname{loss}\left(T^*_{(k)}\right)}\right) && \text{Theorem 2.7} \\
&\leq \operatorname{loss}\left(T^*_{(k)}\right) \cdot \ln\left(1 + 2\right) && \text{Lemma 2.11} \\
&= \ln(3) \cdot \operatorname{loss}\left(T^*_{(k)}\right).
\end{aligned}$$

This is essentially the same simplifications that they make in their analysis. This means that their analysis assumes the same stopping condition as ours, so the two algorithms are effectively identical.

5 Future Work

In this thesis, we discussed simplified algorithms based on Zelikovsky's 1.83-approximation [Zel93], and Robins and Zelikovsky's 1.55-approximation [RZ05] for the Network Steiner Problem. Both of these results used combinatorial approaches, which made a series of greedy improvements to the minimum spanning tree. We were able to view these greedy improvements as submodular functions. This allowed us to use approximation algorithms for submodular function maximization to get the same approximation results.

One direction we could continue in is to see if we can use submodular maximization to simplify Steiner Tree approximations found by other techniques. Specifically, the current state of the art Steiner Tree approximation algorithm is due to Byrka, Grandoni, Rothvoß, and Sanità, who achieve a 1.39-approximation [BGRS10]. Their approach is completely different than the combinatorial algorithms we discussed earlier. They start by modeling the problem using a LP (linear programming) relaxation. In their LP, they assign a variable to each component. They solve the LP, and get an assignment to the variables associated with each component. They then sample a component based on the relative weights assigned to each component by the LP, and contract the sampled component. They then iterate the process until they have a complete spanning tree.

This is a randomized algorithm, but they show that using limited independence, they can run their algorithm using $\mathcal{O}(\log n)$ random bits. This means that they can derandomize their algorithm by running the original algorithm for each possible setting of the $\mathcal{O}(\log n)$ random bits, which is still in polynomial-time. One possible future research direction would be to see if and how this approach can be adapted using techniques from submodular maximization.

References

[BD97] Al Borchers and Ding-Zhu Du. The k-steiner ratio in graphs. *SIAM J. Comput.*, 26(3):857–869, June 1997.

[BGRS10] Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. An improved lp-based approximation for steiner tree. In *Proceedings of the Forty-Second ACM Symposium on Theory of Computing*, STOC '10, page 583–592, New York, NY, USA, 2010. Association for Computing Machinery.

[Cha18] Deeparnab Chakrabarty. Personal conversation, August 2018.

[Fei98] Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, July 1998.

[Hor] Thibaut Horel. Notes on greedy algorithms for submodular maximization. Available at `https://thibaut.horel.org/submodularity/notes/02-12.pdf` (2015/02/26).

[Kar72] R.M. Karp. Reducibility among combinatorial problems. In R.E. Miller, J.W. Thatcher, and Bohlinger J.D., editors, *Complexity of Computer Computations*. Springer, Boston MA, 1972.

[NWF78] George Nemhauser, Laurence Wolsey, and M. Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical Programming*, 14:265–294, 12 1978.

[RZ05] Gabriel Robins and Alexander Zelikovsky. Tighter bounds for graph steiner tree approximation. *SIAM J. Discrete Math.*, 19(1):122–134, 2005.

[Svi04] Maxim Sviridenko. A note on maximizing a submodular set function subject to a knapsack constraint. *Operations Research Letters*, 32(1):41–43, 2004.

[Zel93] A. Zelikovsky. An 11/6-approximation algorithm for the network steiner problem. *Algorithmica*, 9:463–470, 05 1993.