Quasi-Polynomial Algorithms for Submodular Tree Orienteering and Other Directed Network Design Problems

Rohan Ghuge* Viswanath Nagarajan*

Abstract

We consider the following general network design problem on directed graphs. The input is an asymmetric metric (V, c), root $r^* \in V$, monotone submodular function $f : 2^V \to \mathbb{R}_+$ and budget B. The goal is to find an r^*-rooted arborescence T of cost at most B that maximizes $f(T)$. Our main result is a very simple quasi-polynomial time $O(\frac{\log k}{\log\log k})$-approximation algorithm for this problem, where $k \le |V|$ is the number of vertices in an optimal solution. To the best of our knowledge, this is the first non-trivial approximation ratio for this problem. As a consequence we obtain an $O(\frac{\log^2 k}{\log\log k})$-approximation algorithm for directed (polymatroid) Steiner tree in quasi-polynomial time. We also extend our main result to a setting with additional length bounds at vertices, which leads to improved $O(\frac{\log^2 k}{\log\log k})$-approximation algorithms for the single-source buy-at-bulk and priority Steiner tree problems. For the usual directed Steiner tree problem, our result matches the best previous approximation ratio [15], but improves significantly on the running time: our algorithm takes $n^{O(\log^{1+\epsilon} k)}$ time whereas the previous algorithm required $n^{O(\log^5 k)}$ time. For polymatroid Steiner tree and single-source buy-at-bulk, our result improves prior approximation ratios by a logarithmic factor. For directed priority Steiner tree, our result seems to be the first non-trivial approximation ratio. Under certain complexity assumptions, our approximation ratios are best possible (up to constant factors).

1 Introduction

Network design problems, involving variants of the *minimum spanning tree* (MST) and *traveling salesman problem* (TSP), are extensively studied in approximation algorithms. These problems are also practically important as they appear in many applications, e.g. networking and vehicle routing. Designing algorithms for problems on directed networks is usually much harder than their undirected counterparts. This difference is already evident in the most basic MST problem: the undirected case admits a simple greedy algorithm whereas the directed case requires a much more complex algorithm [11]. Indeed, one of the major open questions in network design concerns the directed Steiner tree problem. Given a directed graph with edge costs and a set of terminal vertices, the goal here is a minimum cost arborescence that contains all terminals. No polynomial-time polylogarithmic approximation is known for directed Steiner tree. This is in sharp contrast with undirected Steiner tree, for which a 2-approximation is folklore and there are even better constant approximation ratios [2, 22].

In this paper, we consider a variant of directed Steiner tree, where the goal is to find an arborescence maximizing the number (or profit) of vertices subject to a hard constraint on its cost. We call this problem *directed tree orienteering* (DTO). To the best of our knowledge, this problem has not been studied explicitly before. An α-aproximation algorithm for DTO implies an $(\alpha \cdot \ln k)$-approximation algorithm for directed Steiner tree, using a set-covering approach. No approximation preserving reduction is known in the reverse direction: so approximation algorithms for directed Steiner tree do not imply anything for DTO. In this paper, we obtain a *quasi-polynomial* time $O(\frac{\log k}{\log\log k})$-approximation algorithm for DTO, where k is the number of vertices in an optimal solution. This also implies an $O(\frac{\log^2 k}{\log\log k})$-approximation algorithm for directed Steiner tree (in quasi-polynomial time) where k denotes the number of terminals.

In contrast to DTO, the "path" or "tour" version of directed orienteering, where one wants a path/tree of maximum profit subject to the cost limit, is much better understood. There are polynomial time approximation algorithms with guarantees $O(\log n)$ [20, 25] and $O(\log^2 k)$ [8]. However, these results do not imply anything for DTO. Unlike undirected graphs, in the directed case, we cannot go between trees and tours by doubling edges.

Our algorithm for DTO in fact follows as a special case of a more general algorithm for the *submodular tree orienteering* (STO) problem. Here, we are also given a monotone submodular function $f : 2^V \to \mathbb{R}_+$ on the vertex set, and the goal is to find an arborescence containing vertices $T \subseteq V$ that maximizes $f(T)$ subject to a cost

*Industrial and Operations Engineering Department, University of Michigan. Supported in part by NSF CAREER grant CCF-1750127. Email: rghuge@umich.edu, viswa@umich.edu.

limit B. The "tour" or "path" version of submodular orienteering was studied previously in [9], where a quasi-polynomial time $O(\frac{\log k}{\log\log k})$-approximation algorithm was obtained. While we rely on many ideas from [9], we also need a number of new ideas- as discussed next.

Interestingly, our techniques can be easily extended to obtain tight quasi-polynomial time approximation algorithms for several other directed network design problems such as polymatroid Steiner tree, single-source buy-at-bulk and priority Steiner tree.

1.1 Results and Techniques

Our main result is:

Theorem 1.1 *There is an $O(\frac{\log k}{\log\log k})$-approximation algorithm for submodular tree orienteering that runs in $(n\log B)^{O(\log^{1+\epsilon} k)}$ time for any constant $\epsilon > 0$.*

The high-level approach here is the elegant "recursive greedy" algorithm from [9] for the submodular *path* orienteering problem, which in turn is similar to the recursion used in Savitch's theorem [24]. In order to find an approximately optimal $s-t$ path with budget B, the algorithm in [9] *guesses* the "middle node" v on the optimal $s-t$ path as well as the cost B' of the optimal path segment from s to v. Then, it solves two smaller instances recursively and sequentially:

1. find an approximately optimal $s-v$ path P_{left} with budget B'.

2. find an approximately optimal $v-t$ path P_{right} with budget $B-B'$ that *augments* P_{left}.

Clearly, the depth of recursion is $\log_2 k$ where k denotes the number of nodes in an optimal path. The key step in the analysis is to show that the approximation ratio is equal to the depth of recursion. In the *tree* version that we consider there are two additional issues:

- Firstly, there is no middle node v in an arborescence. A natural choice is to consider a balanced separator node as v: it is well known that any tree has a $\frac{1}{3}-\frac{2}{3}$ balanced separator. Indeed, this is what we use. Although, this leads to an imbalanced recursion (not exactly half the nodes in each subproblem), the *maximum* recursion depth is still $O(\log k)$ and we show that the approximation ratio can be bounded by this quantity.

- Secondly (and more importantly), we cannot simply concatenate the solutions to the two subproblems. If r is the root of the original instance, the two subproblems involve arborescences rooted at r and v respectively. In order to finally obtain an r-arborescence, we need to additionally ensure that the subproblem with root r returns an arborescence containing the separator node v, and such requirements can accumulate recursively! Fortunately, there is a clean solution to this issue. We generalize the recursion by also specifying a "responsibility" subset $Y \subseteq V$ for each subproblem, which means that the resulting arborescence *must* contain all nodes in Y. Crucially, we can show that the size of any responsibility subset is bounded by the recursion depth $d = O(\log k)$. This allows us to implement the recursive step by additionally *guessing* how the responsibility subset Y is passed on to the two subproblems. The number of such guesses is at most $2^d = poly(k)$, and so the overall time remains quasi-polynomial. The responsibility subset Y is empty at the highest level of recursion and has size at most one at the lowest level of recursion: $|Y|$ may increase and decrease in the intermediate levels.

A direct consequence of Theorem 1.1 is an $O(\frac{\log k}{\log\log k})$-approximation algorithm for DTO and an $O(\frac{\log^2 k}{\log\log k})$-approximation for directed Steiner tree in quasi-polynomial time. This matches the previous best bound (in quasi-polynomial time) for directed Steiner tree [15]. However, our approach is much simpler and also achieves a better exponent in the running time: our time is $n^{O(\log^{1+\epsilon} k)}$ whereas the previous algorithm required $n^{O(\log^5 k)}$ time. [15] also showed that one cannot obtain an $o(\log^2 k/\log\log k)$-approximation ratio for directed Steiner tree in quasi-polynomial time assuming the Projection Game Conjecture and $NP \not\subseteq \cap_{0<\epsilon<1} ZPTIME(2^{n^\epsilon})$. Hence Theorem 1.1 is also tight under the same assumptions.

Another application of Theorem 1.1 is to the directed polymatroid Steiner tree problem, where there is a matroid with groundset V (same as the vertices) and one needs to find a minimum cost arborescence that visits some base of the matroid. We obtain an $O(\log^2 k/\log\log k)$-approximation, which improves over the previous best $O(\log^3 k)$ bound [3].

We also extend our main result (Theorem 1.1) to a setting with additional length constraints. In addition to the input to STO, here we are given a length function $\ell : E \to \mathbb{Z}_+$ and a bound L. The goal here is to find an arborescence on vertices T maximizing $f(T)$ where (i) the cost of edges in T is at most B and (ii) the sum $\sum_{v\in T} \ell_T(v) \le L$ where $\ell_T(v)$ is the length of the $r-v$ path in T. We assume that the lengths are polynomially bounded. Our technique can be extended to:

Theorem 1.2 *There is an $O(\frac{\log k}{\log\log k})$-approximation algorithm that runs in quasi-polynomial time for the submodular tree orienteering problem with length constraints.*

This algorithm follows a similar recursive structure as for STO, where we guess and maintain some additional quantities: the length budget L' available to the subproblem and a bound $D(v)$ on the length of the $r - v$ path for each vertex v in the responsibility subset Y. This idea can also be used to obtain an $O(\frac{\log k}{\log\log k})$-approximation for the variant of STO with hard deadlines on length (see Section 4.2 for details).

As a direct application of Theorem 1.2, we obtain an $O(\frac{\log^2 k}{\log\log k})$-approximation for single-source buy-at-bulk network design. This improves over the previous best $O(\log^3 k)$-approximation [1]. Buy-at-bulk network design is a well-studied generalization of Steiner tree that involves concave cost-functions on edges. See Section 4.3 for more details. Our result holds for the harder "non uniform" version of the problem, where cost-functions may differ across edges.

Another application of Theorem 1.2 is to the priority Steiner tree problem, where edges/terminals have priorities (that represent quality-of-service) and the path for each terminal must contain edges of at least its priority. We obtain a quasi-polynomial time $O(\frac{\log^2 k}{\log\log k})$-approximation even for this problem. We are not aware of any previous result for directed priority Steiner tree.

It follows from the hardness result in [15] that all our approximation ratios are tight (up to constant factors) assuming the Projection Game Conjecture and $NP \not\subseteq \cap_{0<\epsilon<1} ZPTIME(2^{n^\epsilon})$.

1.2 Related Work The first approximation algorithm for directed Steiner tree was obtained by [26] which gave an $O\left(k^\epsilon (\log k)^{1/\epsilon}\right)$ approximation in $n^{O(1/\epsilon)}$ time, for any $\epsilon > 0$. This result was improved by [4], where an $O(\log^3 k)$-approximation ratio was obtained in quasi-polynomial time: this was the first poly-logarithmic approximation ratio. This was a recursive algorithm with a very different structure than ours: the idea here was to solve (approximately) the "density" problem that finds a partial Steiner tree minimizing the ratio of the cost to the number of terminals. In contrast, we obtain a recursive approximation algorithm for the "orienteering" problem that finds a bounded-cost Steiner tree maximizing the number of terminals. Using a set-covering approach along with an algorithm for either the density or orienteering problem, it is straightforward obtain an algorithm for directed Steiner tree (at the loss of an additional $\ln k$ approximation factor). We note that the recursion used in [4] relies on an explicit bound on the tree depth, which results in the loss of an additional log-factor (which we save). Moreover, the approach of [4] is not applicable to the orienteering problem (DTO), whereas any approximation algorithm for DTO immediately implies one for the density problem.

The natural cut-covering LP relaxation of directed Steiner tree was shown to have an $\Omega(\sqrt{k})$ integrality gap by [27]. Later, [12] showed that one can also obtain an $O(\log^3 k)$-approximation ratio relative to the $O(\log k)$-level Sherali-Adams lifting of the natural LP. (Previously, [23] used the stronger Lasserre hierarchy to obtain the same approximation ratio.) Very recently, [15] improved the approximation ratio to $O(\log^2 k/\log\log k)$, still in quasi-polynomial time. Their approach was to reduce directed Steiner tree to a new problem, called "label consistent subtree" for which they provided an $O(\log^2 k/\log\log k)$-approximation algorithm (in quasi-polynomial time) by rounding a Sherali-Adams LP. In contrast, we take a simpler and more direct approach by extending the recursive-greedy algorithm of [9]. Our algorithm is easier to implement and has a much better running time. The approach in [12, 15, 23] is also not applicable to the (harder) tree orienteering problem that we solve.

A well-known special case of directed Steiner tree is the group Steiner tree problem [14], for which the best polynomial-time approximation ratio is $O(\log^2 k \log n)$. This is relative to the natural LP relaxation. A combinatorial algorithm with slightly worse approximation ratio was given by [6]. In quasi-polynomial time, there is an $O(\log^2 k/\log\log k)$-approximation algorithm, which follows from [9]. There is also an $\Omega(\log^{2-\epsilon} k)$-hardness of approximation for group Steiner tree [17]. Recently, [15] showed that this reduction can be refined to prove an $\Omega(\log^2 k/\log\log k)$-hardness of approximation (under stronger assumptions).

[3] considered a polymatroid generalization of both undirected and directed Steiner tree. For the directed version, they obtained an $O(\log^3 k)$-approximation in quasi-polynomial time by extending the approach of [4]. We improve this ratio to $O(\log^2 k/\log\log k)$, which is also the best possible. It is unclear if one can use LP-based methods such as [12,15,23] to address this problem.

Buy-at-bulk network design problems, that involve concave cost-functions, have been studied extensively as they model economies of scale (which is common in several applications). In the undirected case, a constant-factor approximation algorithm is known for *uniform* single-source buy-at-bulk [16] and an $O(\log k)$-approximation algorithm is known for the *non-uniform* version [19]. The non-uniform problem is also hard to approximate better than $O(\log\log n)$ [10]. For the directed case that we consider, the only prior result is [1] which implies a quasi-polynomial time $O(\log^3 k)$-approximation for the non-uniform version. Buy-at-bulk problems have also been studied for multi-commodity flows [7], which we do not consider in this paper.

The priority Steiner problem was introduced to model quality-of-service requirements in networking [5]. It is fairly well-understood in the undirected setting: the best approximation ratio known is $O(\log n)$ [5] and it is $\Omega(\log\log n)$ hard-to-approximate [10].

The DTO problem in undirected graphs has also received significant attention [13, 18, 21], in particular a 2-approximation algorithm is known for it [21].

1.3 Preliminaries

The input to the submodular tree orienteering (STO) problem consists of (i) a directed graph $G = (V, E)$ with edge costs $c : E \to \mathbb{Z}_+$, (ii) root vertex $r^* \in V$, (iii) a budget $B \geq 0$ and (iv) a monotone submodular function $f : 2^V \to \mathbb{R}_+$ on the power set of the vertices. As usual, we may assume (without loss of generality) that the underlying graph is complete and the costs c satisfy triangle inequality. We assume throughout that all edge costs are integer valued. We also use the standard value-oracle model for submodular functions, which means that our algorithm can access the value $f(S)$ for any $S \subseteq V$ in constant time. Finally, we assume (for simplicity) that for all $S \subseteq V$, $f(S)$ is polynomially bounded in $n = |V|$. Using standard scaling arguments, we can handle arbitrary submodular functions at the loss of an additional constant-factor in the approximation.

The goal in STO is to find an out-directed arborescence T^* that is rooted at r^* and maximizes $f(V(T^*))$ such that the cost of edges in T^* is less than B, i.e. $\sum_{e \in E(T^*)} c(e) \leq B$. Henceforth, we will use $f(V(T))$ and $f(T)$ interchangeably to mean f evaluated at the vertex set of T.

2 Algorithms for Submodular Tree Orienteering

We first descibe the basic algorithm that leads to an $(nB)^{O(\log k)}$ time $O(\log k)$-approximation algorithm for STO in Section 2.1. This already contains the main ideas. Then, in Section 2.2 we show how to make the algorithm truly quasi-polynomial time by implementing it in $(n \log B)^{O(\log k)}$ time. Finally, in Section 2.3 we show how to obtain a slightly better $O(\frac{\log k}{\log\log k})$ approximation ratio in $(n \log B)^{O(\log^{1+\epsilon} k)}$ time.

2.1 The Main Algorithm

The procedure $\mathrm{RG}(r, Y, B, X, i)$ implements the algorithm.

- The parameters $r \in V$ and $B \geq 0$ denote that we are searching for an r-rooted arborescence with cost at most B.
- $Y \subseteq V$ is a set of vertices that *must* be visited from r. We refer to set Y as the responsibilities for this subproblem.
- The parameter $X \subseteq V$ indicates that we aim to maximize the function $f_X(T) = f(T \cup X) - f(X)$; that is we seek to find an arborescence that *augments* a given set X.
- The parameter $i \geq 1$ indicates that the arborescence returned can contain at most $(\frac{3}{2})^i$ vertices excluding the root; that is, it controls the depth of the recursion.

Algorithm 1 $\mathrm{RG}(r, Y, B, X, i)$

if $(|Y| > (\frac{3}{2})^i)$ **then return Infeasible**
if $i = 1$ **then**
  if $(|Y| = 0)$ **then** ▷ No responsibility for r
    pick $v \in V : c(r, v) \leq B$ that maximizes $f_X(v)$, and return $\{(r, v)\}$. ▷ Guess base-case vertex
  if $(|Y| = 1)$ **then** ▷ r must visit vertex $v \in Y$
    if $(c(r, v) \leq B)$ **then return** $\{(r, v)\}$
    else return Infeasible
$T \leftarrow \emptyset$
$m \leftarrow f_X(\emptyset)$
for each $v \in V$ **do** ▷ Guess separator vertex
  for $S \subseteq Y$ **do** ▷ Guess responsibilities for left/right subtrees
    for $1 \leq B_1 \leq B$ **do** ▷ Guess subtree budget
      $T_1 \leftarrow \mathrm{RG}(r, (S \cup \{v\}) \setminus \{r\}, B_1, X, i-1)$
      $T_2 \leftarrow \mathrm{RG}(v, Y \setminus (S \cup \{v\}), B - B_1, X \cup V(T_1), i-1)$
      if $(f_X(T_1 \cup T_2) > m)$ **then**
        $T \leftarrow T_1 \cup T_2$
        $m \leftarrow f_X(T)$
return T

Remark 2.1 *Given a valid input to the STO problem, our solution is $T \leftarrow RG(r^*, \emptyset, B, \emptyset, d)$ for $d \geq \log_{3/2} k$ where k is the number of vertices in an optimal solution.*

Fact 2.1 *Any tree on n vertices has a vertex v whose removal leads to each connected components having size at most $n/2$. These components can be clubbed together to form two connected components (both containing v), each of size at most $2n/3$.*

Proposition 2.1 *The maximum size of set Y in any subproblem of $RG(r, \emptyset, B, \emptyset, d)$ is d.*

Proof. To prove the above statement, we argue that the invariant $|Y| + i \leq d$ holds in every subproblem of $\mathrm{RG}(r, \emptyset, B, \emptyset, d)$ of the form $\mathrm{RG}(r, Y, B, X, i)$. We prove this by induction on i. For the base case, let $i = d$. In this case, $|Y| = 0$, and thus the aforementioned invariant clearly holds. Inductively, assume that the invariant holds at some depth $i > 1$ for some responsibility set Y. Let $\text{RG}(r', Y', B', X', i-1)$ be a subproblem of $\text{RG}(r, Y, B, X, i)$. From the description of the algorithm, we can see that the size of Y increases by at most 1 in any subproblem: so $|Y'| \le 1 + |Y|$. Combining this observation with the induction hypothesis $|Y| + i \le d$, we get $|Y'| + i - 1 \le |Y| + 1 + i - 1 \le d$ which completes the induction.

Finally, as $i \ge 1$, we have $|Y| < d$ in any subproblem of $\text{RG}(r, \emptyset, B, \emptyset, d)$. ■

Proposition 2.2 *The running time of the procedure $RG(r, Y, B, X, i)$ is $O((nB \cdot 2^{d+2})^i)$.*

Proof. We prove the above claim by induction on i. Let us denote the running time of $\text{RG}(r, Y, B, X, i)$ by $T(i)$. We want to show that $T(i) \le c \cdot (nB \cdot 2^{d+2})^i$ for some fixed constant c. For the base case, let $i = 1$. From the description of the procedure, we can see that when $i = 1$, it only performs a linear number of operations. Thus $T(1) = O(n)$ which proves the base case. Inductively, assume that the claim holds for all values $i' < i$. From the description of the procedure, we have the following recurrence relation: $T(i) = nB \cdot 2^d(2T(i-1) + O(n))$. This follows from the fact that we have n guesses for the separator vertex, B guesses for the split in the cost of the left and right subtree and at most 2^d guesses on the responsibility set assigned to each subtree (since $|Y| \le d$). For every combination of the guesses, we make 2 recursive calls. Applying the induction hypothesis, we get $T(i) = nB \cdot 2^d(2 \cdot (c \cdot (nB \cdot 2^{d+2})^{i-1}) + O(n)) \le c \cdot (nB \cdot 2^{d+2})^i$ which completes the induction. ■

Lemma 2.1 *Let T be the arborescence returned by $RG(r, Y, B, X, i)$. Let T^* be a compatible arborescence for the parameters (r, Y, B, X, i), i.e. T^* is an r-rooted arborescence that visits all vertices in Y, and contains at most $(\frac{3}{2})^i$ non-root vertices with a total cost of at most B . Then $f_X(T) \ge f_X(T^*)/i$.*

Proof. We prove the lemma by induction on i. For the base case, let $i = 1$. Since T^* is feasible for $i = 1$, T^* is either empty or contains a single edge. If $|Y| = 0$, then we guess the base-case vertex and return the one that maximizes f_X subject to the given budget: so $f_X(T) \ge f_X(T^*)$ in this case. If $|Y| = 1$, then T^* has a single edge, say (r, v). Our procedure here will return the arborescence with (r, v), and so $f_X(T) = f_X(T^*)$. Thus, in either case, we get $f_X(T) \ge f_X(T^*)$ which proves the base case.

Suppose that $i > 1$. Let v be the vertex in T^* obtained from Fact 2.1 such that we can separate T^* into two connected components: T_1^* containing r and $T_2^* = T^* \setminus T_1^*$, where $\max(|V(T_1^*)|, |V(T_2^*)|) \le \frac{2}{3}|V(T^*)|$. Note that T_1^* is an r-rooted arborescence that contains v and T_2^* is a v-rooted arborescence. Let $Y_2 \subseteq Y \setminus \{v\}$ be those vertices of $Y \setminus v$ that are contained in T_2^*, and let $Y_1 = Y \setminus Y_2$. Because T^* contains Y, it is clear that $\{v\} \cup Y_1 \cup Y_2 \supseteq Y$. Finally, let $c(T_1^*) = B_1$ and $c(T_2^*) = B_2 \le B - B_1$. Note also that $|V(T^*) \setminus \{r\}| \le (\frac{3}{2})^i$. By the property of the separator vertex v, $\max(|V(T_1^*)|, |V(T_2^*)|) \le \frac{2}{3}|V(T^*)| \le (\frac{3}{2})^{i-1} + \frac{2}{3}$. Excluding the root vertex in T_1^* and T_2^*, the number of non-root vertices in either arborescence is $\le (\frac{3}{2})^{i-1}$. We can thus claim that:

T_1^* is compatible with

$$(r, Y_1 \cup \{v\} \setminus \{r\}, B_1, X, i-1) \text{ and} \tag{2.1}$$

T_2^* is compatible with

$$(v, Y \setminus (Y_1 \cup \{v\}), B - B_1, X \cup V(T_1), i-1). \tag{2.2}$$

Now consider the call $\text{RG}(r, Y, B, X, i)$. Since we iteratively set every vertex to be the separator vertex, one of the guesses is v. Moreover, we iterate over all subsets $S \subseteq Y$, and thus some guess must set $S = Y_1$. Since $B_1 \le B$, we also correctly guess B_1 in some iteration. Thus, we see that one of the set of calls made is

$$T_1 \leftarrow \text{RG}(r, Y_1 \cup \{v\} \setminus \{r\}, B_1, X, i-1) \text{ and}$$
$$T_2 \leftarrow \text{RG}(v, Y \setminus (Y_1 \cup \{v\}), B - B_1, X \cup V(T_1), i-1)$$

We now argue that $T = T_1 \cup T_2$ has the property that $f_X(T) \ge f_X(T^*)/i$. By (2.1) and induction,

$$f_X(T_1) \ge \frac{1}{i-1} f_X(T_1^*) \tag{2.3}$$

Let $X' = X \cup V(T_1)$. Similarly, by (2.2) and induction, we have

$$f_{X'}(T_2) \ge \frac{1}{i-1} f_{X'}(T_2^*) \tag{2.4}$$

The rest of this proof is identical to a corresponding result in [9]. We have $f_{X'}(T_2^*) = f(T_2^* \cup T_1 \cup X) - f(T_1 \cup X) = f_X(T_1 \cup T_2^*) - f_X(T_1)$. Using this in (2.4), we get

$$\begin{aligned} f_{X'}(T_2) &\ge \frac{1}{i-1}(f_X(T_1 \cup T_2^*) - f_X(T_1)) \\ &\ge \frac{1}{i-1}(f_X(T_2^*) - f_X(T)) \end{aligned} \tag{2.5}$$

where the last inequality follows from the monotonicity of the function f. We see that $f_X(T) = f_X(T_1 \cup T_2) = f(T_1 \cup T_2 \cup X) - f(X) + f(T_1 \cup X) - f(T_1 \cup X) = f_X(T_1) + f_{X'}(T_2)$. Thus using (2.3) and (2.5), we get

$$\begin{aligned} f_X(T) &\ge \frac{1}{i-1}(f_X(T_1^*) + f_X(T_2^*) - f_X(T)) \\ &\ge \frac{1}{i-1}(f_X(T^*) - f_X(T)) \end{aligned}$$

where the last inequality follows by the submodularity of f_X. On rearranging the terms, we get

$$f_X(T) \geq \frac{1}{i} f_X(T^*)$$

which concludes the induction. ∎

Remark 2.1, Proposition 2.2 and Lemma 2.1 imply:

Theorem 2.1 *There is a* $(\log_{1.5} k)$*-approximation algorithm for the submodular tree orienteering problem that runs in time* $O(nB)^{O(\log k)}$.

2.2 Quasi-Polynomial Time Algorithm Here we show how our algorithm can be implemented more efficiently in $(n \log B)^{O(\log k)}$ time. The idea here is the same as [9], but applied on top of Algorithm 1.

Algorithm 2 RG-QP(r, Y, B, X, i)

```
if (|Y| > (3/2)^i) then return Infeasible
if (i = 1) then
    if (|Y| = 0) then                 ▷ No responsibility for r
        pick v ∈ V : c(r, v) ≤ B that maximizes f_X(v), and return {(r, v)}   ▷ Guess base-case vertex
    if (|Y| = 1) then                 ▷ r must visit vertex v ∈ Y
        if (c(r, v) ≤ B) then return {(r, v)}
        else return Infeasible
T ← φ
m ← f_X(φ)
for each v ∈ V do                     ▷ Guess separator vertex
    for S ⊆ Y do                      ▷ Guess responsibilities for left/right subtrees
        for 1 ≤ u ≤ U do              ▷ Guess subtree function value
            B_1 ← min_b(RG-QP(r, (S ∪ {v}) \ {r}, b, X, i − 1) ≥ u)   ▷ Binary search for B_1
            if (B_1 = ∞) then continue
            T_1 ← RG-QP(r, (S ∪ {v}) \ {r}, B_1, X, i − 1)
            T_2 ← RG-QP(v, Y \ (S ∪ {v}), B − B_1, X ∪ V(T_1), i − 1)
            if (f_X(T_1 ∪ T_2) > m) then
                T ← T_1 ∪ T_2
                m ← f_X(T)
return T
```

The key idea here is that we no longer iterate through all values in $[1, B]$ to guess the recursive budget B_1. Instead, the step $B_1 \leftarrow \min_b(\text{RG-QP}(r, (S \cup \{v\}) \setminus \{r\}, b, X, i-1) \geq u)$ is implemented as a binary search over the range $[1, B]$. We assume that U is an upper bound on the function value. The following results are straightforward extensions of those in Section 2.1.

Proposition 2.3 *The running time of the procedure RG-QP*(r, Y, B, X, i) *is* $O((nU \cdot 2^d \cdot \log B)^i)$.

Lemma 2.2 *Let* T *be the arborescence returned by RG-QP*(r, Y, B, X, i)*. Let* T^* *be a compatible arborescence for the parameters* (r, Y, B, X, i)*, and* $f_X(T^*) \leq U$*. Then* $f_X(T) \geq f_X(T^*)/i$.

The proof of this lemma is similar to Lemma 2.1 and can be found in the appendix. Combining Proposition 2.3 and Lemma 2.2 and using polynomially bounded profits, we obtain:

Theorem 2.2 *There is an* $O(\log k)$*-approximation algorithm for the submodular tree orienteering problem that runs in time* $O(n \log B)^{O(\log k)}$.

2.3 Improved Approximation Ratio Here we show how to reduce the depth of our recursion at the cost of additional guessing. The high-level idea is the same as a similar result in [9], but we need some more care because our recursion is more complex.

Let $s = \epsilon \cdot \log \log k$ where $\epsilon > 0$ is some fixed constant. At each level of recursion, our new algorithm will guess all relevant quantities in s levels of the recursion in Algorithm 1. So the new recursion depth will be $d/s = O(\frac{\log k}{\log \log k})$ where $d = O(\log k)$ was the old depth. Recall that the number of subproblems generated at each level of recursion in Algorithm 1 is $2n2^dB$. Since we want to generate all subproblems in the next s levels of Algorithm 1, each subproblem in the new algorithm generates $(2n2^dB)^{2^s}$ many subproblems. As $d = O(\log k)$ and $s = \epsilon \cdot \log \log k$, the overall running time for the new algorithm is at most $(2n2^dB)^{2^s d} \leq (nB)^{O(\log^{1+\epsilon} k)}$.

Next, we will prove a lemma bounding the objective value at each level of the recursion. Below, $i \in \{1, 2, \cdots d/s\}$ denotes the depth allowed in any subproblem of the new recursion.

Lemma 2.3 *Let* T *be the arborescence returned by the improved approximation algorithm for parameters* (r, Y, B, X, i)*. Let* T^* *be some arborescence compatible with the same parameters. Then* $f_X(T) \geq f_X(T^*)/i$.

Proof. We will prove the claim by induction on i. For the base case, let $i = 1$. This is equivalent to the s^{th} level of the earlier algorithms, which implies that T^* contains at most $(\frac{3}{2})^s$ vertices excluding the root vertex. Since we guess all parameters for s levels of recursion, there exist guesses such that we can write $T^* = \cup_{j=0}^{2^s} T_j^*$ such that each T_j^* is either empty or contains a single edge. Since the edges in T_j^* are compatible with our guesses, and we will pick the best possible edge for T_j, we can conclude that $f_X(T) \geq f_X(T^*)$ which proves the base case.

Fix some $i > 1$. Consider the call to the new algorithm with the parameters (r, Y, B, X, i) where i denotes the new depth. Since T^* is compatible with the given parameters, one can iteratively obtain a choice of separator node v, responsibility set S and budget B' at each subproblem in the next s levels (exactly as in Lemma 2.1). This allows us to write $T^* = \cup_{j=1}^{2^s} T_j^*$ such that each T_j^* is compatible with some subproblem at new depth $(i-1)$. For each $j = 1, \ldots 2^s$ let T_j denote the solution returned by the j^{th} subproblem. The solution to the current subproblem is then $T = \cup_{j=1}^{2^s} T_j$. By induction, we have that $f_X(T_j) \geq f_{X_j}(T_j^*)/(i-1)$ where $X_j = X \cup (\bigcup_{a=1}^{j-1} T_a)$. Let $h = 2^s$. We will show below that

$$\sum_{j=1}^{h} f_{X_j}(T_j^*) \geq f_X(T^*) - f_X(T). \tag{2.6}$$

This would imply

$$\begin{aligned} f_X(T) = \sum_{j=1}^{h} f_{X_j}(T_j) &\geq \sum_{j=1}^{h} \frac{f_{X_j}(T_j^*)}{(i-1)} \\ &\geq \frac{1}{(i-1)}(f_X(T^*) - f_X(T)), \end{aligned}$$

which upon rearranging terms yields $f_X(T) \geq f_X(T^*)/i$ as desired.

To prove (2.6) consider

$$\begin{aligned} &\sum_{j=1}^{h} f_{X_j}(T_j^*) + f_X(T) \\ &= \sum_{j=1}^{h-1} f_{X_j}(T_j^*) + f_{X_h}(T_h^*) + f_X(T) \\ &= \Big(\sum_{j=1}^{h-1} f_{X_j}(T_j^*)\Big) + f(T_h^* \cup X \cup (\bigcup_{j=0}^{h-1} T_j)) \\ &\qquad - f(X \cup (\bigcup_{j=0}^{h-1} T_j)) + f(T \cup X) - f(X) \end{aligned}$$

applying submodularity to the 2^{nd} and 4^{th} term

$$\begin{aligned} &\geq \Big(\sum_{j=1}^{h-1} f_{X_j}(T_j^*)\Big) + f(T_h^* \cup X \cup T) \\ &\quad + f(X \cup (\bigcup_{j=0}^{h-1} T_j)) - f(X \cup (\bigcup_{j=0}^{h-1} T_j)) - f(X) \\ &= \Big(\sum_{j=1}^{h-1} f_{X_j}(T_j^*)\Big) + f(T_h^* \cup X \cup T) - f(X) \end{aligned}$$

inductively for all $k = h-1, \cdots 1, 0$ using the same steps as above

$$\begin{aligned} &\geq \Big(\sum_{j=1}^{k} f_{X_j}(T_j^*)\Big) + f((\bigcup_{j=k+1}^{h} T_j^*) \cup X \cup T) - f(X) \\ &\geq f(T^* \cup T \cup X) - f(X) \\ &\geq f(T^* \cup X) - f(X) = f_X(T^*). \end{aligned}$$

where the last inequality follows from the monotonicity of f. This completes the proof. ■

We further improve the runtime by applying the binary-search idea described in Section 2.2. Combining this with Lemma 2.3 and using polynomially bounded profits, we obtain Theorem 1.1.

3 Applications

Directed tree orienteering (DTO) This is the special case of STO when the reward function is linear, i.e. of the form $f(S) = \sum_{v \in S} p_v$ where each $v \in V$ has reward $p_v \in \mathbb{Z}$. So Theorem 1.1 applies directly to yield a quasi-polynomial time $O(\frac{\log k}{\log \log k})$-approximation algorithm. To the best of our knowledge, no non-trivial approximation ratio followed from prior techniques.

Directed Steiner tree Here, we are given a graph (V, E) with edge costs $c \in \mathbb{R}_+^E$, root r and a subset $U \subseteq V$ of terminals. The goal is to find an r-rooted arborescence that contains all of U and minimizes the total cost. By shortcutting over non-terminal vertices of degree at most two, we can assume that there is an optimal solution where every non-terminal vertex has degree at least three. So there is an optimal solution containing at most $2k$ vertices where $k = |U|$ is the number of terminals. We can use a standard set-covering approach to solve directed Steiner tree using DTO. We first guess (up to factor 2) a bound B on the optimal cost. Then we iteratively run the DTO algorithm with budget B and a reward of one for all *uncovered* terminal vertices. Assuming that the bound B is a correct guess, the optimal value of each DTO instance solved above equals k', the number of uncovered terminals in the current iteration. As we use a $\rho = O(\frac{\log k}{\log \log k})$ approximation for DTO, the number of iterations before covering all terminals is at most $O(\rho \cdot \log k)$. When all terminals have been covered, we return a min-cost arborescence in the union of all arborescences found so far. Using Theorem 1.1, this implies:

Theorem 3.1 *There is a deterministic $O(\frac{\log^2 k}{\log\log k})$-approximation algorithm for directed Steiner tree in $n^{O(\log^{1+\epsilon} k)}$ time, for any constant $\epsilon > 0$.*

Our approximation ratio matches that obtained recently [15]. Our algorithm is deterministic and has a better running time: the algorithm in [15] requires $n^{O(\log^5 k)}$ time. Moreover, our approach is much simpler. However, we note that an LP relaxation based approach as in [15] may have other advantages.

Polymatroid Directed Steiner tree This problem was introduced in [3] with applications in sensor networks. As before, we are given a directed graph (V, E) with edge costs $c \in \mathbb{R}_+^E$ and root r. In addition, there is a matroid defined on groundset V (same as the vertices) and the goal is to find a min-cost arborescence rooted at r that contains some base of the matroid. As matroid rank functions are submodular (and integer valued), we can apply Theorem 1.1 to obtain an $O(\frac{\log k}{\log\log k})$-approximation algorithm for the corresponding STO instance (reward-maximization), where $k \le |V|$ is the rank of the matroid. We then use a set-covering approach as outlined above, that iteratively solves STO instances until the set of covered vertices contains a base of the matroid. Crucially, the contraction of any matroid is another matroid: so the function f used in each such STO instance is still a matroid rank function. This yields an $O(\frac{\log^2 k}{\log\log k})$-approximation algorithm for polymatroid Steiner tree as well. This result improves over the $O(\log^3 k)$ ratio in [3].

4 Extensions of Submodular Tree Orienteering

In this section, we will consider two extensions of STO that involve additional length constraints. We then use this extension to obtain an improved approximation algorithm for directed buy-at-bulk network design and priority Steiner tree. Complete details and proofs can be found in the full version [].

4.1 STO with Length Constraints For the first extension, along with the input to STO, we are given a length function $\ell : E \to \mathbb{Z}_+$, and an additional bound L. Note that in an arborescence, given a vertex v, there is a unique path from the root to v. Let $p_T(v)$ denote the path from the root r^* to vertex v in arborescence T, and $l_T(v) = \sum_{e\in p_T(v)} \ell(e)$ represents the length of this path. The length constraint requires the sum of path lengths $l_T(v)$ to be at most L. More formally, the goal now is to find an out-directed arborescence T^* rooted at r^* maximizing $f(T^*)$ such that $c(T^*) \le B$ and $\sum_v l_{T^*}(v) \le L$. We will refer to this problem as STO with length constraints.

Theorem 4.1 *There is an $O(\log k)$-approximation algorithm for the submodular tree orienteering problem with length constraints that runs in time $O(nBL^2)^{O(\log k)}$.*

As mentioned in Section 2.2, since $f(\cdot)$ is assumed to be polynomially bounded in n, we can guess an upper bound U on the maximum function value. We can then guess the bound B_1 using binary search instead of enumerating through all values in the range $[1, B]$. Moreover, we will assume that L is polynomially bounded. This assumption will become clear when we use STO with distance constraints to solve the buy-at-bulk problem in directed graphs.

Moreover, as in Section 2.3, we can reduce the depth of our recursion at the cost of additional guessing to obtain Theorem 1.2.

4.2 STO with Deadlines For the second extension, along with the input to STO, we have a length function $\ell : E \to \mathbb{Z}_+$, and deadlines $\{d_v\}_{v\in V}$. We are able to claim the reward of a vertex v in arborescence T only if $l_T(v) \le d_v$. The goal of the problem is to find an out-directed arborescence T^* rooted at r^* maximizing $f(S(T^*))$ such that $c(T^*) \le B$ where $S(T^*) = \{v \in V : l_{T^*}(v) \le d_v\}$. We call this problem STO with Deadlines. Note that $l_T(v)$ and r^* are as defined in Section 4.1. We obtain the following result.

Theorem 4.2 *There is an $O(\frac{\log k}{\log\log k})$-approximation algorithm that runs in quasi-polynomial time for the submodular tree orienteering problem with deadlines.*

4.3 Single source Buy-at-Bulk Here we use the approximation algorithm for STO with length constraints to obtain an approximation algorithm for the single source buy-at-bulk problem in directed graphs. In this problem, we are given a directed graph $G(V, E)$, a set of terminals S and a source/root r^*. Moreover, each edge $e \in E$ is associated with a monotone concave cost function $g_e : \mathbb{R}_+ \to \mathbb{R}_+$. The goal is to route a unit of flow from r^* to each terminal in S while minimizing the total cost $\sum_{e\in E} g_e(x_e)$ where x_e denotes the total flow through edge e. It is straightforward to show (using concavity) that the edges carrying non-zero flow must form an r-arborescence. We adopt an alternative representation of the buy-at-bulk problem (at the loss of a constant factor in approximation) as described in [7, 19]. The input to the problem is now a directed multi-graph $G(V, E)$, a cost function $c : E \to \mathbb{R}_+$, a length function $\ell : E \to \mathbb{R}_+$, a set of terminals S, and a source r. The goal is to find an r-rooted arborescence T that has a directed path to all terminals such that $\sum_{e\in T} c(e) + \sum_{v\in S} \ell_T(v)$ is minimized. Here too, the function $\ell_T(\cdot)$ denotes the length of the path from r to v. Following a set covering approach as in Section 3 and Theorem 1.2 we obtain:

Theorem 4.3 *There is a quasi-polynomial time $O(\frac{\log^2 k}{\log\log k})$-approximation algorithm for the single-source buy-at-bulk problem in directed graphs.*

4.4 Priority Steiner tree This is a generalization of Steiner tree that has been used to model quality-of-service (QoS) considerations [5]. In the *priority* Steiner tree problem, we are given a directed graph $G(V, E)$ with edge-costs $\{c_e : e \in E\}$, a set of terminals S and a root r^*. There are p priority levels, with 1 denoting the lowest and p denoting the highest priority levels. Each edge e has a priority θ_e which denotes its QoS capability. Each terminal $t \in S$ also has a priority λ_t which denotes its QoS requirement. The goal is to find a minimum cost r^*-arborescence where the $r^* - t$ path for each terminal $t \in S$ has all edges with priority at least λ_t. Following a set covering approach as in Section 3 and a variant of Theorem 1.2 we obtain:

Theorem 4.4 *There is a quasi-polynomial time $O(\frac{\log^2 k}{\log\log k})$-approximation algorithm for the priority Steiner tree problem in directed graphs.*

References

[1] Spyridon Antonakopoulos. Approximating directed buy-at-bulk network design. In *Approximation and Online Algorithms*, pages 13–24, 2011.

[2] Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. Steiner tree approximation via iterative randomized rounding. *J. ACM*, 60(1):6:1–6:33, 2013.

[3] Gruia Călinescu and Alexander Zelikovsky. The polymatroid steiner problems. *J. Comb. Optim.*, 9(3):281–294, 2005.

[4] Moses Charikar, Chandra Chekuri, To-Yat Cheung, Zuo Dai, Ashish Goel, Sudipto Guha, and Ming Li. Approximation algorithms for directed steiner problems. *J. Algorithms*, 33(1):73–91, 1999.

[5] Moses Charikar, Joseph Naor, and Baruch Schieber. Resource optimization in qos multicast routing of real-time multimedia. *IEEE/ACM Trans. Netw.*, 12(2):340–348, 2004.

[6] Chandra Chekuri, Guy Even, and Guy Kortsarz. A greedy approximation algorithm for the group steiner problem. *Discrete Applied Mathematics*, 154(1):15–34, 2006.

[7] Chandra Chekuri, Mohammad Taghi Hajiaghayi, Guy Kortsarz, and Mohammad R. Salavatipour. Approximation algorithms for nonuniform buy-at-bulk network design. *SIAM J. Comput.*, 39(5):1772–1798, 2010.

[8] Chandra Chekuri, Nitish Korula, and Martin Pál. Improved algorithms for orienteering and related problems. *ACM Trans. Algorithms*, 8(3):23:1–23:27, 2012.

[9] Chandra Chekuri and Martin Pál. A recursive greedy algorithm for walks in directed graphs. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), 23-25 October 2005, Pittsburgh, PA, USA, Proceedings*, pages 245–253, 2005.

[10] Julia Chuzhoy, Anupam Gupta, Joseph Naor, and Amitabh Sinha. On the approximability of some network design problems. *ACM Trans. Algorithms*, 4(2):23:1–23:17, 2008.

[11] Jack Edmonds. Optimum branchings. *J. Res. Nat. Bur. Standards*, 71(B):233–240, 1967.

[12] Zachary Friggstad, Jochen Könemann, Young Kun-Ko, Anand Louis, Mohammad Shadravan, and Madhur Tulsiani. Linear programming hierarchies suffice for directed steiner tree. In *Integer Programming and Combinatorial Optimization - 17th International Conference, IPCO 2014, Bonn, Germany, June 23-25, 2014. Proceedings*, pages 285–296, 2014.

[13] Naveen Garg. Saving an epsilon: a 2-approximation for the k-mst problem in graphs. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 396–402, 2005.

[14] Naveen Garg, Goran Konjevod, and R. Ravi. A polylogarithmic approximation algorithm for the group steiner tree problem. *J. Algorithms*, 37(1):66–84, 2000.

[15] Fabrizio Grandoni, Bundit Laekhanukit, and Shi Li. O(log^{2}k/log log k)-approximation algorithm for directed steiner tree: A tight quasi-polynomial-time algorithm. *CoRR*, abs/1811.03020, 2018 (to appear in STOC 2019).

[16] Sudipto Guha, Adam Meyerson, and Kamesh Munagala. A constant factor approximation for the single sink edge installation problem. *SIAM J. Comput.*, 38(6):2426–2442, 2009.

[17] Eran Halperin and Robert Krauthgamer. Polylogarithmic inapproximability. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*, pages 585–594, 2003.

[18] David S. Johnson, Maria Minkoff, and Steven Phillips. The prize collecting steiner tree problem: theory and practice. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, January 9-11, 2000, San Francisco, CA, USA.*, pages 760–769, 2000.

[19] Adam Meyerson, Kamesh Munagala, and Serge A. Plotkin. Cost-distance: Two metric network design. *SIAM J. Comput.*, 38(4):1648–1659, 2008.

[20] Viswanath Nagarajan and R. Ravi. The directed orienteering problem. *Algorithmica*, 60(4):1017–1030, 2011.

[21] Alice Paul, Daniel Freund, Aaron Ferber, David B. Shmoys, and David P. Williamson. Budgeted Prize-Collecting Traveling Salesman and Minimum Spanning Tree Problems. *Mathematics of Operations Research (to appear)*, 2019.
[22] Gabriel Robins and Alexander Zelikovsky. Tighter bounds for graph steiner tree approximation. *SIAM J. Discrete Math.*, 19(1):122–134, 2005.
[23] Thomas Rothvoß. Directed steiner tree and the lasserre hierarchy. *CoRR*, abs/1111.5473, 2011.
[24] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970.
[25] Ola Svensson, Jakub Tarnawski, and László A. Végh. A constant-factor approximation algorithm for the asymmetric traveling salesman problem. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 204–213, 2018.
[26] Alexander Zelikovsky. A series of approximation algorithms for the acyclic directed steiner tree problem. *Algorithmica*, 18(1):99–110, 1997.
[27] Leonid Zosin and Samir Khuller. On directed steiner trees. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 6-8, 2002, San Francisco, CA, USA.*, pages 59–63, 2002.

Proof of Lemma 2.2

Proof. We prove the lemma by induction on i. For the base case, let $i = 1$. Since T^* is feasible for $i = 1$, T^* is either empty or contains a single edge. If $|Y| = 0$, then we guess the base-case vertex and return the one that maximizes f_X subject to the given budget: so $f_X(T) \geq f_X(T^*)$ in this case. If $|Y| = 1$, then T^* has a single edge, say (r, v). Our procedure here will return the arborescence with (r, v), and so $f_X(T) = f_X(T^*)$. Thus, in either case, we get $f_X(T) \geq f_X(T^*)$ which proves the base case.

Suppose that $i > 1$. Let v be the vertex in T^* obtained from Fact 2.1 such that we can separate T^* into two connected components: T_1^* containing r and $T_2^* = T^* \setminus T_1^*$, where $\max(|V(T_1^*)|, |V(T_2^*)|) \leq \frac{2}{3}|V(T^*)|$. Note that T_1^* is an r-rooted arborescence that contains v and T_2^* is a v-rooted arborescence. Let $Y_2 \subseteq Y \setminus \{v\}$ be those vertices of $Y \setminus v$ that are contained in T_2^*, and let $Y_1 = Y \setminus Y_2$. Because T^* contains Y, it is clear that $\{v\} \cup Y_1 \cup Y_2 \supseteq Y$. Finally, let $c(T_1^*) = B_1^*$ and $c(T_2^*) = B_2^* \leq B - B_1^*$. Note also that $|V(T^*) \setminus \{r\}| \leq (\frac{3}{2})^i$. By the property of the separator vertex v, $\max(|V(T_1^*)|, |V(T_2^*)|) \leq \frac{2}{3}|V(T^*)| \leq (\frac{3}{2})^{i-1} + \frac{2}{3}$. Excluding the root vertex in T_1^* and T_2^*, the number of non-root vertices in either arborescence is $\leq (\frac{3}{2})^{i-1}$. Let $f_X(T_1^*) = U_1$. We set B_1 in the algorithm using a binary search approach. Since we iterate over all values in $[1, U]$, one of the guesses, say u', is $= \lceil U_1/(i-1) \rceil$. By the induction hypothesis, the value of the arborescence returned by RG-QP$(r, Y_1 \cup \{v\} \setminus \{r\}, B_1^*, X, i-1) \geq U_1/(i-1)$. Also notice that RG-QP$(r, Y, b, X, i-1)$ is an increasing function in the parameter b (this allows us to use binary search to find B_1). Thus, the value $B_1 \leftarrow \min_b(\text{RG-QP}(r, Y_1 \cup \{v\} \setminus \{r\}, b, X, i-1) \geq u')$ has the property that $B_1 \leq B_1^*$.

We can thus claim that: T_1^* is compatible with

$$(r, Y_1 \cup \{v\} \setminus \{r\}, B_1, X, i-1) \text{ and} \tag{.1}$$

T_2^* is compatible with

$$(v, Y \setminus (Y_1 \cup \{v\}), B - B_1, X \cup V(T_1), i-1). \tag{.2}$$

Now consider the call RG(r, Y, B, X, i). Since we iteratively set every vertex to be the separator vertex, one of the guesses is v. Moreover, we iterate over all subsets $S \subseteq Y$, and thus some guess must set $S = Y_1$. From the above argument, one of the guesses $u \in [1, B]$ gives us $B_1 \leq B_1^*$. Thus, we see that one of the set of calls made is

$$T_1 \leftarrow \text{RG}(r, Y_1 \cup \{v\} \setminus \{r\}, B_1, X, i-1) \text{ and}$$

$$T_2 \leftarrow \text{RG}(v, Y \setminus (Y_1 \cup \{v\}), B - B_1, X \cup V(T_1), i-1)$$

We now argue that $T = T_1 \cup T_2$ has the property that $f_X(T) \geq f_X(T^*)/i$. By (.1) and induction,

$$f_X(T_1) \geq \frac{1}{i-1} f_X(T_1^*) \tag{.3}$$

Let $X' = X \cup V(T_1)$. Similarly, by (.2) and induction, we have

$$f_{X'}(T_2) \geq \frac{1}{i-1} f_{X'}(T_2^*) \tag{.4}$$

The rest of this proof is identical to the proof of Lemma 2.1. ∎