Maximizing a Submodular Function with Viability Constraints

Wolfgang Dvořák[1], Monika Henzinger[1], and David P. Williamson[2]

[1] Universität Wien, Fakultät für Informatik, Währingerstraße 29, A-1090 Vienna, Austria

[2] School of Operations Research and Information Engineering, Cornell University, Ithaca, New York, 14853, USA

Abstract. We study the problem of maximizing a monotone submodular function with viability constraints. This problem originates from computational biology, where we are given a phylogenetic tree over a set of species and a directed graph, the so-called food web, encoding viability constraints between these species. These food webs usually have constant depth. The goal is to select a subset of k species that satisfies the viability constraints and has maximal phylogenetic diversity. As this problem is known to be NP-hard, we investigate approximation algorithm. We present the first constant factor approximation algorithm if the depth is constant. Its approximation ratio is $(1-\frac{1}{\sqrt{e}})$. This algorithm not only applies to phylogenetic trees with viability constraints but for arbitrary monotone submodular set functions with viability constraints. Second, we show that there is no $(1-1/e+\epsilon)$-approximation algorithm for our problem setting (even for additive functions) and that there is no approximation algorithm for a slight extension of this setting.

1 Introduction

We consider the problem of maximizing a monotone submodular set function f over subsets of a ground set X, subject to a restriction on what subsets are allowed. As discussed below, this problem has been well-studied with constraints on the allowed sets that are *downward-closed*; that is, if S is allowed subset, then so is any $S' \subset S$. Here we study the problem of maximizing such a function with a constraint that is not downwards-closed. Specifically, we assume that there exists a directed acyclic graph D with the elements of X as nodes in the graph and only consider so-called *viable* sets of a certain size. A set S is viable if each element either has no outgoing edges in D or it has a path P to such an element with $P \subseteq S$. Such viability constraints are a natural way to model dependencies between elements, where an element can only contribute to the function if it appears together with specific other elements.

We are motivated by a problem arising in conservation biology. The problem is given as a rooted phylogenetic tree $\mathcal{T}$ with nonnegative weights on the edges, where the leaves of the tree represent species, and the weights represent genetic distance. Given a conservation limit k, we would like to select k species so as to maximize the overall phylogenetic diversity of the set, which is equivalent to maximizing the weight of the induced subtree on the k selected leaves plus the root. This problem, known as *Noah's Ark problem* [16], can be solved in polynomial time via a greedy algorithm [3, 12, 14]. Moulton, Semple, and Steel [10] introduced a more realistic extension of the problem which takes into account the dependence of various species upon one another in a food web. In this food web an arc is directed from species a towards species b if a's survival depend on species b's. Moulton et al. now consider selecting viable subsets given by the food web of size k, i.e. a species is viable if also at least one of its successors in the food web is preserved. Note that in real life the *depth* of the food web, i.e., the longest shortest path between any node in D and the "nearest" node with no out-edge, is constant (usually no larger than 30). Faller et al. [4] show that the problem of maximizing phylogenetic diversity with viability constraints is NP-hard, even in simple special cases with constant depth (e.g. the food web is a directed tree of constant depth).

Since phylogenetic diversity induces a monotone, submodular function on a set of species, this problem is a special case of the problem of maximizing a submodular function with viability constraints. There exists a long line of research on approximately maximizing monotone submodular functions with constraints. This line of work was initiated by Nemhauser et al. [11] in 1978; they give a greedy $(1-\frac{1}{e})$-approximation algorithm for maximizing a monotone submodular function subject to a cardinality constraint. Fisher et al. [6] introduced approximation algorithms for maximizing a monotone submodular function subject to matroid constraints (in which the set S must be an independent set in a single or multiple matroids). In recent work other types of constraints have been studied, as well as nonmonotone submodular functions; see the surveys by Vondrak [15] and Goundan and Schulz [7].

In our case, the viability constraints are *not downwards-closed* while most of the prior work studies downwards-closed constraints. One notable exception, where not downwards-closed constraints are considered, are matroid base constraints [9]. The viability constraint could be extended to be downwards-closed by simply defining every subset of a viable set to be allowable. However, this extension violates the exchange property of matroids and thus viability constraints also differ from matroid base constraints. Hence we consider a new type of constraint in submodular function maximization. We show how variants on the standard greedy algorithm can be used to derive approximation algorithms for maximizing a monotone, submodular function with viability constraints; thus we show that a new type of constraint can be handled in submodular function maximization.

Specifically we first present a scheme of $(1-\frac{1}{e^{p/(p+d-1)}})/2$ - approximation algorithms for monotone submodular set functions with viability constraints, where d is the depth of the food web and p is a parameter of the algorithm, such that the running time is exponential in p but is polynomial for any fixed p. For instance if we set $p = d$ we achieve a $(1-\frac{1}{\sqrt{e}})/2$ - approximation algorithm. We further present a variant of these algorithms which are $(1-\frac{1}{e^{p/(p+d-1)}})$ -

approximations, but whose running time is exponential in both d and p. For fixed $d{=}p$ this is polynomial and provides an $(1-\frac{1}{\sqrt{e}})$ - approximation algorithm.

Next by a reduction from the maximum coverage problem, we show that there is no $(1-1/e+\epsilon)$-approximation algorithm for the phylogenetic diversity problem with viability constraints (unless $\mathsf{P}=\mathsf{NP}$). Finally we consider a generalization of our problem where we additionally allow AND-constraints such as "species a is only viable if we preserve *both* species b and species c" and show that this generalization has no approximation algorithm (assuming $\mathsf{P}\neq\mathsf{NP}$) by a reduction from 3-SAT.

We define the problem more precisely in Section 2, introduce our algorithms in Section 3, and give the hardness results in Section 4. All omitted proofs can be found in the appendix.

2 Phylogenetic Diversity with Viability Constraints

We first give a formal definition of the problem.

Definition 1. *A (rooted)* phylogenetic tree $\mathcal{T}=(T,E_{\mathcal{T}})$ *is a rooted tree with root* r *and each non-leaf node having at least 2 child-nodes together with a weight function* w *assigning non-negative integer weights to the edges. Let* X *denote the set of leaf nodes of* $\mathcal{T}$*. For any set* $A\subseteq X$ *the operator* $\mathcal{T}(A)$ *yields the spanning tree of the set* $A\cup\{r\}$*, and by* $\mathcal{T}_E(A)$ *we denote the edges of this spanning tree. Then for any set* $S\subseteq X$ *the* phylogenetic diversity *is defined as*

$$\mathcal{PD}(S)=\sum_{e:e\in\mathcal{T}_E(S)} w(e)$$

A food web D *for the phylogenetic tree* $\mathcal{T}=(T,E_{\mathcal{T}})$ *is an acyclic directed graph* (X,E)*. A set* $S\subseteq X$ *is called viable if each* $s\in S$ *is either a sink (a node with out-degree* 0*) in* D *or there is a* $s'\in S$ *such that* $(s,s')\in E$*.*

Now our problem of interest is defined as follows.

Definition 2. *The* Optimizing Phylogenetic Diversity with Viability Constraints *(OptPDVC) problem is defined as follows. You are given a phylogenetic tree* $\mathcal{T}$ *and a food web* $D=(X,E)$*, and a positive integer* k*. Find a viable subset* $S\subseteq X$ *of size (at most)* k *maximizing* $\mathcal{PD}(S)$*.*

OptPDVC is known to be NP-hard [4], even for restricted classes of phylogenetic trees and dependency graphs.

First we study fundamental properties of the function $\mathcal{PD}$.

Definition 3. *The set function* $\mathcal{PD}(.|.):2^X\times 2^X\mapsto\mathbb{N}_0$ *for each* $A,B\subseteq X$ *is defined as* $\mathcal{PD}(A|B)=\mathcal{PD}(A\cup B)-\mathcal{PD}(B)$*.*

The intuitive meaning of $\mathcal{PD}(A|B)$ is the gain of diversity we get by adding the set A to the already selected species B.

We next recall the definition of submodular set functions.We call a set function *submodular* if

$$\forall A, B, C \subseteq \Omega : A \subseteq B \Rightarrow f(A \cup C) - f(A) \geq f(B \cup C) - f(B).$$

Proposition 1. *$\mathcal{PD}$ is a non-negative, monotone, and submodular function [2].*

Now consider the function $\mathcal{PD}(.|.)$. As $\mathcal{PD}(.)$ is monotone also $\mathcal{PD}(.|.)$ is monotone in the first argument and because of the submodularity of $\mathcal{PD}(.)$ the function $\mathcal{PD}(.|.)$ is anti-monotone in the second argument.

In the remainder of the paper we will not refer to the actual definition of the functions $\mathcal{PD}(.)$, $\mathcal{PD}(.|.)$, but only exploit monotonicity, submodularity and the fact that these functions can be efficiently computed.

Moreover we will consider a function VE (*viable extension*) which, given a set of species S, returns a viable set S' of minimum size containing S. In the simplest case where S consist of just one species it computes a shortest path to any sink node in the food web. We define the *depth* d of a food web (X, E) as

$$d(X, E) = \max_{s \in X} |\operatorname{VE}(\{s\})|.$$

If the food web is clear from the context we just write d instead of $d(X, E)$. Note that the problem remains NP-hard for instances with $d = 2$, even if $\mathcal{PD}$ is additive [4]. Finally we define the *costs* of adding a set of species A to a set B

$$c(A|B) = |\operatorname{VE}(B \cup A)| - |B|.$$

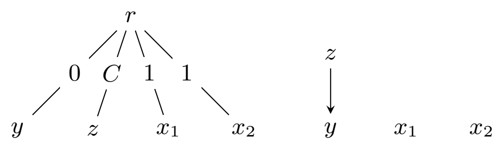
We will tacitly assume that $d \leq k$. Otherwise we can eliminate species s with $c(\{s\}|\emptyset) > k$ by polynomial time preprocessing, using a shortest path algorithm.

3 Approximation Algorithms

In this section we assume that a non-negative, monotone submodular function $\mathcal{PD}(.)$ is given as an oracle and we want to maximize $\mathcal{PD}(.)$ under viability constraints together with a cardinality constraint. We first review the greedy algorithm given by Faller et al. [4] presented in Algorithm 1. The idea is that, in each step, one considers only species which either have no successors in the food web or for which one of the successors has already been selected (adding one of these species will keep the set viable). Then one adds the species that gives the

Algorithm 1 Greedy, Faller et al.

```
1: $S \leftarrow \emptyset$
2: while $|S| < k$ do
3:     $s \leftarrow \operatorname{argmax}_{c(s|S)=1} v(\{s\}|S)$
4:     $S \leftarrow S \cup \{s\}$
5: end while
```



(a) Phylogenetic Tree (b) Food Web

Fig. 1. An illustration of Example 1.

largest gain of preserved diversity. By the restriction on the considered species the constructed set is always viable, but we might miss highly valuable species which is illustrated by the following example.

Example 1. Consider the set of species $X = \{y, z, x_1, x_2\}$ the phylogenetic tree $\mathcal{T} = (\{r\} \cup X, \{(r,y),(r,z),(r,x_1),(r,x_2)\}$, weights $w(r,x_i) = 1$, $w(r,y) = 0$, $w(r,z) = C$ with $C > 2$ and the food web $(X, \{(z,y)\})$ (see Fig. 1). Assume a budget $k = 2$, now as the species y has weight 0, Algorithm 1 would pick x_1, and x_2. Hence Algorithm 1 results a viable set with diversity 2. But the set $\{z, y\}$ is viable and has diversity C, which can be made arbitrarily large.

This example shows that the greedy solution might have an approximation ratio that is arbitrarily bad, because it ignores highly weighted species if they are "on the top of" less valuable species.

Hence, to get approximation ratio, we have to consider all subsets of species up to a certain size which can be made viable and pick the most valuable subset. Algorithm 2 deals with this observation. It generalizes concepts from [1], which itself builds on [8]. Lines 5 - 10 of the algorithm implement a greedy algorithm that in each step selects the most "cost efficient" subset of species of size p, i.e. the subset S of species that maximizes the ratio of the increase in PD over the cost of adding S, and adds it to the solution. But Algorithm 2 does not solely run the greedy algorithm, it first computes the set with maximal $\mathcal{PD}$ among

Algorithm 2

```
 1: $\mathcal{B} \leftarrow \{B \subseteq X \mid |B| \leq p, 1 \leq c(B|\emptyset) \leq k\}$
 2: $S \leftarrow \operatorname{argmax}_{B \in \mathcal{B}} \mathcal{PD}(B)$
 3: $\mathcal{G} = \mathrm{VE}(S)$
 4: $G \leftarrow \emptyset$
 5: while $\mathcal{B} \neq \emptyset$ do
 6:     $S \leftarrow \operatorname{argmax}_{B \in \mathcal{B}} \frac{\mathcal{PD}(B|G)}{c(B|G)}$
 7:     $G = \mathrm{VE}(G \cup S)$
 8:     $\mathcal{B} \leftarrow \{B \in \mathcal{B} \mid |B| \leq p, 1 \leq c(B|G) \leq k - |G|\}$
 9: end while
10: if $\mathcal{PD}(G) > \mathcal{PD}(\mathcal{G})$ then
11:     $\mathcal{G} \leftarrow G$
12: end if
```

all sets of size $\leq p$ that can be made viable. In certain cases this set is better than the viable set obtained by the greedy algorithm, a fact that we exploit in the proof of Theorem 1. In the algorithm G denotes the current set of selected species; $\mathcal{B}$ contains the species we might add to G; $\mathcal{G}$ denotes the best viable set we have found so far.

The next theorem will analyze the approximation ratio of Algorithm 2.

Theorem 1. *Algorithm 2 is a* $(1 - \frac{1}{e^{p/(p+d-1)}})/2$ *approximation (for any* $p \in \{1, \ldots, \lfloor k/3 \rfloor\}$*).*

To prove Theorem 1, we introduce some notation. First let $O \subseteq X$ denote the optimal solution. We will consider a decomposition $\mathcal{D}_O$ of O into sets $O_1, \ldots, O_{\lceil k/p \rceil}$ of size $\leq p$. By decomposition we mean that (i) $\bigcup_{i=1}^{\lceil k/p \rceil} O_i = O$ and (ii) $O_i \cap O_j = \emptyset$ if $i \neq j$. Moreover we require that $|\operatorname{VE}(O_i)| \leq p + d - 1$ and $\sum_i |\operatorname{VE}(O_i)| \leq \frac{k}{p}(p + d - 1)$. Next we show that such a decomposition $\mathcal{D}_O$ always exists.

Lemma 1. *There exist* $\lceil \frac{k}{p} \rceil$ *many pairs* $(O_1, B_1), \ldots, (O_{\lceil \frac{k}{p} \rceil}, B_{\lceil \frac{k}{p} \rceil})$ *such that* $O = \bigcup_{1 \leq i \leq \lceil \frac{k}{p} \rceil} O_i$, $O_i \cup B_i$ *is viable,* $|O_i| \leq p$, $|B_i| \leq d - 1$ *and* $\sum_i |O_i \cup B_i| \leq \frac{k}{p}(p + d - 1)$.

Proof. The optimal solution O is a viable subset of size at most k. Consider the reverse graph G of D projected on the set O i.e. $G = (O, E^- \cap (O \times O))$, and add an artificial root r that has an edge to all roots of G. Start a depth-first-search in r and with the empty sets O_1, B_1. Whenever the DFS removes a node from the stack, we add this node to the current set O_i, $i \geq 1$. When $|O_i| = p$ then we add the nodes on the stack, except r, to the set B_i, but do not change the stack itself. Then we continue the DFS with the next pair (O_{i+1}, B_{i+1}), again initialized by empty sets. Eventually the DFS stops, then the stack is empty and thus $(O_{\lceil \frac{k}{p} \rceil}, \emptyset)$ is the last pair. Notice that by the definition of d there are at most d nodes on the stack, one being the root r and hence $|B_i| \leq d - 1$. Since the DFS removes each node exactly once from the stack, all the sets $O_i \subseteq O$ are disjoint and all except the last one are of size p. Hence the DFS produces $\lceil \frac{k}{p} \rceil$ many sets O_i satisfying $O_i \cup B_i$ is viable, $|O_i| \leq p$ and $|B_i| \leq d - 1$. Finally as, by construction, $B_{\lceil \frac{k}{p} \rceil} = \emptyset$ and $|O_{\lceil \frac{k}{p} \rceil}| = k \mod p$ we obtain that $\sum_{i=1}^{\lceil \frac{k}{p} \rceil} |O_i \cup B_i| = \sum_{i=1}^{\lfloor \frac{k}{p} \rfloor} |O_i \cup B_i| + (k \mod p) \leq \lfloor \frac{k}{p} \rfloor (p + d - 1) + (k \mod p) \leq \frac{k}{p}(p + d - 1)$. □

First we consider the greedy algorithm and the value l where the l-th iteration is the first iteration such that after executing the loop body, $\max_{O_j \in \mathcal{D}_O} \frac{\mathcal{PD}(O_j|G)}{c(O_j|G)} > \max_{B \in \mathcal{B}} \frac{\mathcal{PD}(B|G)}{c(B|G)}$. If $\mathcal{G} \neq O$ the inequality holds at least for the last iteration of the loop where $\mathcal{B} = \emptyset$. We define $O'_{l+1} = \underset{O_j \in \mathcal{D}_O}{\operatorname{argmax}} \frac{\mathcal{PD}(O_j|G)}{c(O_j|G)}$, i.e. O'_{l+1} is in the optimal viable set and would be a better choice than the selection of the algorithm, but the greedy algorithm cannot make $S \cup O'_{l+1}$ viable without violating the cardinality constraint.

Let S_i denote the set S added to G in iteration i of the while loop in Line 6. Moreover, for $i \leq l$ we denote the set G after the i-th iteration by G_i, with $G_0 = \emptyset$, the set $G \cup S$ from Line 7 as $G_i^* = G_{i-1} \cup S_i$ and the "costs" of adding set S_i by $c_i = c(S_i|G_{i-1}) = c(G_i|G_{i-1})$. With a slight abuse of notation we will use G_{l+1} to denote the viable set $\mathrm{VE}(G_l \cup O'_{l+1})$, c_{l+1} to denote $c(O'_{l+1}|G_l)$ and G^*_{l+1} to denote the set $G_l \cup O'_{l+1}$ (G_{l+1} is not a feasible solution as $|G_{l+1}| > k$). Notice that while the sets G_i^* are not necessarily viable sets, all the $G_i, i \geq 0$ are viable sets and moreover $\mathcal{PD}(\mathcal{G}) \geq \mathcal{PD}(G_i), i \leq l$.

First we show that in each iteration of the algorithm the set S_i gives us a certain approximation of the missing part of the optimal solution.

Lemma 2. *For $1 \leq i \leq l+1$, $p \in \{1, \ldots, \lfloor k/3 \rfloor\}$:*

$$\frac{\mathcal{PD}(S_i|G_{i-1})}{c_i} \geq \frac{p}{(p+d-1)k} \cdot \mathcal{PD}(O|G_{i-1})$$

Proof. By the definition of S_i for each $O_j \in \mathcal{D}_O$ the following holds:

$$\frac{\mathcal{PD}(O_j|G_{i-1})}{c(O_j|G_{i-1})} \leq \frac{\mathcal{PD}(S_i|G_{i-1})}{c(G_i|G_{i-1})}$$

Next we use the monotonicity and submodularity of $\mathcal{PD}$ (for the first inequality) and the inequality from above (for the second inequality).

$$\begin{aligned}\mathcal{PD}(O|G_{i-1}) &\leq \sum_{O_j \in \mathcal{D}_O} \mathcal{PD}(O_j|G_{i-1}) = \sum_{O_j \in \mathcal{D}_O} \frac{\mathcal{PD}(O_j|G_{i-1})}{c(O_j|G_{i-1})}\, c(O_j|G_{i-1}) \\ &\leq \sum_{O_j \in \mathcal{D}_O} \frac{\mathcal{PD}(S_i|G_{i-1})}{c_i}\, c(O_j|G_{i-1}) \leq \frac{\mathcal{PD}(S_i|G_{i-1})}{c_i}\, \frac{p+d-1}{p} \cdot k\end{aligned}$$

The last step exploits that by Lemma 1 $\sum_{O_j \in \mathcal{D}_O} c(O_j|G_{i-1}) \leq \frac{k}{p} \cdot (p+d-1)$. □

Lemma 3. *For $1 \leq i \leq l+1$:*

$$\mathcal{PD}(G_i^*) \geq \left[1 - \prod_{j=1}^{i} \left(1 - \frac{p \cdot c_j}{(d+p-1) \cdot k}\right)\right] \mathcal{PD}(O)$$

Proof. The proof is by induction on i. The base case for $i = 1$ is by Lemma 2. For the induction step we show that if the claim holds for all $i' < i$ then it must also hold for i. For convenience we define $C_i = \frac{p \cdot c_i}{(d+p-1) \cdot k}$.

$$\begin{aligned}\mathcal{PD}(G_i^*) &= \mathcal{PD}(G_{i-1}) + \mathcal{PD}(G_i^*|G_{i-1}) \geq \mathcal{PD}(G_{i-1}) + C_i \cdot \mathcal{PD}(O|G_{i-1}) \\ &= \mathcal{PD}(G_{i-1}) + C_i \cdot (\mathcal{PD}(O \cup G_{i-1}) - \mathcal{PD}(G_{i-1})) \\ &\geq (1 - C_i) \cdot \mathcal{PD}(G_{i-1}) + C_i \cdot \mathcal{PD}(O) \geq (1 - C_i) \cdot \mathcal{PD}(G_{i-1}^*) + C_i \cdot \mathcal{PD}(O) \\ &\geq (1 - C_i) \left[1 - \prod_{j=1}^{i-1} (1 - C_j)\right] \mathcal{PD}(O) + C_i \cdot \mathcal{PD}(O) \\ &= \left[1 - \prod_{j=1}^{i} (1 - C_j)\right] \mathcal{PD}(O)\end{aligned}$$

□

Proof (Theorem 1). We first give a bound for G^*_{l+1}. To this end consider $\sum_{m=1}^{l+1} c_m$. As G_{l+1} exceeds the cardinality constraint $\sum_{m=1}^{l+1} c_m > k$ it follows that:

$$1 - \prod_{j=1}^{l+1}\left(1 - \frac{p \cdot c_j}{(d+p-1)\cdot(k)}\right) \geq 1 - \prod_{j=1}^{l+1}\left(1 - \frac{p \cdot c_j}{(d+p-1)\cdot \sum_{m=1}^{l+1} c_m}\right)$$
$$\geq 1 - \left(1 - \frac{p}{(d+p-1)\cdot(l+1)}\right)^{l+1} \geq 1 - \frac{1}{e^{p/(d+p-1)}}$$

To obtain Line 2 we used the fact the term $1 - \prod_{j=1}^{l+1}\left(1 - \frac{c'_j}{C}\right)$ with constant C and the constraint $\sum_{j=1}^{l+1} c'_j = 1$ has its maximum at $c'_j = 1/(l+1)$.

By Lemma 3 we obtain $\mathcal{PD}(G^*_{l+1}) \geq \left(1 - \frac{1}{e^{p/(d+p-1)}}\right) \cdot \mathcal{PD}(O)$, thus it only remains to relate $\mathcal{PD}(G^*_{l+1})$ to $\mathcal{PD}(G_l)$. To this end we consider the optimal set of size p computed in Line 3 and denote it by S_o. If the greedy solution has higher $\mathcal{PD}$ than S_o the algorithm returns a superset of G^*_l otherwise a superset of S_o. Hence, $\mathcal{PD}(G)$ is larger or equal to the maximum of $\mathcal{PD}(G^*_l)$ and $\mathcal{PD}(S_o)$. From the definitions of G^*_l and S_o it follows that

$$\mathcal{PD}(G^*_{l+1}) \leq \mathcal{PD}(G^*_l) + \mathcal{PD}(O'_{l+1}) \leq \mathcal{PD}(G^*_l) + \mathcal{PD}(S_o).$$

With the above result for $\mathcal{PD}(G^*_{l+1})$ we obtain that:

$$\mathcal{PD}(G) \geq \max(\mathcal{PD}(G^*_l), \mathcal{PD}(S_o)) \geq \left(1 - \frac{1}{e^{p/(d+p-1)}}\right) \cdot \frac{\mathcal{PD}(O)}{2}$$

Hence Algorithm 2 provides an $\left(1 - \frac{1}{e^{p/(d+p-1)}}\right)/2$ - approximation. □

Theorem 2. *Algorithm 2 is in time* $\mathcal{O}\left(k \cdot (3^p n^{p+2} + n^{p+1} m)\right)$.

Proof. First notice that computing the function VE can be reduced to a Steiner tree problem by (i) taking all the species in S that are already connected (via nodes in S) to a sink node in S, and merging these nodes into a single terminal node t, and (ii) connecting the remaining sink nodes to t. As starting nodes for the Steiner tree problem we use the remaining species in S. A viable set S is reduced to a single vertex t and thus the number of terminal nodes in the Steiner tree problems is bounded by a constant, hence we can solve them in polynomial time: The Steiner tree problem on acyclic directed graphs can be solved in time $\mathcal{O}(3^j n^2 + nm)$ [13], where j is the number of starting and terminal nodes. In Line 2 we have to consider $\mathcal{O}(n^p)$ sets S and for each of them we solve a Steiner tree problem. with at most p starting nodes. So this first loop can be done in time $\mathcal{O}(3^p n^{p+2} + n^{p+1} m)$. The number of iterations of the while loop is bounded by k and in each iteration, in Line 6, we have to solve $\mathcal{O}(n^p)$ Steiner tree problems with at most p starting nodes. Now as each iteration takes time $\mathcal{O}(3^p n^{p+2} + n^{p+1} m)$ we get a total running time of $\mathcal{O}(k \cdot (3^p n^{p+2} + n^{p+1} m))$. □

Finally, notice that one can use a modification of the enumeration technique as described in [8], to get rid of the factor 1/2 in the approximation ratio. The idea is to consider all (viable) sets of a certain size and for each of them to run the greedy algorithm starting with this set. Finally one chooses the best of the produced solutions. These sets typically have to contain three objects of interest, in the case of the maximum coverage problem [8] (cf. Def. 4 below) just three sets from the collection SC. However, in our setting an object of interest is a pair (O_i, B_i), i.e. a set O_i of size $\leq p$ and a set B_i of size $< d$ making O_i viable. Thus three objects result in a set of size of $3p + 3d - 3$. This increases the running time by a factor of $n^{3p+3d-3}$. The proof of the following theorem is very similar to the above analysis for Algorithm 2 (details are provided in the appendix).

Theorem 3. *There exists an* $\left(1 - \frac{1}{e^{p/(d+p-1)}}\right)$ *- approximation algorithm for OptPDVC which runs in time* $\mathcal{O}\left(k \cdot (3^p n^{4p+3d-1} + n^{4p+3d-2} m)\right)$.

4 Impossibility Results

If we allow arbitrary monotone submodular functions it is easy to see that no $1 - \frac{1}{e} + \epsilon$-approximation algorithm exists (unless $\mathsf{P} = \mathsf{NP}$). This is immediate by the corresponding result for Max Coverage (with cardinality constraints). Here we show that, when considering viability constraints, this also holds for additive functions and in particular for the phylogenetic diversity $\mathcal{PD}$.

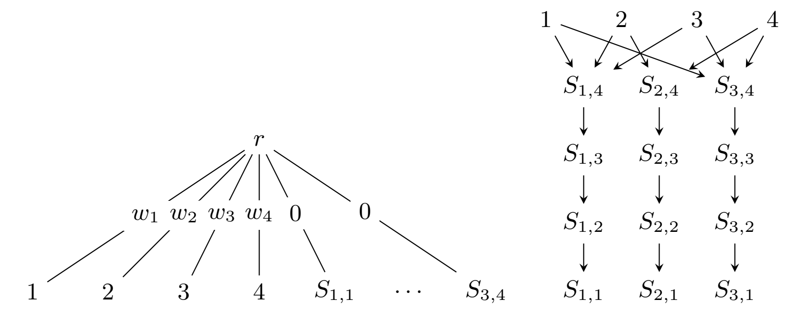
Definition 4. *The input to the* Max Coverage *problem is a set of domain elements* $D = \{1, 2, \ldots, n\}$*, together with non-negative integer weights* $w_1, \ldots w_n$*, a collection of subsets of* D $\mathsf{SC} = \{S_1, \ldots S_m\}$ *and a positive integer* k*. The goal is to find a set* $\mathsf{SC}' \subseteq \mathsf{SC}$ *of cardinality* k *maximizing* $\sum_{i \in \bigcup_{S \in \mathsf{SC}'} S} w_i$.

Proposition 2. *There is no* α*-approximation algorithm for Max Coverage with* $\alpha > 1 - \frac{1}{e}$ *(unless* $\mathsf{P} = \mathsf{NP}$*) [5, 8].*

Reduction 1. *Given an instance* (D, SC, k) *of the Max Coverage problem, we build an instance of OptPDVC as follows (cf. Fig. 2)*

$$\begin{aligned}
X &= D \cup \{S_{i,j} \mid S_i \in \mathsf{SC}, 1 \leq j \leq n\} \\
E &= \{(j, S_{i,n}) \mid j \in S_i\} \cup \{(S_{i,j+1}, S_{i,j}) \mid 1 \leq i \leq m, 1 \leq j < n\} \\
\mathcal{T} &= (\{r\} \cup X, \{(r, s) \mid s \in X\}) \\
w_e &= \begin{cases} w_i & e = (r, i), i \in D \\ 0 & \textit{otherwise} \end{cases} \\
k' &= (k+1) \cdot n
\end{aligned}$$

Lemma 4. *Let* (D, SC, k) *be an instance of the Max Coverage problem and let* $(\mathsf{TC}, (X, E), k')$ *be an instance of OptPDVC given by Reduction 1. Let* $W > 0$*. Then there exists a cover* $C \subseteq \mathsf{SC}$ *of size* k *with* $w(C) \geq W$ *for* (D, SC, k) *iff there exists a viable set* A *of size* $k' = (k+1) \cdot n$ *with* $PD(A) \geq W$.



(a) Phylogenetic Tree $(T, E_{\mathcal{T}})$ with weights w_i (b) Food Web (X, E)

Fig. 2. An illustration of Reduction 1, applied to $D = \{1, 2, 3, 4\}$, $\mathsf{SC} = \{S_1, S_2, S_3\}$, $S_1 = \{1, 2, 3\}$, $S_2 = \{2, 4\}$, $S_3 = \{1, 3, 4\}$.

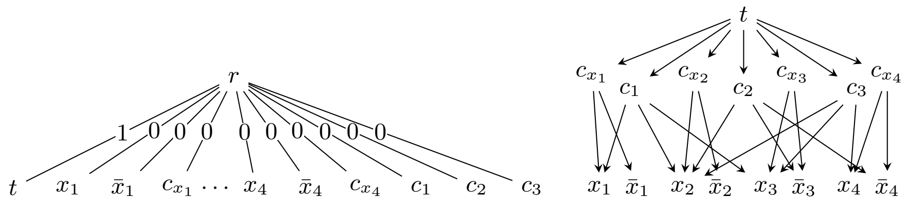
Proof. $\Rightarrow$: First assume that there is a cover C of size k with $w(C) = W$. Then $A' = \{S_{i,j} \mid S_i \in C, 1 \leq j \leq n\} \cup \bigcup_{S_i \in C} S_i$ is a viable set of size $\leq k \cdot n + n$. Clearly $PD(A') = W$ and thus we have a viable set A of size $(k + 1) \cdot n$ with $PD(A) \geq W$ by adding arbitrary viable species.

$\Leftarrow$: Assume there is a viable set A of size $(k+1) \cdot n$ with $PD(A) = W$. There are at most $k + 1$ elements $S_i \in \mathsf{SC}$ such that $S_{i,n} \in A$. This is by the fact that if $S_{i,n} \in A$ then also $S_{i,1}, \ldots, S_{i,n-1} \in A$. Now consider the case where there are exactly $k + 1$ such elements. Then we already have $(k + 1) \cdot n$ species in A and thus no $x \in D$ is contained in A. But then $PD(A) = 0$ as only the edges (r, x) with $x \in D$ have non-zero weight. Assuming $W > 0$ we thus have at most k elements $S_i \in E$ such that $S_{i,n} \in A$ and further as A is viable for each $x \in A$ there is an $S_{i,n} \in A$ such that $x \in S_i$. Hence $C' = \{S_i \mid S_{i,n} \in A\}$ is of size at most k and covers all $x \in A \cap D$, i.e. $w(C') = W$. Now by adding arbitrary $S_i \in \mathsf{SC}$ we can construct a cover C of size k with $w(C) \geq W$. $\square$

Theorem 4. *There is no α-approximation algorithm for OptPDVC with $\alpha > 1 - \frac{1}{e}$ (unless $\mathsf{P} = \mathsf{NP}$), even if $\mathcal{PD}$ is an additive function.*

Proof. Immediate by Proposition 2, Lemma 4 and the fact that Reduction 1 can be performed in polynomial time. $\square$

Finally let us consider a straightforward generalization of viability constraints. So far we assumed that a species is viable iff at least one of its successors survives, but one can also imagine cases where one node needs several or even all of its successors to survive to be viable. In the following we consider food webs where we allow two types of nodes: (i) nodes that are viable if at least one successors survives and (ii) nodes that are only viable if all successors survive. We will show that in this setting no approximation algorithm is possible using a reduction from the NP-hard problem of deciding whether a propositional formula in 3-CNF is satisfiable. A 3-CNF formula is a propositional formula which is the conjunction of clauses, and each clause is the disjunction of exactly three literals, e.g. $\phi = (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee \neg x_4) \wedge (x_2 \vee x_3 \vee x_4)$.



(a) Phylogenetic Tree $(T, E_{\mathcal{T}})$ with weights w_i (b) Food Web (X, E)

Fig. 3. An illustration of Reduction 2, applied to the propositional formula $\phi = (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee \neg x_4) \wedge (x_2 \vee x_3 \vee x_4)$.

Reduction 2. *Given a propositional formula ϕ in 3-CNF over propositional variables $\mathcal{V} = \{x_1, \ldots, x_n\}$ with clauses $c_1, \ldots, c_m$ build the following instance $(T, E_{\mathcal{T}})$, (X, E) and weight w_e (cf. Fig. 3) :*

$$\begin{aligned}
X &= \{c_1, \ldots, c_m\} \cup \{x, \bar{x}, c_x \mid x \in \mathcal{V}\} \cup \{t\} \\
\mathcal{T} &= (\{r\} \cup X, \{(r, s) \mid s \in X\}) \\
w_e &= \begin{cases} 1 & e = (r, t) \\ 0 & \textit{otherwise} \end{cases} \\
E &= \{(c_x, x), (c_x, \bar{x}) \mid x \in \mathcal{V}\} \cup \{(c_i, x) \mid x \in c_i\} \cup \{(c_i, \bar{x}) \mid \neg x \in c_i\} \\
&\quad \cup \{(t, c_i), (t, c_x) \mid 1 \le i \le m, x \in \mathcal{V}\} \\
k &= 2 \cdot |\mathcal{V}| + m + 1
\end{aligned}$$

The species $\{c_1, \ldots c_m\} \cup \{x, \bar{x}, c_x \mid x \in \mathcal{V}\}$ are viable in the traditional sense and t is viable iff all its successors survive. More formally, a set $S \subseteq X$ is viable if (i) for each $s \in S$ either s is a sink or there is a $s' \in S$ with $(s, s') \in E$ and (ii) if $t \in S$ it holds for all s' with $(t, s') \in E$ that $s' \in S$.

Lemma 5. *Given a propositional formula ϕ and the instance $(\mathsf{TC}, (X, E), k)$ of OptPDVC given by Reduction 2. Then ϕ is satisfiable iff there exists a viable set A of size $\le k$ with $PD(A) > 0$.*

Now assuming that there is an approximation algorithm for OptPDVC with generalized viability constraints we would immediately get an procedure deciding 3-CNF formulas: apply Reduction 2, compute $\mathcal{PD}$ using the α-approximation algorithm, and return satisfiable if $\mathcal{PD}$ is positive.

Theorem 5. *It is* NP*-hard to decide whether an instance of OptPDVC with generalized viability constraints has a viable set S with $\mathcal{PD}(S) > 0$. Thus no approximation algorithm for the problem can exist unless* $\mathsf{P} = \mathsf{NP}$.

Proof. Immediate by Lemma 5, and the fact that Reduction 2 can be performed in polynomial time.

References

1. Magnus Bordewich and Charles Semple. Nature reserve selection problem: A tight approximation algorithm. *Computational Biology and Bioinformatics, IEEE/ACM Transactions on*, 5(2):275–280, 2008.
2. Magnus Bordewich and Charles Semple. Budgeted nature reserve selection with diversity feature loss and arbitrary split systems. *Journal of mathematical biology*, 64(1-2):69–85, 2012.
3. Daniel P. Faith. Conservation evaluation and phylogenetic diversity. *Biological Conservation*, 61(1):1–10, 1992.
4. Beáta Faller, Charles Semple, and Dominic Welsh. Optimizing Phylogenetic Diversity with Ecological Constraints. *Annals of Combinatorics*, 15:255–266, 2011.
5. Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998.
6. M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey. An analysis of approximations for maximizing submodular set functions – II. *Mathematical Programming Study*, 8:73–87, 1978.
7. Pranava R. Goundan and Andreas S. Schulz. Revisiting the greedy approach to submodular set function maximization. Working Paper, Massachusetts Institute of Technology, 2007. Available at `http://www.optimization-online.org/DB_HTML/2007/08/1740.html`.
8. Samir Khuller, Anna Moss, and Joseph Naor. The budgeted maximum coverage problem. *Inf. Process. Lett.*, 70(1):39–45, 1999.
9. Jon Lee, Vahab S. Mirrokni, Viswanath Nagarajan, and Maxim Sviridenko. Non-monotone submodular maximization under matroid and knapsack constraints. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 323–332. ACM, 2009.
10. Vincent Moulton, Charles Semple, and Mike Steel. Optimizing phylogenetic diversity under constraints. *Journal of Theoretical Biology*, 246(1):186 – 194, 2007.
11. G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions — I. *Mathematical Programming*, 14:265–294, 1978.
12. Fabio Pardi and Nick Goldman. Species choice for comparative genomics: being greedy works. *PLoS Genetics*, 1(6):e71, 2005.
13. Tsan sheng Hsu, Kuo-Hui Tsai, Da-Wei Wang, and D. T. Lee. Two variations of the minimum steiner problem. *J. Comb. Optim.*, 9(1):101–120, 2005.
14. Mike Steel. Phylogenetic diversity and the greedy algorithm. *Systematic Biology*, 54(4):527–529, 2005.
15. Jan Vondrák. Submodular functions and their applications. SODA 2013 plenary talk. Slides available at `http://theory.stanford.edu/∼jvondrak/data/SODA-plenary-talk.pdf`.
16. Martin L. Weitzman. The Noah's ark problem. *Econometricay*, 66:1279 – 1298, 1998.

Appendix

Lemma 5 Given a propositional formula ϕ and the instance $(\mathsf{TC}, (X, E), k)$ of OptPDVC given by Reduction 2. Then ϕ is satisfiable iff there exists a viable set A of size $\leq k$ with $PD(A) > 0$.

Proof. $\Rightarrow$: Let α be a truth assignment satisfying ϕ, i.e. $\alpha(\phi) = 1$. Then it is easy to verify that $A = \{x \mid x \in \mathcal{V}, \alpha(x) = 1\} \cup \{\bar{x} \mid x \in \mathcal{V}, \alpha(x) = 0\} \cup \{c_1, \dots c_m\} \cup \{c_x \mid x \in \mathcal{V}\} \cup \{t\}$ is a viable set of size $k = 2 \cdot |\mathcal{V}| + m + 1$ with $PD(A) = 1$.

$\Leftarrow$: If there is a viable subset A' with $PD(A') > 0$ there is also a viable set $A \supset A'$ of size $k = 2 \cdot |\mathcal{V}| + m + 1$ and $PD(A) > 0$, because $|X|$ is of size $3 \cdot |\mathcal{V}| + m + 1$. We show that truth-assignment α setting each $s \in A \cap \mathcal{V}$ to 1 and each $s \in \mathcal{V} \backslash A$ to 0 satisfies ϕ. As $PD(A) > 0$ we clearly have that $t \in A$. Now as A is viable and we have an AND constraint on t also $\{c_1, \dots c_m\} \cup \{c_x \mid x \in X\} \subseteq A$. By $c_x \in A$ we obtain that for each $x \in X$ either $x \in A$ or $\bar{x} \in A$, but not both of them (as there are only $|\mathcal{V}|$ species left). Finally as $c_i \in A$ we have that for each clause there is either an $x \in C$ with $\alpha(x) = 1$ or a $\neg x \in C$ with $\alpha(x) = 0$. Thus each clause c_i is satisfied by α, i.e. $\alpha(c_i) = 1$, and hence also $\alpha(\phi) = 1$. $\square$

Proof of Theorem 3

With Algorithm 3 we give a precise formulation of the algorithm in Theorem 3.

Algorithm 3

```
 1: $\mathcal{G} \leftarrow \emptyset$
 2: for each $G \subseteq X$, $G$ viable, $|G| \leq \min(3p + 3d - 3, k)$ do
 3:     $\mathcal{B} \leftarrow \{B \subseteq X \mid |B| \leq p, 1 \leq c(B|G) \leq k - |G|\}$
 4:     while $\mathcal{B} \neq \emptyset$ do
 5:         $S \leftarrow \underset{B \in \mathcal{B}}{\operatorname{argmax}} \frac{\mathcal{PD}(B|G)}{c(B|G)}$
 6:         $G = \mathrm{VE}(G \cup S)$
 7:         $\mathcal{B} \leftarrow \{B \in \mathcal{B} \mid |B| \leq p, 1 \leq c(B|G) \leq k - |G|\}$
 8:     end while
 9:     if $\mathcal{PD}(G) > \mathcal{PD}(\mathcal{G})$ then
10:         $\mathcal{G} \leftarrow G$
11:     end if
12: end for
```

Again let O be the optimal viable set and $\mathcal{D}_O$ a decomposition of O, given by Lemma 1. We consider the set $G_0^* = O_1^* \cup O_2^* \cup O_3^*$ with $\{O_1^*, O_2^*, O_3^*\} \subseteq \mathcal{D}_O$ such that $\{O_1^*, O_2^*\} = \underset{\{O_i, O_j\} \subseteq \mathcal{D}_O}{\operatorname{argmax}} \mathcal{PD}(O_i \cup O_j)$ and $O_3^* = \underset{O_i \in \mathcal{D}_O}{\operatorname{argmax}} \mathcal{PD}(O_1^* \cup O_2^* \cup O_i)$ and the viable extension $G_0 = G_0^* \cup B_1^* \cup B_2^* \cup B_3^*$. At some point Algorithm 2 will consider G_0. We consider this iteration of the for loop in line 2 and use the same notation as in the proof of Theorem 1, the only difference being the definition of the set G_0 above.

Lemma 6. *For $1 \leq i \leq l+1$, $p \in \{1, \dots, \lfloor k/3 \rfloor\}$:*

$$\frac{\mathcal{PD}(S_i|G_{i-1})}{c_i} \geq \frac{p}{(p+d-1)(k-|G_0|)} \cdot \mathcal{PD}(O|G_{i-1})$$

Proof. By the definition of S_i for each $O_j \in \mathcal{D}_O$ the following holds:

$$\frac{\mathcal{PD}(O_j|G_{i-1})}{c(O_j|G_{i-1})} \leq \frac{\mathcal{PD}(S_i|G_{i-1})}{c(G_i|G_{i-1})}$$

Next we use the monotonicity and submodularity of $\mathcal{PD}$ (for the first inequality), the inequality from above (for the second inequality). We use $\mathcal{D}'_O$ to denote $\mathcal{D}_O \setminus \{O_j \subseteq G_{i-1}\}$.

$$\begin{aligned}
\mathcal{PD}(O|G_{i-1}) &\leq \sum_{O_j \in \mathcal{D}'_O} \mathcal{PD}(O_j|G_{i-1}) = \sum_{O_j \in \mathcal{D}'_O} \frac{\mathcal{PD}(O_j|G_{i-1})}{c(O_j|G_{i-1})} \, c(O_j|G_{i-1}) \\
&\leq \sum_{O_j \in \mathcal{D}'_O} \frac{\mathcal{PD}(S_i|G_{i-1})}{c_i} \, c(O_j|G_{i-1}) = \frac{\mathcal{PD}(S_i|G_{i-1})}{c_i} \sum_{O_j \in \mathcal{D}'_O} c(O_j|G_{i-1}) \\
&\leq \frac{\mathcal{PD}(S_i|G_{i-1})}{c_i} \, \frac{p+d-1}{p} \cdot (k-|G_0|)
\end{aligned}$$

The last step exploits that by Lemma 1 $\sum_{O_j \in \mathcal{D}'_O} c(O_j|G_{i-1}) \leq \frac{k-|G_0|}{p} \cdot (p+d-1)$. □

Lemma 7. *For $1 \leq i \leq l+1$:*

$$\mathcal{PD}(G^*_i|G_0) \geq \left[1 - \prod_{j=1}^{i} \left(1 - \frac{p \cdot c_j}{(d+p-1) \cdot (k-|G_0|)}\right)\right] \mathcal{PD}(O|G_0)$$

Proof. The proof is by induction on i. The base case for $i = 1$ is by Lemma 6. For the induction step we show that if the claim holds for all $i' < i$ then it must also hold for i. For convenience we define $C_i = \frac{p \cdot c_i}{(d+p-1) \cdot (k-|G_0|)}$.

$$\begin{aligned}
\mathcal{PD}(G^*_i|G_0) &= \mathcal{PD}(G_{i-1}|G_0) + \mathcal{PD}(G^*_i|G_{i-1}) \\
&\geq \mathcal{PD}(G_{i-1}|G_0) + C_i \cdot (\mathcal{PD}(O|G_{i-1})) \\
&= \mathcal{PD}(G_{i-1}|G_0) + C_i \cdot (\mathcal{PD}(O \cup G_{i-1}) - \mathcal{PD}(G_0) - (\mathcal{PD}(G_{i-1}) - \mathcal{PD}(G_0))) \\
&\geq (1 - C_i) \cdot \mathcal{PD}(G_{i-1}|G_0) + C_i \cdot \mathcal{PD}(O|G_0) \\
&\geq (1 - C_i) \cdot \mathcal{PD}(G^*_{i-1}|G_0) + C_i \cdot \mathcal{PD}(O|G_0) \\
&\geq (1 - C_i) \left[1 - \prod_{j=1}^{i-1} (1 - C_j)\right] \mathcal{PD}(O|G_0) + C_i \cdot \mathcal{PD}(O|G_0) \\
&= \left[1 - \prod_{j=1}^{i} (1 - C_j)\right] \mathcal{PD}(O|G_0)
\end{aligned}$$

□

Proof (Theorem 3 approximation ratio). We first give a bound for G_{l+1}. To this end consider $\sum_{m=1}^{l+1} c_m$. As G_{l+1} exceeds the cardinality constraint $\sum_{m=1}^{l+1} c_m > k - |G_0|$ and hence:

$$\begin{aligned} 1 - \prod_{j=1}^{l+1} \left(1 - \frac{p \cdot c_j}{(d+p-1) \cdot (k - |G_0|)}\right) &\geq 1 - \prod_{j=1}^{l+1} \left(1 - \frac{p \cdot c_j}{(d+p-1) \cdot \sum_{m=1}^{l+1} c_m}\right) \\ \geq 1 - \left(1 - \frac{p}{(d+p-1) \cdot (l+1)}\right)^{l+1} &\geq 1 - \frac{1}{e^{p/(d+p-1)}} \end{aligned}$$

To obtain Line 2 we used the fact the term $1 - \prod_{j=1}^{l+1} \left(1 - \frac{c'_j}{C}\right)$ with constant C and the constraint $\sum_{j=1}^{l+1} c'_j = 1$ has its maximum at $c'_j = 1/(l+1)$.

By Lemma 7 we obtain $\mathcal{PD}(G^*_{l+1}|G_0) \geq \left(1 - \frac{1}{e^{p/(d+p-1)}}\right) \mathcal{PD}(O|G_0)$, thus it only remains to relate G^*_{l+1} to G_l. First as $G_0 = O^*_1 \cup O^*_2 \cup O^*_3$ and by the definition of O^*_1, O^*_2 we get $\mathcal{PD}(O^*_3|O^*_1 \cup O^*_2) \leq \mathcal{PD}(G_0)/3$. Next consider

$$\begin{aligned} \mathcal{PD}(G^*_{l+1}) - \mathcal{PD}(G_l) = \mathcal{PD}(O'_{l+1}|G_l) &\leq \mathcal{PD}(O'_{l+1}|O^*_1 \cup O^*_2) \\ &\leq \mathcal{PD}(O^*_3|O^*_1 \cup O^*_2) \leq \mathcal{PD}(G_0)/3 \end{aligned}$$

Finally, we can combine our results to obtain the claim:

$$\begin{aligned} \mathcal{PD}(G) &\geq \mathcal{PD}(G_l) \geq \mathcal{PD}(G^*_{l+1}) - \mathcal{PD}(G_0)/3 \\ &= \mathcal{PD}(G^*_{l+1}|G_0) + \mathcal{PD}(G_0) - \mathcal{PD}(G_0)/3 \\ &\geq \left(1 - \frac{1}{e^{p/(p+d-1)}}\right) (\mathcal{PD}(O) - \mathcal{PD}(G_0)) + 2/3 \cdot \mathcal{PD}(G_0) \\ &\geq \left(1 - \frac{1}{e^{p/(p+d-1)}}\right) \cdot \mathcal{PD}(O) \end{aligned}$$

The last in equality is by the fact that $1 - \frac{1}{e^{p/(p+d-1)}} \leq 2/3$ for all $p, d \geq 1$. □

Proof (Theorem 3 - running time). As mentioned before computing the function VE is basically a Steiner tree problem and can be solved in time $\tilde{\mathcal{O}}(3^j n^2 + nm)$, where j is the number of starting nodes. In the algorithm we have to consider $\mathcal{O}(n^{3p+3d-3})$ sets S and for each of them we start the greedy algorithm. The number of iterations of the while loop is bounded by k and in each iteration, in Line 6, we have to solve $\mathcal{O}(n^p)$ Steiner tree problems with at most p starting nodes. As each iteration takes time $\mathcal{O}(3^p n^{p+2} + n^{p+1} m)$ we get a running time of $\mathcal{O}\left(n^{3p+3d-3} \cdot k \cdot (3^p n^{p+2} + n^{p+1} m)\right) = \mathcal{O}\left(k \cdot (3^p n^{4p+3d-1} + n^{4p+3d-2} m)\right)$. □