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

Rohan Ghuge, Viswanath Nagarajan

MATHEMATICS OF OPERATIONS RESEARCH
Vol. 47, No. 2, May 2022, pp. 1612–1630
ISSN 0364-765X (print), ISSN 1526-5471 (online)
http://pubsonline.informs.org/journal/moor

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

Rohan Ghuge,[a] Viswanath Nagarajan[a]

[a] Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, Michigan 48109
Contact: rghuge@umich.edu (RG); viswa@umich.edu, https://orcid.org/0000-0002-9514-5581 (VN)

Received: August 19, 2020
Revised: April 12, 2021
Accepted: May 12, 2021
Published Online in Articles in Advance: November 23, 2021

MSC2020 Subject Classification: Primary: 68M10, 90C27

https://doi.org/10.1287/moor.2021.1181

Abstract. We consider the following general network design problem. 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 simple quasi-polynomial time $O\left(\frac{\log k}{\log\log k}\right)$-approximation algorithm for this problem, in which $k \leq |V|$ is the number of vertices in an optimal solution. As a consequence, we obtain an $O\left(\frac{\log^2 k}{\log\log k}\right)$-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 $O\left(\frac{\log^2 k}{\log\log k}\right)$-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 but improves significantly on the running 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 nontrivial approximation ratio. Under certain complexity assumptions, our approximation ratios are the best possible (up to constant factors).

Funding: This work was supported by the National Science Foundation Division of Civil, Mechanical and Manufacturing Innovation [Grant CMMI-1940766] and Division of Computing and Communication Foundations [Grant CCF-1750127].

Keywords: approximation algorithms • network design • submodularity

1. Introduction

Network design problems, involving variants of the *minimum spanning tree* (MST) and *traveling salesman problem*, are extensively studied in approximation algorithms. These problems are also practically important as they appear in many applications, such as 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 (Edmonds [13]). 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 is to compute a minimum cost arborescence that contains all terminals. No polynomial-time poly-logarithmic approximation is known for directed Steiner tree. This is in sharp contrast with undirected Steiner tree, for which a two-approximation is folklore, and there are even better constant approximation ratios (Byrka et al. [3], Robins and Zelikovsky [26]). Most of the prior work on directed Steiner tree has focused on *quasi-polynomial* time algorithms, which have a running time of the form $N^{\log^b N}$, where N is the input size and b is some constant. Note that quasi-polynomial time algorithms are slower than polynomial-time algorithms but faster than exponential-time algorithms.

In this paper, we consider a variant of directed Steiner tree, in which 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. Any α-approximation 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 $O\left(\frac{\log k}{\log\log k}\right)$-approximation algorithm for DTO, where k is the number of vertices in an optimal solution. This also implies an $O\left(\frac{\log^2 k}{\log\log k}\right)$-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, in which one wants a path or tour of maximum profit subject to the cost limit, is much better understood. There are polynomial time approximation algorithms with guarantees $O(\log n)$ (Nagarajan and Ravi [24], Svensson et al. [29]) and $O(\log^2 k)$ (Chekuri et al. [10]). 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 the cost limit. The tour or path version of submodular orienteering was studied previously in Chekuri and Pál [8], in which a quasi-polynomial time $O\left(\frac{\log k}{\log k}\right)$-approximation algorithm was obtained. Although we rely on many ideas from Chekuri and Pál [8], 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 the following:

Theorem 1. *There is an* $O\left(\frac{\log k}{\log\log k}\right)$*-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 Chekuri and Pál [8] for the submodular *path* orienteering problem, which, in turn, is similar to the recursion used in Savitch's [28] theorem. In order to find an approximately optimal $s - t$ path with budget B, the algorithm in Chekuri and Pál [8] *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* orienteering problem that we consider, there are two additional issues:

• First, 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.

• Second (and more importantly), we cannot simply concatenate arbitrary solutions to the two subproblems. If r is the root of the original instance, the two subproblems find arborescences T_{left} and T_{right} rooted at r and v, respectively. Simply taking the union $T_{left} \cup T_{right}$ may lead to a disconnected graph. In order to obtain an r-arborescence, we additionally ensure that the subproblem with root r returns an arborescence T_{left} containing the separator node v. However, 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 is an $O\left(\frac{\log k}{\log\log k}\right)$-approximation algorithm for DTO and an $O\left(\frac{\log^2 k}{\log\log k}\right)$-approximation for directed Steiner tree. This matches the previous best bound (in quasi-polynomial time) for directed Steiner tree (Grandoni et al. [18]). 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. Grandoni et al. [18] 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 $NP \nsubseteq \cap_{0<\epsilon<1} ZPTIME(2^{n^\epsilon})$. Hence, Theorem 1 is also tight under the same assumptions.

Another application of Theorem 1 is to the directed polymatroid Steiner tree problem, in which there is a matroid with ground set V (same as the vertices), and one needs to find a minimum cost arborescence that visits some base of the matroid. We obtain a quasi-polynomial time $O(\log^2 k/\log\log k)$-approximation, which improves over the previous best $O(\log^3 k)$ bound (Călinescu and Zelikovsky [4]).

We also extend our main result (Theorem 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 is to find an arborescence on vertices T maximizing $f(T)$ in which (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 obtain the following:

Theorem 2. *There is an* $O\left(\frac{\log k}{\log\log k}\right)$*-approximation algorithm for submodular tree orienteering with length constraints that runs in* $(nL\log B)^{O(\log^{1+\epsilon} k)}$ *time for any constant* $\epsilon > 0$.

This algorithm follows a similar recursive structure as for STO, in which 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 $O\left(\frac{\log k}{\log\log k}\right)$-approximation for the variant of STO with hard deadlines on length (see Section 4.3 for details).

As a direct application of Theorem 2, we obtain a quasi-polynomial time $O\left(\frac{\log^2 k}{\log\log k}\right)$-approximation for single-source buy-at-bulk network design. This improves over the previous best $O(\log^3 k)$-approximation (Antonakopoulos [2]). Buy-at-bulk network design is a well-studied generalization of Steiner tree that involves concave cost-functions on edges. See Section 4.2 for more details. Our result holds for the harder "nonuniform" version of the problem, in which cost functions may differ across edges.

Another application of Theorem 2 is to the priority Steiner tree problem, in which 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 $O\left(\frac{\log^2 k}{\log\log k}\right)$-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 Grandoni et al. [18] that all our approximation ratios are tight (up to constant factors) for quasi-polynomial-time algorithms, assuming the projection game conjecture and $NP \nsubseteq \cap_{0<\epsilon<1} ZPTIME(2^{n^\epsilon})$.

1.2. Related Work

The first approximation algorithm for directed Steiner tree was obtained by Zelikovsky [30], and it gives 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 Charikar et al. [6], in which an $O(\log^3 k)$-approximation ratio was obtained in quasi-polynomial time: this was the first polylogarithmic approximation ratio. The algorithm in Charikar et al. [6] is a recursive procedure with a very different structure than ours: the idea there is 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 to obtain an algorithm for directed Steiner tree (at the loss of an additional $\ln k$-approximation factor). We note that the recursion used in Charikar et al. [6] relies on an explicit bound on the tree depth, which results in the loss of an additional logarithmic factor (which we save). Moreover, the approach of Charikar et al. [6] 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 linear programming (LP) relaxation of directed Steiner tree was shown to have an $\Omega(\sqrt{k})$ integrality gap by Zosin and Khuller [31]. Later, Friggstad et al. [14] 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, Rothvoß [27] used the stronger Lasserre hierarchy to obtain the same approximation ratio.) Very recently, Grandoni et al. [18] 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 Chekuri and Pál [8]. Our algorithm is easier to implement and has a much better running time. The approaches in Friggstad et al. [14], Grandoni et al. [18], and Rothvoß [27] are 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 (Garg et al. [16]), for which the best polynomial-time approximation ratio is $O(\log^2 k\cdot\log n)$. This is relative to the natural LP relaxation. A combinatorial algorithm with a slightly worse approximation ratio was given by Chekuri et al. [9]. In quasi-polynomial time, there is an $O(\log^2 k/\log\log k)$-approximation algorithm, which follows from Chekuri and Pál [8]. There is also an $\Omega(\log^{2-\epsilon} k)$-hardness of approximation for group Steiner tree (Halperin and Krauthgamer [20]). Recently, Grandoni et al. [18] showed that this reduction can be refined to prove an $\Omega(\log^2 k/\log\log k)$-hardness of approximation (under stronger assumptions).

Călinescu and Zelikovsky [4] 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 Charikar et al. [6]. 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 Friggstad et al. [14], Grandoni et al. [18], and Rothvoß [27], to address this problem.

Buy-at-bulk network design problems that involve concave cost functions are 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 (Guha et al. [19]) and an $O(\log k)$-approximation algorithm is known for the *nonuniform* version (Meyerson et al. [23]). The nonuniform problem is also hard to approximate better than $O(\log\log n)$ (Chuzhoy et al. [12]). For the directed case that we consider, the only prior result is Antonakopoulos [2], which implies a quasi-polynomial time $O(\log^3 k)$-approximation for the nonuniform version. Buy-at-bulk problems are also studied for multicommodity flows (Chekuri et al. [11]), which we do not consider in this paper.

The priority Steiner problem was introduced to model quality-of-service requirements in networking (Charikar et al. [5]). It is fairly well understood in the undirected setting: the best approximation ratio known is $O(\log n)$ (Charikar et al. [5]), and it is $\Omega(\log\log n)$-hard-to-approximate (Chuzhoy et al. [12]).

The DTO problem in undirected graphs has also received significant attention (Garg [15], Johnson et al. [21], Paul et al. [25]). In particular, a 2-approximation algorithm is known for it (Paul et al. [25]).

1.3. Preliminaries

The input to the submodular tree orienteering (STO) problem consists of (i) a directed graph $G=(V,E)$ with edge costs $c:E\rightarrow\mathbb{Z}_+$, (ii) root vertex $r^*\in V$, (iii) a budget $B\geq 0$, and (iv) a monotone submodular function $f:2^V\rightarrow\mathbb{Z}_+$ on the power set of the vertices. We assume throughout that all edge costs are integer valued. We also use the standard value-oracle model for submodular functions, which means that, for any $S\subseteq V$, our algorithm can evaluate $f(S)$ in polynomial time. Finally, we assume (for simplicity) that the maximum function value $U:=f(V)$ 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 (see Appendix B). An arborescence $T=(V(T),E(T))$ is a subgraph of G with a special vertex $r\in V(T)$ called its root, at which every vertex $v\in V(T)$ has a unique walk from r to v in T.

The goal in STO is to find an arborescence T^* rooted at r^* that maximizes $f(V(T^*))$ such that the cost of edges in T^* is at most B, that is, $\sum_{e\in E(T^*)}c(e)\leq B$. For an arborescence T, when it is clear from context, we also use T to denote its vertex-set $V(T)$; so $f(T)=f(V(T))$ is just the value of f evaluated at $V(T)$.

For any subsets $X\subseteq V$ and $S\subseteq V$, we use $f_X(S):=f(X\cup S)-f(X)$. For any $X\subseteq V$, the function f_X is also monotone and submodular.

2. Algorithms for Submodular Tree Orienteering

We first describe 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 many of 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 $O\left(\frac{\log k}{\log\log k}\right)$-approximation ratio in $(n \log B)^{O(\log^{1+\epsilon} k)}$-time.

2.1. The Main Algorithm

This is a recursive procedure, described formally in Algorithm 1. Each recursive call involves parameters (r, Y, B, X, i), where

- $r \in V$ is the current root node, which means that we find an r-rooted arborescence.
- B is an upper bound on the cost of the arborescence.
- $Y \subseteq V$ is a set of vertices that *must* be contained in the arborescence. We refer to set Y as the responsibility set for this recursive call.
- $X \subseteq V$ is the set of already-selected vertices. In this recursive call, we aim to maximize function $f_X(T) = f(T \cup X) - f(X)$, which represents the incremental value.
- i is the depth of recursion allowed, which means that the arborescence must contain at most $(\frac{3}{2})^i$ vertices, excluding the root.

An arborescence T is said to be *compatible* with the parameters (r, Y, B, X, i) if it is rooted at r, contains all vertices in Y, has cost at most B, and contains at most $(\frac{3}{2})^i$ nonroot vertices.

The key aspect of this recursion is the responsibility-set Y. Figure 1 gives an example of how the responsibility set changes over the recursive calls.

Algorithm 1 RG(r, Y, B, X, i)

```
1. if ($|Y| > (\frac{3}{2})^i$) then return infeasible
2. if $i = 1$ then
3.     if ($|Y| = 0$) then                                  ▷ No responsibility for $r$
4.         pick $v \in V : c(r,v) \le B$ that maximizes $f_X(v)$   ▷ Guess base-case vertex
5.         return $\{(r,v)\}$
6.     if ($Y = \{v\}$) then                                ▷ $r$ must visit vertex $v \in Y$
7.         if ($c(r,v) \le B$) then return $\{(r,v)\}$
8.         else return infeasible
9. $T \leftarrow$ infeasible and $m \leftarrow -\infty$
10. for each $v \in V$ do                                   ▷ Guess separator vertex
11.     for $S \subseteq Y$ do                              ▷ Guess responsibilities for left/right subtrees
12.         for $0 \le B_1 \le B$ do                        ▷ Guess subtree budget
13.             $T_1 \leftarrow$ RG$(r, (S \cup \{v\}) \setminus \{r\}, B_1, X, i-1)$
14.             $T_2 \leftarrow$ RG$(v, Y \setminus (S \cup \{v\}), B - B_1, X \cup T_1, i-1)$
15.             If ($f_X(T_1 \cup T_2) > m$) then $T \leftarrow T_1 \cup T_2$ and $m \leftarrow f_X(T)$.
16. return $T$
```

Remark 1. For any STO instance, our solution is RG$(r^*, \emptyset, B, \emptyset, d)$, where $d \ge \log_{1.5} k$ and k is the number of vertices in an optimal solution. We show that the approximation ratio is $\log_{1.5} k$ and the runtime is $(nB)^{O(\log k)}$. If k is not known in advance, we can use $d = \lceil \log_{1.5} n \rceil$: this provides the same approximation ratio at a slightly worse runtime of $(nB)^{O(\log n)}$.

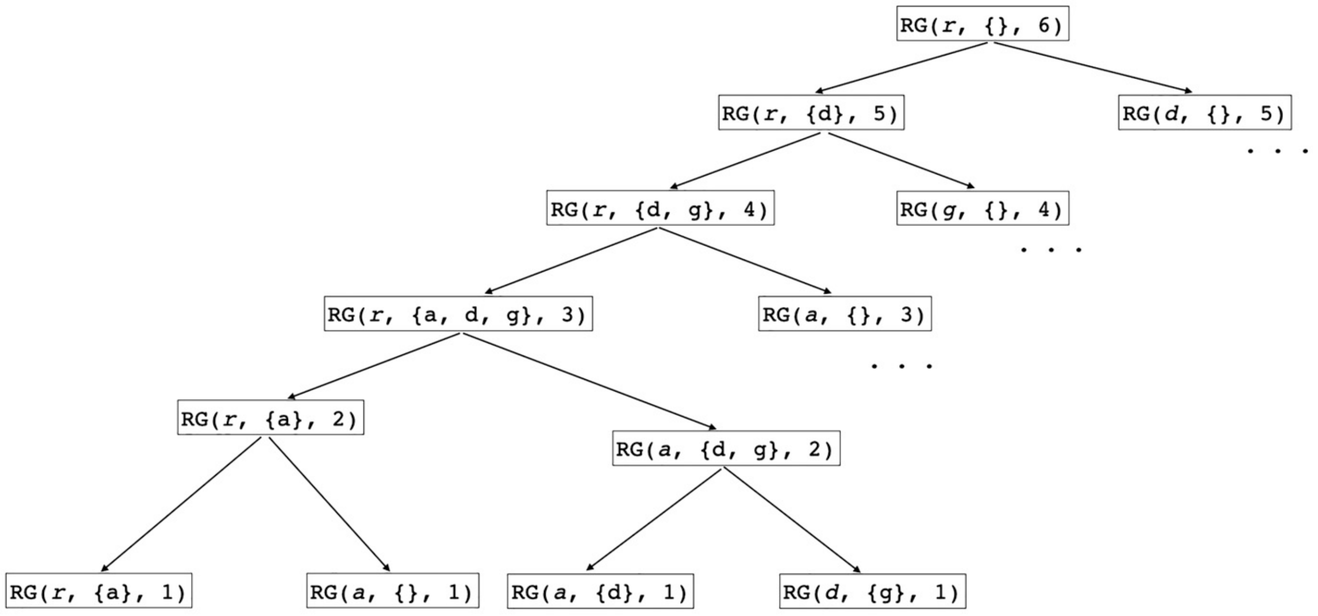
We use the following well-known fact about vertex separators in a tree.

Proposition 1. *Any tree on $n + 1 \ge 3$ vertices has a vertex v whose removal leads to connected components having size at most $\frac{n+1}{2}$ each. These components can be grouped together to form two connected components (both containing v) of size at most $1 + \frac{2n}{3}$ each.*

Lemma 1. *The maximum size of set Y in any subproblem of* RG$(r, \emptyset, B, \emptyset, d)$ *is d.*

Proof. To prove this statement, we argue that the invariant $|Y| + i \le d$ holds in every subproblem of RG$(r, \emptyset, B, \emptyset, d)$ of the form RG(r, Y, B, X, i). We prove this by induction on the depth $i = d, d-1, \cdots 1$. The base case has $i = d$. In this case, $|Y| = 0$, and thus, the aforementioned invariant clearly holds. Inductively, assume that the invariant holds for some depth $i > 1$. Let RG$(r', Y', B', X', i-1)$ be any depth $i - 1$ subproblem arising from a depth i subproblem RG(r, Y, B, X, i). From the algorithm, we can see that the size of the responsibility set increases

Figure 1. An example of the calls made by the recursive greedy algorithm. In each call, we show RG(r, Y, i), where r is the root node, Y is the responsibility set, and i is the recursion depth. If we keep following the left recursive call, the responsibility set increases steadily by one until it grows to a size of three, after which it decreases.



by at most one in any subproblem: so $|Y'| \leq 1 + |Y|$. Combined with the induction hypothesis $|Y| + i \leq d$, we get $|Y'| + i - 1 \leq |Y| + 1 + i - 1 \leq d$, which completes the induction.

Finally, as $i \geq 1$, we have $|Y| < d$ in any subproblem of RG$(r, \emptyset, B, \emptyset, d)$. $\square$

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

Proof. We prove this claim by induction on i. Let us denote the running time of RG(r, Y, B, X, i) by $T(i)$. We want to show that $T(i) \leq c \cdot (nB \cdot 2^{d+2})^i$ 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 subtrees, and at most 2^d guesses on the responsibility set assigned to each subtree (because $|Y| \leq d$). For every combination of guesses, we make two recursive calls. Applying the induction hypothesis, we get $T(i) = nB \cdot 2^d(2 \cdot (c \cdot (nB \cdot 2^{d+2})^{i-1}) + O(n)) \leq c \cdot (nB \cdot 2^{d+2})^i$, which completes the induction. $\square$

Lemma 3. *Let T be the arborescence returned by* RG(r, Y, B, X, i) *and let T^* be an arborescence compatible with the parameters (r, Y, B, X, i). Then, $f_X(T) \geq f_X(T^*)/i$.*

Proof. We prove the lemma by induction on i. First consider the base case $i = 1$. As T^* is compatible with $i = 1$, we have $T^* = \{r\}$ or $T^* = \{r, v^*\}$ for some vertex v^*. If $|Y| = 0$, then one of our guesses has $v = v^*$ (if v^* is present in T^*) and $f_X(T) \geq f_X(T^*)$ because we return the maximum value over all guesses. If $|Y| = 1$, then T^* must be of the form $\{r, v^*\}$, where $Y = \{v^*\}$. Here, the only possibility is $v = v^*$, so $f_X(T) = f_X(T^*)$. In either case, we get $f_X(T) \geq f_X(T^*)$.

Suppose now that $i > 1$. Let v be the vertex in T^* obtained from Proposition 1 such that we can separate T^* into two connected components: T_1^* containing r and $T_2^* = T^* \setminus T_1^*$. 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, $\max(|V(T_1^*)|,$

$|V(T_2^*)|) \le 1 + \frac{2}{3}|V(T^*) \setminus \{r\}| \le (\frac{3}{2})^{i-1} + 1$. Excluding the root vertex in T_1^* and T_2^*, the number of nonroot vertices in either arborescence is at most $(\frac{3}{2})^{i-1}$. We can, thus, claim that

$$T_1^* \text{ is compatible with } (r, Y_1 \cup \{v\} \setminus \{r\}, B_1, X, i-1) \text{ and} \tag{1}$$
$$T_2^* \text{ is compatible with } (v, Y \setminus (Y_1 \cup \{v\}), B - B_1, X \cup T_1, i-1). \tag{2}$$

Now consider the call RG(r, Y, B, X, i). Because 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 $S = Y_1$. Because $B_1 \le B$, we also correctly guess B_1 in some iteration. Thus, we see that one of the set of calls made is

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

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

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

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

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

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 (4),

$$f_{X'}(T_2) \ge \frac{1}{i-1}(f_X(T_1 \cup T_2^*) - f_X(T_1)) \ge \frac{1}{i-1}(f_X(T_2^*) - f_X(T)), \tag{5}$$

where the last inequality follows from the monotonicity of the function f.

We also have

$$f_X(T) = f_X(T_1 \cup T_2) = f(T_1 \cup T_2 \cup X) - f(X) + f(T_1 \cup X) - f(T_1 \cup X) = f_X(T_1) + f_{X'}(T_2).$$

Thus, using (3) and (5), we get

$$f_X(T) \quad \ge \quad \frac{1}{i-1}(f_X(T_1^*) + f_X(T_2^*) - f_X(T)) \quad \ge \quad \frac{1}{i-1}(f_X(T^*) - f_X(T)),$$

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

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

which concludes the induction. □

Lemmas 2 and 3 imply the following:

Theorem 3. *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 in Chekuri and Pál [8] but applied on top of Algorithm 1.

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

```
1. if ($|Y| > (\frac{3}{2})^i$) then return infeasible
2. if ($i = 1$) then
3.     if ($|Y| = 0$) then                              ▷ No responsibility for $r$
4.         pick $v \in V : c(r, v) \le B$ that maximizes $f_X(v)$   ▷ Guess base-case vertex
5.         return $\{(r, v)\}$
6.     if ($Y = \{v\}$) then                            ▷ $r$ must visit vertex $v \in Y$
7.         if ($c(r, v) \le B$) then return $\{(r, v)\}$
8.         else return infeasible
9. $T \leftarrow$ infeasible and $m \leftarrow -\infty$
10. for each $v \in V$ do                               ▷ Guess separator vertex
11.     for $S \subseteq Y$ do                          ▷ Guess responsibilities for left/right subtrees
```

```
12.     for $0 \le u \le U$ do                                          ▷ Guess subtree function value
13.         $B_1 \leftarrow \min_b(\text{RG-QP}(r, (S \cup \{v\}) \setminus \{r\}, b, X, i-1) \ge u)$    ▷ Binary search for $B_1$
14.         if ($B_1 = \infty$) then continue
15.         $T_1 \leftarrow \text{RG-QP}(r, (S \cup \{v\}) \setminus \{r\}, B_1, X, i-1)$
16.         $T_2 \leftarrow \text{RG-QP}(v, Y \setminus (S \cup \{v\}), B - B_1, X \cup T_1, i-1)$
17.         If ($f_X(T_1 \cup T_2) > m$) then $T \leftarrow T_1 \cup T_2$ and $m \leftarrow f_X(T)$.
18. return $T$.
```

The key idea here is that we no longer iterate through all values in $[0, B]$ to guess the recursive budget B_1. Instead, the step $B_1 \leftarrow \min_b(\text{RG-QP}(r, (S \cup \{v\}) \setminus \{r\}, b, X, i-1) \ge u)$ is implemented as a binary search over the range $\{1, 2, \cdots B\} \cup \{\infty\}$. We can perform binary search in this step because $\text{RG-QP}(r, (S \cup \{v\}) \setminus \{r\}, b, X, i-1)$ is nondecreasing in b. We assume that U is an upper bound on the function value: by our assumptions (Section 1.3), this is a polynomial. The following results are straightforward extensions of those in Section 2.1.

Lemma 4. *The running time of the procedure RG-QP(r, Y, B, X, i) is $O((nU \cdot 2^d \cdot \log B)^i)$.*

Lemma 5. *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 $f_X(T^*) \le U$. Then, $f_X(T) \ge f_X(T^*)/i$.*

The proofs of these lemmas can be found in Appendix A. By Lemmas 4 and 5 and the fact that U is polynomially bounded, we obtain the following:

Theorem 4. *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 Chekuri and Pál [8], 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 new level ℓ of recursion, the new algorithm guesses all relevant quantities in s levels of the recursion in Algorithm 1. In particular, we guess the (r, Y, B) parameters for all these recursive calls. So the new recursion depth is $d/s = O\left(\frac{\log k}{\log\log k}\right)$, where $d = O(\log k)$ was the old depth. Recall that the number of guesses at each level of recursion in Algorithm 1 is $n2^dB$. Because we guess all parameters for the next s levels (that involve 2^s recursive subproblems), each level in the new algorithm makes $(n2^dB)^{2^s}$ guesses. As $d = O(\log k)$ and $s = \epsilon \cdot \log\log k$, the total number of recursive calls in the new algorithm is at most $(n2^dB)^{2^s d} \le (nB)^{O(\log^{1+\epsilon} k)}$. The base case ($\ell = 1$) in the new recursion corresponds to s levels of the old recursion, so we are looking for an arborescence with at most $(3/2)^s$ nonroot nodes (that also contains the responsibility set Y). This can be easily found by enumerating over all such possibilities, which are at most $n^{O(1.5^s)}$ in number, and choosing the maximum value solution. So the overall runtime is $(nB)^{O(\log^{1+\epsilon} k)}$.

Next, we prove a lemma bounding the objective value at each level of the new recursion. We use $\ell \in \{1, 2, \cdots d/s\}$ to index the level in the new recursion.

Lemma 6. *Let T be the arborescence returned by the new algorithm for parameters (r, Y, B, X, ℓ). If T^* is an arborescence compatible with these parameters, then $f_X(T) \ge f_X(T^*)/\ell$.*

Proof. We prove the claim by induction on ℓ. The base case $\ell = 1$ is trivial because we enumerate over all possibilities. Now, consider $\ell > 1$ and any call to the new algorithm with parameters (r, Y, B, X, ℓ). As 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 3). 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 level $(\ell - 1)$. For each $j = 1, \ldots 2^s$, let T_j denote our algorithm's solution to the jth level-$(\ell-1)$ subproblem. Then, the solution to the level-ℓ problem is $T = \cup_{j=1}^{2^s} T_j$. By induction, we have $f_{X_j}(T_j) \ge f_{X_j}(T_j^*)/(\ell - 1)$, where $X_j = X \cup (\cup_{a=1}^{j-1} T_a)$. Let $h = 2^s$. We will show that

$$\sum_{j=1}^{h} f_{X_j}(T_j^*) \ge f_X(T^*) - f_X(T). \tag{6}$$

This implies

$$f_X(T) = \sum_{j=1}^{h} f_{X_j}(T_j) \geq \sum_{j=1}^{h} \frac{f_{X_j}(T_j^*)}{(\ell-1)} \geq \frac{1}{(\ell-1)}(f_X(T^*) - f_X(T)),$$

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

To prove (6), consider

$$\begin{aligned}
\sum_{j=1}^{h} f_{X_j}(T_j^*) + f_X(T) &= \sum_{j=1}^{h-1} f_{X_j}(T_j^*) + f_{X_h}(T_h^*) + f_X(T) \\
&= \left(\sum_{j=1}^{h-1} f_{X_j}(T_j^*)\right) + f\left(T_h^* \cup X \cup \left(\bigcup_{j=0}^{h-1} T_j\right)\right) - f\left(X \cup \left(\bigcup_{j=1}^{h-1} T_j\right)\right) + f(T \cup X) - f(X) \\
&\text{applying submodularity to the 2nd and 4th term} \\
&\geq \left(\sum_{j=1}^{h-1} f_{X_j}\left(T_j^*\right)\right) + f(T_h^* \cup X \cup T) + f\left(X \cup \left(\bigcup_{j=1}^{h-1} T_j\right)\right) - f\left(X \cup \left(\bigcup_{j=1}^{h-1} T_j\right)\right) - f(X) \\
&= \left(\sum_{j=1}^{h-1} f_{X_j}\left(T_j^*\right)\right) + f_X(T_h^* \cup T) \\
&\text{inductively for all } k = h-1, \cdots 1, 0, \text{using the same steps as above} \\
&\geq \left(\sum_{j=1}^{k} f_{X_j}\left(T_j^*\right)\right) + f_X\left(\left(\bigcup_{j=k+1}^{h} T_j^*\right) \cup T\right) \\
&\text{using } k = 0 \text{ above} \\
&\geq f_X\left(\left(\bigcup_{j=1}^{h} T_j^*\right) \cup T\right) = f(T^* \cup T \cup X) - f(X) \\
&\geq f(T^* \cup X) - f(X) = f_X(T^*).
\end{aligned}$$

This completes the proof. □

Remark 2. An alternative proof for the induction step in Lemma 6 is as follows. We can view this problem as a submodular maximization problem constrained to a partition matroid. Let $\mathcal{T}_j$ be the set of feasible solutions to subproblem j for $j = 1, 2, \ldots, h = 2^s$. The feasible solutions or "meta" elements of $\mathcal{T}_j$ are trees themselves. We treat each such tree as a unique element so that the sets $\{\mathcal{T}_j\}$ are pairwise disjoint and, hence, form a partition matroid. For any $Z_1, \cdots Z_p \in \cup_{j=1}^{h} \mathcal{T}_j$, let $g(\{Z_1, Z_2, \ldots, Z_p\}) := f_X(\cup_{i=1}^{p} Z_i)$. Observe that the function $g(\cdot)$ over the meta elements is monotone and submodular because $f_X(\cdot)$ is monotone and submodular. We want to solve the following optimization problem:

$$\begin{array}{lll} \text{maximize} & g(T_1, \ldots, T_h) = f_X(\cup_{j=1}^{h} T_j) & \\ \text{subject to:} & T_j \in \mathcal{T}_j, & \forall\, j = 1, 2, \cdots h. \end{array} \tag{7}$$

We complete the proof by leveraging a result from Chekuri and Kumar [7] on a greedy algorithm for the *maximum coverage problem* under partition constraints. Although the result in Chekuri and Kumar [7] is stated only for coverage functions, it also holds for any monotone submodular function. For completeness, we provide a proof of this fact in Appendix C. This algorithm considers the parts in an arbitrary order and picks an element from each part that (approximately) maximizes the increase in the function value. If the element chosen from each part is an α-approximate maximizer, then this algorithm is an $(\alpha+1)$ approximation. Note that the steps in our new recursive algorithm for STO correspond exactly to this greedy algorithm applied to (7) when $\alpha = (\ell - 1)$, which follows from the inductive hypothesis. So, we obtain an ℓ-approximation to (7) that proves Lemma 6.

2.3.1. Quasi-Polynomial Time for Improved Approximation.

We can further improve the $(nB)^{O(\log^{1+\epsilon} k)}$ runtime to a truly quasi-polynomial time of $(n \log B)^{O(\log^{1+\epsilon} k)}$ by applying the binary-search idea described in Section 2.2. Note that the dependence on B in the previous algorithm is because of guessing the cost budgets in the subproblems.

We now modify the algorithm as follows. At each new level ℓ of recursion, we guess all the (r, Y) parameters for s levels of the old recursion (Algorithm 1). Note that we do not guess the budgets B. So the number of guesses in one level of the new algorithm is $(n2^d)^{2^s} = n^{O(\log^\epsilon k)}$, which is quasi-polynomial. Recall that $s = \epsilon \cdot \log\log k$. Consider any level-$\ell$ problem in the new recursion, denoted RG-IQP(r, Y, B, X, ℓ). This gives rise to $h = 2^s$-level $\ell - 1$ subproblems; we index these subproblems by $j = 1, \cdots h$ as in the proof of Lemma 6. For each subproblem j, let (r_j, Y_j) denote the guessed (r, Y) parameters. So far, we have not fully specified these subproblems because the budgets $B_1, \cdots B_h$ have not been guessed. We now handle the budgets using the binary-search idea from Section 2.2. For each subproblem $j = 1, \cdots h$, we guess a target $u_j \in \{0, 1, \cdots U\}$ on the "incremental function value." Note that the number of additional guesses is $U^{2^s} = U^{\log^\epsilon k}$, which is quasi-polynomial by our assumption on U. We then do the following for $j = 1, 2, \cdots h$:

1. Perform binary search on $b \in \{0, 1, \cdots B, \infty\}$ to find

$$B_j \leftarrow \min\{b : \text{RG-IQP}(r_j, Y_j, b, X_j, \ell - 1) \ge u_j\}, \text{ where } X_j = X \cup \left(\bigcup_{a=1}^{j-1} T_a\right).$$

2. If $B_j = \infty$, then abort the current guess.
3. $T_j \leftarrow \text{RG-IQP}(r_j, Y_j, B_j, X_j, \ell - 1)$.

At the end, we check whether $\sum_{j=1}^h B_j \le B$: if this is satisfied, then the current guess is said to be *successful*. Finally, we return the solution $T = \cup_{j=1}^h T_j$ corresponding to a successful guess with maximum value $f_X(T)$. Note that $f_X(T) = \sum_{j=1}^h f_{X_j}(T_j) \ge \sum_{j=1}^h u_j$.

The total number of recursive calls made in RG-IQP(r, Y, B, X, ℓ) is at most $(n2^d U)^{2^s} \cdot 2^s \cdot O(\log B)$. So the overall runtime is $(nU\log B)^{O(\log^{1+\epsilon} k)} = (n\log B)^{O(\log^{1+\epsilon} k)}$ as $U = poly(n)$.

For the performance guarantee, we claim that Lemma 6 also holds for RG-IQP. As in the proof of Lemma 6, there exists a guess for the (r_j, Y_j) parameters so that $T^* = \cup_{j=1}^h T_j^*$, where each T_j^* is compatible with subproblem j. Let B_j^* denote the cost of tree T_j^* for each subproblem j, so $\sum_{j=1}^h B_j^* \le B$. We show that there exists a successful guess $\{u_j^*\}_{j=1}^h$ for the incremental function values for which our solution $T = \cup_{j=1}^h T_j$ satisfies $f_X(T) \ge f(T^*)/\ell$, which would prove Lemma 6. For each j, define $u_j^* := \frac{1}{\ell-1} \cdot f_{X_j}(T_j^*)$. Recall that $X_j = X \cup (\cup_{a=1}^{j-1} T_a)$, where $T_1, \cdots T_{j-1}$ denote our algorithm's solutions for the first $j-1$ subproblems (with guesses $u_1^*, \cdots u_{j-1}^*$). Observe that

$$f_X(T) \ge \sum_{j=1}^h u_h^* = \frac{1}{\ell - 1} \sum_{j=1}^h f_{X_j}(T_j^*) \ge \frac{f_X(T^*) - f_X(T)}{\ell - 1},$$

where the last inequality is by (6), which holds for any $T^* = \cup_{j=1}^h T_j^*$, $T = \cup_{j=1}^h T_j$, X, and monotone submodular f. Rearranging, we obtain $f_X(T) \ge f_X(T^*)/\ell$ as needed. It remains to show that the guess $\{u_j^*\}_{j=1}^h$ is successful, that is, $\sum_{j=1}^h B_j \le B$, where B_j is the cost of our solution T_j to subproblem j. We show that $B_j \le B_j^*$ for each j, which would imply the desired result as $\sum_{j=1}^h B_j^* \le B$. To prove $B_j \le B_j^*$, consider subproblem RG-IQP$(r_j, Y_j, B_j^*, X_j, \ell - 1)$. Note that T_j^* is compatible with these parameters, so by induction, the value of this recursive call is at least $f_{X_j}(T_j^*)/(\ell - 1) = u_j^*$. As B_j is the minimum budget b for which RG-IQP$(r_j, Y_j, b, X_j, \ell - 1) \ge u_j^*$, we have $B_j \le B_j^*$.

This completes the proof of Theorem 1.

3. Applications

3.1. Directed Tree Orienteering (DTO)

This is the special case of STO when the reward function is linear, that is, 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 applies directly to yield a quasi-polynomial-time $O\left(\frac{\log k}{\log\log k}\right)$-approximation algorithm. To the best of our knowledge, no nontrivial approximation ratio followed from prior techniques.

3.2. Directed Steiner Tree

Here, we are given a graph (V, E) with edge costs $c \in \mathbb{Z}_+^E$, 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 using the metric completion of the graph, we may assume (without loss of generality) that the underlying graph is complete, and the costs c

satisfy triangle inequality. Shortcutting over nonterminal vertices of degree at most two, we can assume that there is an optimal solution in which every nonterminal vertex has degree at least three. So there is an optimal solution containing at most $2k$ vertices in which $k = |U|$ 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 two) 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 previously equals k', the number of uncovered terminals in the current iteration. As we use a $\rho = O\left(\frac{\log k}{\log\log k}\right)$ approximation for STO, 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, this implies the following:

Theorem 5. *There is a deterministic $O\left(\frac{\log^2 k}{\log\log k}\right)$-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 (Grandoni et al. [18]). Our algorithm is deterministic and has a better running time; the algorithm in Grandoni et al. [18] requires $n^{O(\log^5 k)}$ time. Moreover, our approach is much simpler. However, an LP-based approach as in Grandoni et al. [18] may have other advantages.

3.3. Polymatroid Directed Steiner Tree

This problem was introduced in Călinescu and Zelikovsky [4] with applications in sensor networks. As before, we are given a directed graph (V, E) with edge costs $c \in \mathbb{Z}_+^E$ and root r. In addition, there is a matroid defined on ground set 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 to obtain an $O\left(\frac{\log k}{\log\log k}\right)$-approximation algorithm for the corresponding STO instance (reward-maximization), in which $k \le |V|$ is the rank of the matroid. We then use a set-covering approach as outlined earlier 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 a quasi-polynomial time $O\left(\frac{\log^2 k}{\log\log k}\right)$-approximation algorithm for polymatroid Steiner tree as well. This result improves over the $O(\log^3 k)$ ratio in Călinescu and Zelikovsky [4].

4. Extensions of Submodular Tree Orienteering

In this section, we consider an extension of STO that involves additional length constraints. This is used to obtain improved approximation algorithms for buy-at-bulk network design, priority Steiner tree, and hop-constrained Steiner tree.

4.1. STO with Length Constraints

Here, along with the input to STO, we are given a *length function* $\ell : E \to \mathbb{Z}_+$ and an additional length-bound L. The lengths ℓ are unrelated to the edge costs $c : E \to \mathbb{Z}_+$. 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 let $l_T(v) = \sum_{e \in p_T(v)} \ell(e)$ denote the length of this path. The length constraint requires the sum of path lengths $l_T(v)$ to be at most L. Formally, the goal now is to find an arborescence T^* rooted at r^*, maximizing $f(T^*)$ such that the cost $c(T^*) \le B$ and sum of lengths $\sum_v l_{T^*}(v) \le L$. We refer to this problem as STO with length constraints.

Recall that, in the case of STO, we use a recursive algorithm that, in each step, guesses (i) a separator vertex v; (ii) the cost bounds B_1 and $B - B_1$ for the two arborescences that are rooted at r and v, respectively; and (iii) a partition of the responsibility set to assign to the two subproblems. For STO with length constraints, we additionally guess bounds L_1 and $L - L_1$ on the sum of lengths for these arborescences. As we need to bound the length of paths from the original root r^* (not the root r of the current subproblem), we also maintain the length k_r from r^* to r as a parameter in the recursion. Another issue is that, even if we guess the bounds L_1 and $L - L_1$ for the two subproblems (rooted at r and v) correctly, the actual length from r to v in the first subproblem (rooted at r) may not be the same as in the optimal solution. This is crucial because the length from r to v (in the subproblem rooted at r) affects the sum of lengths in the subproblem rooted at v. To deal with this, every time we add a separator vertex v to the responsibility set, we also guess a length bound to get from r to v and add this guess to a dictionary $\mathcal{D}$ (which must be satisfied by the recursive subproblems).

Recursive calls in this algorithm are denoted RG-DC$(r, k_r, Y, \mathcal{D}, B, L, X, i)$, where the parameters r, Y, B, X, i are the same as those described in Section 2.1. So we are looking for an r-rooted arborescence that contains all vertices in Y, has cost at most B, and contains at most 1.5^i nonroot vertices. The goal is to maximize the function $f_X(T) = f(T \cup X) - f(X)$, which is the incremental value over subset X. The role of the other parameters are as follows:

• L is an upper bound on the sum of lengths of $r^* - v$ paths over all vertices v in this arborescence.

• k_r indicates that the length of the path from r^* to r is at most k_r. This is used to enforce the constraint on the sum of lengths. Formally, if T is a solution to this subproblem and $l_T(v)$ denotes the length of the $r - v$ path in T, then we must have $\sum_{v \in T}(k_r + l_T(v)) \le L$.

• $\mathcal{D}$ is a dictionary of vertex-length pairs for all vertices in the responsibility set Y. It contains pairs $(w, D(w))$ for all $w \in Y$. The length of the r–w path in the arborescence must be at most $D(w)$ for each $w \in Y$.

Again, we say that arborescence T is compatible with parameters $(r, k_r, Y, \mathcal{D}, B, L, X, i)$ if T is rooted at r, visits all vertices in Y while respecting the length bounds in $\mathcal{D}$, has cost at most B, has sum of lengths $\sum_{v \in T}(k_r + l_T(v)) \le L$, and contains at most $(1.5)^i$ nonroot vertices. See Algorithm 3 for a formal description. Our overall solution is RG-DC$(r^*, 0, \emptyset, \emptyset, B, L, \emptyset, d)$, where $d \ge \log_{3/2} k$ and k is the number of vertices in an optimal solution (as before).

Algorithm 3 RG-DC$(r, k_r, Y, D, B, L, X, i)$

```
1. if (|Y| > (3/2)^i) then return infeasible
2. if i = 1 then
3.     if (|Y| = 0) then                                      ▷ No responsibility for r
4.         pick v ∈ V : c(r,v) ≤ B and k_r + ℓ(r,v) ≤ L that maximizes f_X(v)
5.         return {(r,v)}
6.     if (Y = {v}) then                                      ▷ r must visit vertex v ∈ Y
7.         if (c(r,v) ≤ B, k_r + ℓ(r,v) ≤ L and ℓ(r,v) ≤ D(v)) then
8.             return {(r,v)}
9.         else return infeasible
10. T ← infeasible and m ← −∞
11. for each v ∈ V do                                        ▷ Guess separator vertex
12.     for S ⊆ Y do                                          ▷ Guess responsibilities for subtrees
13.         for 0 ≤ B_1 ≤ B do                                ▷ Guess subtree cost budget
14.             for 0 ≤ L_1 ≤ L do                            ▷ Guess subtree length budget
15.                 for 0 ≤ d_1 ≤ L do                        ▷ Guess length from r to v
16.                     if v ∉ Y then set D(v) ← d_1; else update D(v) ← min{d_1, D(v)}
17.                     𝒟_1 ← {(w, D(w)) : w ∈ S ∪ v}         ▷ Length guesses for r-subtree
18.                     𝒟_2 ← {(w, D(w) − d_1) : w ∈ Y \ (S ∪ v)}   ▷ Length guesses for v-subtree
19.                     T_1 ← RG-DC(r, k_r, (S ∪ v) \ r, 𝒟_1, B_1, L_1, X, i − 1)
20.                     T_2 ← RG-DC(v, k_r + d_1, Y \ (S ∪ v), 𝒟_2, B − B_1, L − L_1, X ∪ T_1, i − 1)
21.                     If (f_X(T_1 ∪ T_2) > m) then T ← T_1 ∪ T_2 and m ← f_X(T).
22. return T
```

Lemma 7. *The running time of RG-DC$(r, k_r, Y, \mathcal{D}, B, L, X, i)$ is $O((nBL^2 \cdot 2^d)^i)$.*

The proof is very similar to that of Lemma 2, and hence, we omit it here.

Lemma 8. *Let T be the arborescence returned by RG-DC$(r, k_r, Y, \mathcal{D}, B, L, X, i)$ and let T^* be any arborescence compatible with $(r, k_r, Y, \mathcal{D}, B, L, X, i)$. Then, $f_X(T) \ge f_X(T^*)/i$.*

Proof. We prove the lemma by induction on i. The base case is $i = 1$. Here, we have $T^* = \{r\}$ or $T^* = \{r, v^*\}$ for some vertex v^*. If $|Y| = 0$, then one of our guesses has $v = v^*$ (if v^* is present in T^*), and $f_X(T) \ge f_X(T^*)$ because we return the maximum value over all guesses. If $|Y| = 1$, then T^* must be of the form $\{r, v^*\}$, where $Y = \{v^*\}$. Here, the only possibility is $v = v^*$, 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.

We now prove the inductive step for $i > 1$. Let v be the vertex in T^* obtained from Proposition 1 such that we can separate T^* into two connected components: T_1^* containing r and $T_2^* = T^* \setminus T_1^*$. Note that T_1^* is an r-rooted arborescence containing 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 $\{v\} \cup Y_1 \cup Y_2 \supseteq Y$. Let $c(T_1^*) = B_1^*$ and

$c(T_2^*) = B_2^* \le B - B_1^*$. Let $L_1^* = \sum_{v\in T_1^*}(k_r + l_{T^*}(v))$ be the sum of lengths of the vertices in T_1^*, and L_2^*, defined analogously, is the sum of lengths of the vertices in T_2^*. Observe that $L_1^* + L_2^* \le L$. Let $l_{T^*}(v) = l_{T_1^*}(v) = d_1^*$. Note that $l_{T_1^*}(u) \le D(u)$ for all $u \in Y_1$, so $d_1^* = l_{T_1^*}(v) \le D(v)$ if $v \in Y$. Define

$$\mathcal{D}_1^* = \{(w, D(w)) : w \in Y_1 \setminus v\} \cup \{(v, d_1^*)\} \quad \text{and} \quad \mathcal{D}_2^* = \{(w, D(w) - d_1^*) : w \in Y_2\}.$$

For each $u \in T_2^*$, note that the length of the $v - u$ path in T_2^* is $l_{T_2^*}(u) = l_{T^*}(u) - l_{T^*}(v)$. So $L_2^* = \sum_{u\in T_2^*}(k_r + l_{T^*}(u)) = \sum_{u\in T_2^*}\Big((k_r + d_1^*) + l_{T_2^*}(u)\Big)$. By Proposition 1, the number of nonroot vertices in both arborescences T_1^* and T_2^* is at most $(\frac{3}{2})^{i-1}$. From this discussion,

$$T_1^* \text{ is compatible with } (r, k_r, (Y_1 \cup v) \setminus r, \mathcal{D}_1^*, B_1^*, L_1^*, X, i-1) \text{ and} \tag{8}$$

$$T_2^* \text{ is compatible with } (v, k_r + d_1^*, Y \setminus (Y_1 \cup v), \mathcal{D}_2^*, B - B_1^*, L - L_1^*, X \cup T_1, i-1). \tag{9}$$

Now consider the call RG-DC$(r, k_r, Y, D, B, L, X, i)$. In one of the guesses, we have vertex v as earlier, subset $S = Y_1$, $B_1 = B_1^*$, $L_1 = L_1^*$, and $d_1 = d_1^*$. Thus, one of the calls made is

$$T_1 \leftarrow \text{RG-DC}(r, k_r, (Y_1 \cup v) \setminus r, \mathcal{D}_1^*, B_1^*, L_1^*, X, i-1) \text{ and}$$
$$T_2 \leftarrow \text{RG-DC}(v, k_r + d_1^*, Y \setminus (Y_1 \cup v), \mathcal{D}_2^*, B - B_1^*, L - L_1^*, X \cup T_1, i-1).$$

We now argue that $T = T_1 \cup T_2$ has $f_X(T) \ge f_X(T^*)/i$. By (8), (9), and induction,

$$f_X(T_1) \ge \frac{1}{i-1} f_X(T_1^*) \text{ and } f_{X'}(T_2) \ge \frac{1}{i-1} f_{X'}(T_2^*),$$

where $X' = X \cup T_1$. Adding these inequalities, the rest of the proof is identical to Lemma 3. □

Combining Lemmas 7 and 8 gives us the following.

Theorem 6. *There is an $O(\log k)$-approximation algorithm for the submodular tree orienteering problem with length constraints that runs in time $(nBL)^{O(\log k)}$.*

We can improve the approximation ratio and runtime, exactly as in Section 2.3 for STO. The only difference is that here we have more recursive parameters. As in Section 2.3, in each new level ℓ of recursion, we guess all parameters other than the cost budget for s levels of the old recursion (Algorithm 3). That is, we guess the parameters (r, k_r, Y, D, L, X) for 2^s recursive subproblems and check that these are compatible with the level-ℓ problem. The cost budgets B are not guessed: they are handled using binary search as described in Section 2.3.1. This leads to an $O\left(\frac{\log k}{\log\log k}\right)$-approximation algorithm in $O(nL\log B)^{O(\log^{1+\epsilon} k)}$-time, which proves Theorem 2.

4.2. Single-Source Buy-at-Bulk

Here, we use the approximation algorithm for STO with length constraints to obtain an improved approximation algorithm for the single-source buy-at-bulk problem. In this problem, we are given a directed graph (V, E), a set of terminals $S \subseteq V$, 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 nonzero 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 Chekuri et al. [11] and Meyerson et al. [23]. The input to the problem is now a directed multigraph (V, E), a cost function $c : E \to \mathbb{Z}_+$, a length function $\ell : E \to \mathbb{Z}_+$, 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. As for STO with length constraints, the function $\ell_T(\cdot)$ denotes the length of the $r - v$ path in T.

We follow a set-covering approach as in Section 3. We first guess an upper bound B on the optimal value, which implies the same bound on the cost $\sum_{e\in T} c(e)$ and the sum of lengths $\sum_{v\in S} \ell_T(v)$ of the optimal arborescence T. Using a binary search approach, the bound B is at most two times the optimal (for one of the guesses). Then, we iteratively run the algorithm for STO with length constraints with both cost and length bounds B and a reward of one for all uncovered terminal vertices. Notice that the term $\sum_{v\in S} \ell_T(v)$ in the objective only takes into account the terminal vertices and not all vertices in the arborescence. Algorithm 3 can easily be modified to incorporate this change. Assuming that the bound B is guessed correctly, the optimal value of each STO instance solved equals k', the number of uncovered terminals. As we use a

$\rho = O\left(\frac{\log k}{\log\log k}\right)$-approximation for STO with length constraints (Theorem 2), the number of iterations before covering all terminals is at most $O(\rho \cdot \log k)$.

Another (minor) issue is that Theorem 2 requires polynomially bounded lengths. In order to ensure this, we perform a standard scaling/rounding as follows. We first remove all edges $e \in E$ with $\ell_e > B$ as these are not used in an optimal solution. Let $\tilde{B} = B/n^4$ and $\tilde{\ell}_e = \lfloor \ell_e/\tilde{B} \rfloor$ for all $e \in E$. Note that the new lengths $\tilde{\ell}_e$ are integers between zero and n^4, so these are polynomially bounded. We run the algorithm from Theorem 2 on the instance with costs c_e, cost bound B, lengths $\tilde{\ell}_e$, and length bound $L = n^4$. Note that $\tilde{B} \cdot \tilde{\ell}_e \le \ell_e \le \tilde{B} \cdot \tilde{\ell}_e + \tilde{B}$ for all $e \in E$. It is clear that the optimal arborescence T to the buy-at-bulk instance is still feasible to this STO instance with the new length constraint. On the other hand, any arborescence $\tilde{T}$ satisfying the new length constraint has total length

$$\ell(\tilde{T}) \le \tilde{B} \cdot \tilde{\ell}(\tilde{T}) + n^2\tilde{B} \le n^4\tilde{B} + n^2\tilde{B} \le (1 + o(1))B,$$

where we use the fact that each path has at most n edges and there are at most n terminals (whose paths contribute to the objective).

Theorem 7. *There is an $O\left(\frac{\log^2 k}{\log\log k}\right)$-approximation algorithm for the single-source buy-at-bulk problem in directed graphs that runs in $n^{O(\log^{1+\epsilon} k)}$ time.*

4.3. STO with Deadlines and Hop-Constrained Steiner Tree

Here, we consider a variant of STO with length constraints in which vertices have hard deadlines on the path lengths. Along with the input to STO, we have a length function $\ell : E \to \mathbb{Z}_+$ and deadlines $\{d_v\}_{v\in V}$. The reward of a vertex v in arborescence T can only be claimed if the $r^* - v$ path length $l_T(v) \le d_v$. So the goal is to find an 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. We first point out the differences from STO with length constraints:

- We do not compute the function f on all the vertices in the arborescence T, but only the vertices $S(T) = \{v \in T : l_T(v) \le d_v\}$ that are "visited by their deadline."
- There is no sum of length bound on a feasible arborescence.

Our algorithm RG-DL for STO with deadlines is very similar to that in Section 4.1. Each recursive call in RG-DL involves parameters $(r, k_r, Y, \mathcal{D}, B, X, i)$ that have the same meaning as in Section 4.1 for STO with length constraints. Note that there is no length bound L in this algorithm. Although there is no length bound, for the recursive step, we still need to guess the length from the current root r to the separator vertex v. To see why this is needed, suppose T^* is the optimal arborescence rooted at r, and T_1^* and T_2^* are its components rooted at r and v, respectively. If we do not reach vertex v from r within length $l_{T^*}(v)$ in our solution (rooted at r), then we are not able to claim the reward contained in T_2^* as some of those deadlines may get violated. We also need to modify how already-selected vertices X are updated between the two recursive calls:

$$T_1 \leftarrow \text{RG-DL}(r, k_r, (S \cup v) \setminus r, \mathcal{D}_1, B_1, X, i-1),$$
$$T_2 \leftarrow \text{RG-DL}(v, k_r + d_1, Y \setminus (S \cup v), \mathcal{D}_2, B - B_1, X \cup S(T_1), i-1),$$

where $S(T_1) = \{v \in T_1 : l_{T_1}(v) \le d_v\}$. The base case also changes slightly (from Algorithm 3) to reflect the deadlines. If $|Y| = 0$, then we pick $v \in V$ with $c(r, v) \le B$ and $k_r + \ell(r, v) \le d_v$ that maximizes $f_X(v)$. If $Y = \{v\}$, then we return solution $\{r, v\}$ only if $c(r, v) \le B$ and $k_r + \ell(r, v) \le D(v)$.

The analysis is omitted as it is very similar to that in Section 4.1.

Theorem 8. *There is an $O\left(\frac{\log k}{\log\log k}\right)$-approximation algorithm that runs in $(n\ell_{max} \log B)^{O(\log^{1+\epsilon} k)}$ time for the submodular tree orienteering problem with deadlines.*

Using this, we obtain an improved approximation algorithm for the *hop-constrained* (or shallow-light) minimum Steiner tree problem (Althaus et al. [1], Gouveia [17], Kortsarz and Peleg [22]). Here, we are given a directed graph (V, E), a set of terminals $S \subseteq V$, a root r^*, and a positive integer H. We want to find a minimum cost r^*-arborescence spanning the set S with an additional constraint that the $r^* - v$ path has at most H edges for each terminal $v \in S$. This problem models the design of centralized computer systems with quality-of-service (QoS) constraints. The root node r^* models a central computing resource, and the terminals S are users that require access to the resource. The hop constraints guarantee a certain quality level to the users accessing the central computing resource.

We cast the hop-constrained Steiner tree problem as an instance of STO with deadlines. The directed graph (V, E), edge costs c, terminals S, and root r^* for STO with deadlines are as given by the input to the hop-constrained Steiner tree problem. We set length $\ell(e) = 1$ for all $e \in E$ and deadline $d_v = H$ for all $v \in V$. Combining Theorem 8 with the set-covering framework, we obtain the following:

Theorem 9. *There is an $O\left(\frac{\log^2 k}{\log\log k}\right)$-approximation algorithm for the hop-constrained Steiner tree problem in directed graphs that runs in $n^{O(\log^{1+\epsilon} k)}$ time.*

4.4. Priority Steiner Tree

This is another generalization of Steiner tree that has been used to model QoS considerations (Charikar et al. [5]). In the *priority* Steiner tree problem, we are given a directed graph (V, E) with edge costs $\{c_e : e \in E\}$, a set of terminals $S \subseteq V$, and a root r^*. There are p priority levels with one 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 in which, for each terminal $t \in S$, all edges on the $r^* - t$ path have priority at least λ_t.

We say that a vertex t is *priority-connected* in an r^*-arborescence if each edge in the $r^* - t$ path has priority at least λ_t. We first consider the "maximum coverage" version of priority Steiner tree, in which we are given a bound B on cost and want an arborescence that priority-connects the maximum number of terminals. We provide a quasi-polynomial time approximation algorithm for this problem (with a submodular objective f) by slightly modifying the algorithm in Section 4.1. Each recursive call here has parameters $(r, p_r, Y, \mathcal{D}, B, X, i)$, where the parameters r, Y, B, X, i are as before (see Section 2.1), and

• p_r is the minimum priority of any edge on the path from r^* (original root) to r (current root).

• $\mathcal{D}$ is the dictionary that contains vertex-priority pairs for all vertices in the responsibility set Y. It contains a pair $(w, D(w))$ for each $w \in Y$, which means that all edges on the $r - w$ path in the arborescence must have priority at least $D(w)$.

Compared with Algorithm 3, there is no longer a length constraint, so the parameters k_r and L are not needed here. Instead, we maintain the priority-level p_r at the current root and also priority requirements (instead of length bounds) in dictionary $\mathcal{D}$.

For the recursion, as before, we guess the separator vertex v, responsibilities $S \subseteq Y$, and budget B_1 for the r-rooted subproblem. In addition, we guess the priority level $q \le p_r$ of the separator v, which is passed to the two subproblems as follows: we set $D_1(v) = q$ for the r-rooted subproblem and $p_v = q$ for the v-rooted subproblem. We also need to modify how already-selected vertices X are updated between the two recursive calls:

$$T_1 \leftarrow \text{recurse with } (r, p_r, (S \cup v) \setminus r, \mathcal{D}_1, B_1, X, i-1),$$
$$T_2 \leftarrow \text{recurse with } (v, q, Y \setminus (S \cup v), \mathcal{D}_2, B - B_1, X \cup P(T_1), i-1),$$

where $P(T_1) = \{v \in T_1 : v \text{ is priority} - \text{connected}\}$. The base case $(i = 1)$ also changes slightly. If $|Y| = 0$, then we pick $v \in V$ with $c(r, v) \le B$ and $\min\{p_r, \theta_{(r,v)}\} \ge \lambda_v$ that maximizes $f_X(v)$. If $Y = \{v\}$, then we return solution $\{r, v\}$ only if $c(r, v) \le B$ and $\min\{p_r, \theta_{(r,v)}\} \ge D(v)$. Using an analysis similar to Section 4.1, we can show that this algorithm is an $O\left(\frac{\log k}{\log\log k}\right)$-approximation algorithm for the maximum-coverage version in $n^{O(\log^{1+\epsilon} k)}$ time.

Combined with the set-covering framework as before, we obtain the following:

Theorem 10. *There is an $O\left(\frac{\log^2 k}{\log\log k}\right)$-approximation algorithm for the priority Steiner tree problem in directed graphs that runs in $n^{O(\log^{1+\epsilon} k)}$ time.*

5. Conclusion

We obtained quasi-polynomial time approximation algorithms for several directed network design problems, including tree orienteering, Steiner tree, and single-source buy-at-bulk. Our approximation ratios are nearly best possible under certain complexity assumptions. Obtaining any poly-logarithmic approximation for these problems in polynomial time remains the main open question. Another interesting question is obtaining a polynomial-time k^ϵ-approximation algorithm for submodular tree orienteering (for any constant $\epsilon > 0$). Although such a result is known for directed Steiner tree, it is not known for the submodular orienteering version.

Acknowledgments
The authors thank Chandra Chekuri for pointing out the alternative proof of Lemma 6. A preliminary version of this paper appeared in the proceedings of the ACM-SIAM Symposium on Discrete Algorithms, 2020.

Appendix A. Missing Proofs

Proof of Lemma 4.
We prove the claim by induction on i. Let us denote the running time of RG-QP(r, Y, B, X, i) by $T(i)$. We want to show that $T(i) = O((nU2^d \log B)^i)$. For the base case, let $i = 1$. From the description in Algorithm 2, 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 Algorithm 2, we have the following recurrence relation: $T(i) \le nU2^d((\log B + 2)T(i-1) + O(n))$. Note that we no longer guess the value of the budgets for the subtrees. Instead, we guess a target $u \in \{0, 1, \ldots, U\}$ on the incremental function value of the left subtree, which is used to guess a budget on the left and right subtrees. The aforementioned recurrence relation now follows from the fact that we have n guesses for the separator vertex, U guesses for the incremental function value of the left subtree, and at most 2^d guesses on the responsibility set assigned to each subtree (because $|Y| \le d$). Thus, for every combination of guesses, we make at most $\log B + 2$ recursive calls. The $\log B$ recursive calls correspond to the binary search to determine the minimum budget B_1, and the remaining two recursive calls denote the calls made to obtain the partial solutions T_1 and T_2. Applying the induction hypothesis, we get $T(i) = nU2^d((\log B + 2) \cdot O((nU2^d \log B)^{i-1}) + O(n)) = O(nU2^d \log B)^i$, which completes the induction.

Proof of Lemma 5.
We prove the lemma by induction on i. For the base case, let $i = 1$. Because 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 returns the arborescence (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 Proposition 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 set B_1 in the algorithm using a binary search approach. Because we iterate over all values in $[1, U]$, one of the guesses equals $u' = \lceil \frac{f_X(T_1^*)}{i-1} \rceil$. For this guess u', using the inductive hypothesis (for level $i - 1$) and the fact that f is integer valued, the value of the arborescence returned by RG-QP$(r, Y_1 \cup \{v\} \setminus \{r\}, B_1^*, X, i-1)$ is at least u'. 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 $B_1 \leftarrow \min_b(\text{RG-QP}(r, Y_1 \cup \{v\} \setminus \{r\}, b, X, i-1) \ge u')$ has the property that $B_1 \le B_1^*$. Hence, $B - B_1 \ge B - B_1^* = B_2^*$, which implies that

$$T_2^* \text{ is compatible with } (v, Y \setminus (Y_1 \cup \{v\}), B - B_1, X \cup T_1, i-1). \tag{A.1}$$

Now consider the call RG(r, Y, B, X, i). Because 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 this argument, the guess $u = u'$ gives us $B_1 \le B_1^*$. Thus, we see that one of the set of calls made is

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

We now argue that $T = T_1 \cup T_2$ has the property that $f_X(T) \ge f_X(T^*)/i$. As the guess $u = u'$,

$$f_X(T_1) \ge u' \ge \frac{1}{i-1} f_X(T_1^*) \tag{A.2}$$

Let $X' = X \cup T_1$. By (A.1) and induction, we have

$$f_{X'}(T_2) \ge \frac{1}{i-1} f_{X'}(T_2^*) \tag{A.3}$$

The rest of this proof is identical to the proof of Lemma 3.

Appendix B. Polynomially Bounded Objective Function

We can ensure (at the loss of a small approximation factor) that the function f is integer valued and at most polynomial in n, the number nodes in the input graph. This is necessary to obtain a truly quasi-polynomial time algorithm using the binary search approach in Sections 2.3 and 2.2. We describe how this can be done for the $O\left(\frac{\log k}{\log\log k}\right)$-approximation algorithm from Section 2.3; the same idea also works for the $O(\log k)$-approximation algorithm in Section 2.2.

Given an arbitrary monotone submodular function $f : 2^V \to \mathbb{Z}_+$, we transform it into a function $\tilde{f}$ such that the transformation only loses a factor of $(1-\eta)$ in the approximation ratio (for any $\eta > 0$), and has the property that $\tilde{f}$ is integral and polynomially bounded. Let $M := \max_{u\in V} f(\{u\})$. Note that $f(V) \le nM$ by subadditivity. We assume that, for every $v \in V$, the arborescence (r^*, v) satisfies the budget constraint: so the optimal value $OPT \ge M$.

Recall from Section 2.3 that $k \le n$ is the size of the optimal solution, $\epsilon > 0$ is a constant, $s = \epsilon \cdot \log\log k$, $h = 2^s$, the old recursion depth (for Algorithm 2) is $d = \log_{1.5} k$, and the new recursion depth is $g = \frac{d}{s}$. Let $\delta := \frac{\eta M}{2(4h)^g}$. We now define function $\tilde{f}$ as follows: $\tilde{f}(S) := \lfloor \frac{f(S)}{\delta} \rfloor$ for any $S \subseteq V$. Note that the maximum function value is

$$U := \tilde{f}(V) \le \frac{nM}{\delta} = \frac{2n}{\eta}(4h)^g = \frac{2n}{\eta}2^{(s+2)g} \le \frac{2n}{\eta}2^{3d} = poly(n).$$

Thus, $\tilde{f}$ is integer valued in the interval $[0, U]$. Moreover, it follows that $\frac{f(S)}{\delta} - 1 \le \tilde{f}(S) \le \frac{f(S)}{\delta}$ for all $S \subseteq V$. Although $\tilde{f}$ may not be submodular, we have

$$\tilde{f}(A) + \tilde{f}(B) \ge \tilde{f}(A\cup B) + \tilde{f}(A\cap B) - 2, \quad \forall\, A, B \subseteq V.$$

In the proof of (6), we applied this submodular inequality h times. Thus, with respect to $\tilde{f}$, we obtain

$$\sum_{j=1}^{h} \tilde{f}_{X_j}(T_j^*) \ge \tilde{f}_X(T^*) - \tilde{f}_X(T) - 2h \tag{B.1}$$

Instead of Lemma 6, the following modification holds true for $\tilde{f}$. If T is the arborescence returned by the new algorithm for parameters (r, Y, B, X, ℓ) and T^* is an arborescence compatible with these parameters, then

$$\tilde{f}_X(T) \ge \frac{\tilde{f}_X(T^*)}{\ell} - \alpha(\ell), \tag{B.2}$$

where $\alpha(\ell) = (4h)^\ell/\ell$. To prove this, we proceed as in Lemma 6. By induction and applying (B.1),

$$\tilde{f}_X(T) = \sum_{j=1}^{h} \tilde{f}_{X_j}(T_j) \ge \sum_{j=1}^{h}\left(\frac{\tilde{f}_{X_j}(T_j^*)}{\ell-1} - \alpha(\ell-1)\right) \ge \frac{1}{\ell-1}\left(\tilde{f}_X(T^*) - \tilde{f}_X(T) - 2h\right) - h\cdot\alpha(\ell-1),$$

which on rearranging gives

$$\tilde{f}_X(T) \ge \frac{\tilde{f}(T^*)}{\ell} - \frac{h}{\ell}\Big((\ell-1)\cdot\alpha(\ell-1) + 2\Big) \ge \frac{\tilde{f}(T^*)}{\ell} - \alpha(\ell).$$

The last inequality uses the following calculation:

$$\frac{h}{\ell\cdot\alpha(\ell)}\Big((\ell-1)\cdot\alpha(\ell-1)+2\Big) = \frac{h((4h)^{\ell-1}+2)}{(4h)^\ell} \le \frac{3h(4h)^{\ell-1}}{(4h)^\ell} < 1.$$

This completes the proof of (B.2). Finally, if T denotes the algorithm's solution for the first recursive call (with $\ell = g$) and T^* is an optimal solution,

$$f(T) \ge \delta\cdot\tilde{f}(T) \ge \delta\cdot\left(\frac{\tilde{f}(T^*)}{g} - \frac{(4h)^g}{g}\right) \ge \delta\cdot\left(\frac{\frac{f(T^*)}{\delta} - 1}{g} - \frac{(4h)^g}{g}\right)$$
$$\ge \frac{f(T^*)}{g} - 2\frac{(4h)^g}{g}\cdot\delta = \frac{OPT}{g} - \frac{\eta M}{g} \ge \frac{OPT}{g}(1-\eta).$$

The second inequality follows from Equation (B.2), the equality uses the definition of δ, and the last inequality uses $OPT \ge M$.

Finally, we need to ensure that the running time is quasi-polynomial. Recall that the final running time of the algorithm in Section 2.3 is $(nU\log B)^{O(\log^{1+\epsilon} k)}$, where U is an upper bound on the objective value. By construction, $U = poly(n)$ for $\tilde{f}$, and so the overall running time is $(n\log B)^{O(\log^{1+\epsilon} k)}$, which is quasi-polynomial in n and $\log B$ as desired.

Appendix C. Submodular Function Maximization Constrained to a Partition Matroid

Here, we show (for completeness) that the result of Chekuri and Kumar [7] extends directly to submodular function maximization under a partition constraint ($\texttt{SFM}-\texttt{PC}$). Formally, we are given a ground set V, and subsets $\mathcal{S} = \{S_i \subseteq V\}_{i=1}^m$. Furthermore, we are given a partition of $\mathcal{S}$ into groups $G_1, \ldots, G_k$. We are also given a nonnegative, monotone submodular function $f : 2^V \to \mathbb{R}_+$. A solution $H \subseteq \mathcal{S}$ is feasible if $H \cap G_j = 1$ for $j = 1, \ldots, k$. The objective is to find a feasible solution H to maximize $f(\cup_{S\in H} S)$.

In the case that m, the number of given subsets of V, is exponential in $|V|$, the sets must be defined implicitly. We assume that there is a polynomial time *oracle* $\mathcal{A}$ that takes as input $U \subseteq V$ and an index $j \in [k]$ and returns a set $S \in G_j$ such that $f_U(S) \geq \frac{1}{\alpha} \cdot \max_{T\in G_j} f_U(T)$. Here, $\alpha \geq 1$ is an approximation parameter. In other words, we assume there is an α-approximation algorithm for finding a set in G_j with the maximum incremental function value. We refer to $\mathcal{A}$ as an α-approximate oracle. The greedy algorithm considers the groups in an arbitrary order.

Algorithm C.1 (Greedy Algorithm for $\texttt{SFM-PC}$ from Chekuri and Kumar [7])

```
1. $H \leftarrow \emptyset$, $U \leftarrow \emptyset$
2. for $j = 1, \ldots, k$ do
3.     $A_j \leftarrow \mathcal{A}(U, j)$
4.     $H \leftarrow H \cup \{A_j\}$, $U \leftarrow U \cup A_j$
5. return $H$
```

Theorem C.1. *If $\mathcal{A}$ is an α-approximate oracle, then Algorithm C.1 is an $(\alpha+1)$-approximation algorithm for submodular maximization constrained to a partition matroid.*

Proof. Let $H = \{A_1, \cdots A_k\}$ be the solution returned by Algorithm C.1, in which $A_j \in G_j$ for all $j \in [k]$. It is clear that H is a feasible solution. Let $\mathcal{O}$ denote an optimal solution, and let $O_j \in G_j$ be the set picked by $\mathcal{O}$ from G_j. Let $A^{(j)} = \cup_{r=1}^{j} A_r$. Thus, $A^{(j)}$ refers to the sets selected in the first j iterations. We overload notation to let $A = \cup_{r=1}^k A_r$ and $O = \cup_{r=1}^k O_r$. Because $\mathcal{A}$ is an α-approximate oracle, we have

$$f(A^{(j)}) - f(A^{(j-1)}) \geq \frac{1}{\alpha}\cdot\left(f(O_j \cup A^{(j-1)}) - f(A^{(j-1)})\right) \geq \frac{1}{\alpha}\cdot\left(f(O_j\cup A) - f(A)\right), \tag{C.1}$$

where the last inequality follows from the submodularity of f because $A^{(j-1)} \subseteq A$. Thus, we have

$$f(O) \leq f(A) + \sum_{j=1}^{k}\left(f(O_j \cup A) - f(A)\right) \tag{C.2}$$

$$\leq f(A) + \alpha \sum_{j=1}^{k} (f(A^{(j)}) - f(A^{(j-1)})) \tag{C.3}$$

$$\leq f(A) + \alpha f(A), \tag{C.4}$$

where (C.2) follows from submodularity of f, (C.3) follows from (C.1), and (C.4) follows on simplifying a telescoping sum and the fact that $f(\emptyset) \geq 0$. Finally, we obtain $f(A) \geq \frac{1}{\alpha+1}\cdot f(O)$ as desired.

References

[1] Althaus E, Funke S, Har-Peled S, Könemann J, Ramos EA, Skutella M (2005) Approximating k-hop minimum-spanning trees. *Oper. Res. Lett.* 33(2):115–120.
[2] Antonakopoulos S (2010) Approximating directed buy-at-bulk network design. Jansen K, Solis-Oba R, eds. *Approximation and Online Algorithms* (Springer), 13–24.
[3] Byrka J, Grandoni F, Rothvoß T, Sanità L (2013) Steiner tree approximation via iterative randomized rounding. *J. ACM* 60(1):1–33.
[4] Călinescu G, Zelikovsky A (2005) The polymatroid Steiner problems. *J. Combin. Optim.* 9(3):281–294.
[5] Charikar M, Naor J, Schieber B (2004) Resource optimization in QoS multicast routing of real-time multimedia. *IEEE/ACM Trans. Networks* 12(2):340–348.
[6] Charikar M, Chekuri C, Cheung T, Dai Z, Goel A, Guha S, Li M (1999) Approximation algorithms for directed Steiner problems. *J. Algorithms* 33(1):73–91.
[7] Chekuri C, Kumar A (2004) Maximum coverage problem with group budget constraints and applications. Jansen K, Khanna S, Rolim JDP, Ron D, eds. *Approximation, Randomization, and Combinatorial Optimization.* (Springer, Berlin), 72–83.
[8] Chekuri C, Pál M (2005) A recursive greedy algorithm for walks in directed graphs. *Proc. 46th Annual IEEE Sympos. Foundations Comput. Sci.* (IEEE, Washington), 245–253.
[9] Chekuri C, Even G, Kortsarz G (2006) A greedy approximation algorithm for the group Steiner problem. *Discrete Appl. Math.* 154(1): 15–34.
[10] Chekuri C, Korula N, Pál M (2012) Improved algorithms for orienteering and related problems. *ACM Trans. Algorithms* 8(3):1–27.
[11] Chekuri C, Hajiaghayi MT, Kortsarz G, Salavatipour MR (2010) Approximation algorithms for nonuniform buy-at-bulk network design. *SIAM J. Comput.* 39(5):1772–1798.

[12] Chuzhoy J, Gupta A, Naor J, Sinha A (2008) On the approximability of some network design problems. *ACM Trans. Algorithms* 4(2):1–17.
[13] Edmonds J (1967) Optimum branchings. *J. Res. National Bureau Standards* 71(B):233–240.
[14] Friggstad Z, Könemann J, Kun-Ko Y, Louis A, Shadravan M, Tulsiani M (2014) Linear programming hierarchies suffice for directed Steiner tree. *Proc. Integer Programming Combin. Optim. 17th Internat. Conf.* (Springer, Berlin) 285–296.
[15] Garg N (2005) Saving an epsilon: A 2-approximation for the k-MST problem in graphs. *Proc. 37th Annual ACM Sympos. Theory Comput.* (ACM, New York), 396–402.
[16] Garg N, Konjevod G, Ravi R (2000) A polylogarithmic approximation algorithm for the group Steiner tree problem. *J. Algorithms* 37(1): 66–84.
[17] Gouveia L (1998) Using variable redefinition for computing lower bounds for minimum spanning and Steiner trees with hop constraints. *INFORMS J. Comput.* 10(2):180–188.
[18] Grandoni F, Laekhanukit B, Li S (2019) O($\log^2$ k/log log k)-approximation algorithm for directed steiner tree: A tight quasi-polynomial-time algorithm. *ACM Sympos. Theory Comput.* (ACM, New York), 253–264.
[19] Guha S, Meyerson A, Munagala K (2009) A constant factor approximation for the single sink edge installation problem. *SIAM J. Comput.* 38(6):2426–2442.
[20] Halperin E, Krauthgamer R (2003) Polylogarithmic inapproximability. *Proc. 35th Annual ACM Sympos. Theory Comput.* (ACM, New York), 585–594.
[21] Johnson DS, Minkoff M, Phillips S (2000) The prize collecting Steiner tree problem: Theory and practice. *Proc. 11th Annual ACM-SIAM Sympos. Discrete Algorithms* (SIAM, Philadelphia), 760–769.
[22] Kortsarz G, Peleg D (1999) Approximating the weight of shallow Steiner trees. *Discrete Appl. Math.* 93(2–3):265–285.
[23] Meyerson A, Munagala K, Plotkin SA (2008) Cost-distance: Two metric network design. *SIAM J. Comput.* 38(4):1648–1659.
[24] Nagarajan V, Ravi R (2011) The directed orienteering problem. *Algorithmica 60*(4):1017–1030.
[25] Paul A, Freund D, Ferber A, Shmoys DB, Williamson DP (2020) Budgeted prize-collecting traveling salesman and minimum spanning tree problems. *Math. Oper. Res.* 45(2):576–590.
[26] Robins G, Zelikovsky A (2005) Tighter bounds for graph Steiner tree approximation. *SIAM J. Discrete Math.* 19(1):122–134.
[27] Rothvoß T (2011) Directed Steiner tree and the Lasserre hierarchy. http://arxiv.org/abs/1111.5473.
[28] Savitch WJ (1970) Relationships between nondeterministic and deterministic tape complexities. *J. Comput. System Sci.* 4(2):177–192.
[29] Svensson O, Tarnawski J, Végh LA (2018) A constant-factor approximation algorithm for the asymmetric traveling salesman problem. *Proc. 50th Annual ACM SIGACT Sympos. Theory Comput.* (ACM, New York), 204–213.
[30] Zelikovsky A (1997) A series of approximation algorithms for the acyclic directed steiner tree problem. *Algorithmica* 18(1):99–110.
[31] Zosin L, Khuller S (2002) On directed Steiner trees. *Proc. 13th Annual ACM-SIAM Sympos. Discrete Algorithms* (SIAM, Philadelphia), 59–63.