Comparing Apples and Oranges: Query Tradeoff in Submodular Maximization

Niv Buchbinder* Moran Feldman† Roy Schwartz‡

October 6, 2014

Abstract

Fast algorithms for submodular maximization problems have a vast potential use in applicative settings, such as machine learning, social networks, and economics. Though fast algorithms were known for some special cases, only recently Badanidiyuru and Vondrák [4] were the first to explicitly look for such algorithms in the general case of maximizing a monotone submodular function subject to a matroid independence constraint. The algorithm of Badanidiyuru and Vondrák matches the best possible approximation guarantee, while trying to reduce the number of value oracle queries the algorithm performs.

Our main result is a new algorithm for this general case which establishes a surprising *tradeoff* between two seemingly unrelated quantities: the number of value oracle queries and the number of matroid independence queries performed by the algorithm. Specifically, one can decrease the former by increasing the latter and vice versa, while maintaining the best possible approximation guarantee. Such a tradeoff is very useful since various applications might incur significantly different costs in querying the value and matroid independence oracles. Furthermore, in case the rank of the matroid is $O(n^c)$, where n is the size of the ground set and c is an absolute constant smaller than 1, the total number of oracle queries our algorithm uses can be made to have a smaller magnitude compared to that needed by [4]. We also provide even faster algorithms for the well studied special cases of a cardinality constraint and a partition matroid independence constraint, both of which capture many real-world applications and have been widely studied both theorically and in practice.

*Statistics and Operations Research Dept., Tel Aviv University, Israel. E-mail: niv.buchbinder@gmail.com. Research supported by ISF grant 954/11 and BSF grant 2010426.

†School of Computer and Communication Sciences, EPFL, Switzerland. E-mail: moran.feldman@epfl.ch. Research supported in part by ERC Starting Grant 335288-OptApprox.

‡Dept. of Computer Science, Princeton University, Princeton, NJ. E-mail: roysch@cs.princeton.edu.

1 Introduction

The study of combinatorial optimization problems with a submodular objective has attracted much attention in recent years, as submodular functions arise naturally in various disciplines, *e.g.*, combinatorics, economics, and machine learning. Many well-known problems in combinatorial optimization are in fact submodular maximization problems, including: `Max Cut` [23, 26, 28, 30, 40], `Max DiCut` [16, 23, 24], `Generalized Assignment` [9, 11, 18, 20], `Max` k`-Coverage` [15, 31], `Max Bisection` [2, 21], and `Max Facility Location` [1, 12, 13]. Furthermore, practical applications of submodular maximization problems are common in social networks [25, 29], vision [5, 27], machine learning [32, 33, 34, 35, 36] (the reader is referred to a comprehensive survey by Bach [3]), and many other areas. Elegant algorithmic techniques were developed in the course of this line of research which achieved provable, and in some cases even tight, approximation guarantees. A prime example for the latter is the *continuous greedy* algorithm of [8] for maximizing a monotone submodular function subject to a matroid independence constraint. Unfortunately, most of these techniques result in algorithms which are efficient in theory but are not practical. Hence, a natural research question is whether one can obtain *faster* algorithms with provable tight guarantees for basic submodular optimization problems.

How does one measure the speed of an algorithm for a submodular maximization problem? Since an explicit representation of the submodular function might be exponential in the size of its ground set, the algorithm is assumed to access the objective function f via a *value oracle*[1] which returns the value of $f(S)$ given any subset S of the ground set. Usually, the number of value oracle queries dominates the number of arithmetic operations in the algorithm up to a polylogarithmic factor. Hence, it is natural to use the number of value oracle queries as a measure for the algorithm's speed. The use of this measure is also facilitated by the observation that implementions of the value oracle have a non-neglagible time complexity in many applications.

Badanidiyuru and Vondrák [4] were the first to consider the question of finding fast algorithms with provable guarantees for maximizing a submodular function in its full generality. They presented algorithms that achieve an almost tight approximation guarantee of $1 - 1/e - \varepsilon$, for any $\varepsilon > 0$, for both the cardinality and the more general matroid independence constraints. The algorithms they designed use $O\left(\frac{n}{\varepsilon}\log\left(\frac{n}{\varepsilon}\right)\right)$ value oracle queries for the cardinality constraint and $O\left(\frac{nk}{\varepsilon^4}\log^2\left(\frac{n}{\varepsilon}\right)\right)$ value oracle queries for a general matroid independence constraint. Here k denotes the rank of the matroid and n is the size of the ground set.

In the context of a simple constraint such as a cardinality constraint, it is easy to determine whether a given solution S is feasible. However, when considering more complex constraints, such as a general matroid independence constraint, one usually assumes the existence of an *independence oracle*. This oracle determines whether a given subset S of elements of the ground set is independent in the matroid, *i.e.*, feasible. In all previous works, as far as we know, the number of value oracle queries dominates the number of independence oracle queries, and thus, the latter is usually overlooked. This overlook is unfortunate since the implementation of these two *distinct* oracles in various applications might have running times of completely different magnitudes.[2] In particular, minimizing the number of value oracle queries, as implicitly done by previous works, might not be the correct goal. Furthermore, it is not clear whether the two different goals mentioned above, *i.e.*, minimizing the number of value oracle queries and minimizing the number of independence oracle queries, are related.

[1] Other types of oracles exist, however, value oracles are the most commonly used type in the literature.

[2] For example, the independence oracle can be implemented very efficiently when the constraint is a uniform or partition matroid. However, no linear time implementation is known when the constraint is a matching or linear matroid.

1.1 Our Results

Our main result is the design of an algorithm for maximizing a monotone submodular function subject to a matroid independence constraint. Our algorithm establishes a *tradeoff* between the following two seemingly unrelated quantities: the number of value oracle queries and the number of independence oracle queries performed by the algorithm. The following theorem summarizes the result.

Theorem 1.1. *There exists an algorithm that given a non-negative monotone submodular function $f : 2^{\mathcal{N}} \to \mathbb{R}^+$, a matroid $\mathcal{M} = (\mathcal{N}, \mathcal{I})$ of rank k, and parameters $\varepsilon > 0$ and $\lambda \in [1, k]$, finds a solution $S \in \mathcal{I}$ where:*

1. *$f(S) \geq (1 - 1/e - \varepsilon) \cdot \max\{f(T) : T \in \mathcal{I}\}$.*
2. *The algorithm performs $O\left(k\lambda + \frac{kn}{\lambda\varepsilon^5} \ln^2\left(\frac{n}{\varepsilon}\right)\right)$ value oracle queries.*
3. *The algorithm performs $O\left(\frac{k^2}{\varepsilon} + \frac{\lambda n}{\varepsilon^2} \ln\left(\frac{n}{\varepsilon}\right)\right)$ independence oracle queries.*

An example, in which the tradeoff between the number of the two distinct oracle queries is perhaps most insightful, is when $k = \Theta(\sqrt{n})$. In this case, if one chooses $\lambda = 1$ our algorithm performs $\tilde{O}_\varepsilon\left(n^{3/2}\right)$ value queries, but only $\tilde{O}_\varepsilon(n)$ independence oracle queries.[3] However, if one chooses $\lambda = k$ the algorithm performs only $\tilde{O}_\varepsilon(n)$ value queries while the number of independence oracle queries grows to $\tilde{O}_\varepsilon\left(n^{3/2}\right)$. This allows flexibility when the two types of queries have different time complexities in the application at hand. Note that in this case when $k = \Theta(\sqrt{n})$ the algorithm of [4] corresponds to choosing $\lambda = 1$ in our algorithm, as it performs $\tilde{O}_\varepsilon\left(n^{3/2}\right)$ value queries and $\tilde{O}_\varepsilon(n)$ independence oracle queries.

It is important to note that our algorithm not only establishes a surprising tradeoff between the two different types of queries, but can also provide a significant speedup to the running time when compared to the state of the art algorithm of [4]. Consider, for simplicity, the case where both types of oracles have the same running time. In this case, obtaining a fast algorithm requires reducing the *total* number of oracle queries regardless of their types. While the algorithm of [4] requires $\tilde{O}_\varepsilon\left(k^2 + nk\right)$ queries, our algorithm requires only $\tilde{O}_\varepsilon(k^2 + \sqrt{k}n)$ queries if one sets $\lambda = \sqrt{k}$. In the above example, where $k = \Theta(\sqrt{n})$, it reduces the number of oracle queries from $\tilde{O}_\varepsilon\left(n^{3/2}\right)$ to $\tilde{O}_\varepsilon\left(n^{5/4}\right)$. In fact, if $k = O(n^c)$ for some absolute constant $c \leq 2/3$, λ can be chosen such that the our algorithm uses $\tilde{O}_\varepsilon\left(n^{1+c/2}\right)$ oracle queries, whereas [4]'s algorithm needs $\tilde{O}_\varepsilon\left(n^{1+c}\right)$ oracle queries (for $2/3 < c < 1$ we still get an improvement, but a smaller one).

Additional Results. We consider two well-studied special cases of the general problem considered above. The first is the case of a cardinality constraint and the second is the case of a partition matroid independence constraint. Building upon the ideas developed in the context of the main result, we present even faster algorithms for the above two special cases. The reader should note that in these two cases the implementation of the independence oracle is trivial, and hence, we focus only on minimizing the number of value oracle queries. The following three theorems summarize these results.

Theorem 1.2. *There exists an algorithm that given a non-negative monotone submodular function $f : 2^{\mathcal{N}} \to \mathbb{R}^+$, a generalized partition matroid $\mathcal{M} = (\mathcal{N}, \mathcal{I})$ of rank k, and a parameter $\varepsilon > 0$, finds a solution $S \in \mathcal{I}$ where: $f(S) \geq (1 - 1/e - \varepsilon) \cdot \max\{f(T) : T \in \mathcal{I}\}$ and the algorithm performs $O\left(k\sqrt{\frac{n}{\varepsilon^5}} \ln\left(\frac{n}{\varepsilon}\right) + \frac{n}{\varepsilon^5} \ln^2\left(\frac{n}{\varepsilon}\right)\right)$ value oracle queries.*

[3] Here $\tilde{O}_\varepsilon$ hides polylogarithmic factors in n and polynomial factors in ε^{-1}.

Theorem 1.3. *There exists an algorithm that given a non-negative monotone submodular function $f : 2^{\mathcal{N}} \to \mathbb{R}^+$, and parameters $k \geq 1$ and $\varepsilon > 0$, finds a solution $S \subseteq \mathcal{N}$ of size $|S| \leq k$ where: $f(S) \geq (1 - 1/e - \varepsilon) \cdot \max\{f(T) : T \subseteq \mathcal{N}, |T| \leq k\}$ and the algorithm performs $O\left(n \ln\left(\frac{1}{\varepsilon}\right)\right)$ value oracle queries.*

Theorem 1.4. *There exists an algorithm that given a general non-negative submodular function $f : 2^{\mathcal{N}} \to \mathbb{R}^+$, and parameters $k \geq 1$ and $\varepsilon > 0$, finds a solution $S \subseteq \mathcal{N}$ of size $|S| \leq k$ where: $f(S) \geq (1/e - \varepsilon) \cdot \max\{f(T) : T \subseteq \mathcal{N}, |T| \leq k\}$ and the number of value oracle queries the algorithm performs is $\min\left\{O\left(\frac{n}{\varepsilon^2} \ln\left(\frac{1}{\varepsilon}\right)\right), O\left(k\sqrt{\frac{n}{\varepsilon} \ln\left(\frac{k}{\varepsilon}\right)} + \frac{n}{\varepsilon} \ln\left(\frac{k}{\varepsilon}\right)\right)\right\}$.*

The best previously known result for partition matroids is identical to the one that was known for general matroids, *i.e.*, it uses $O\left(\frac{nk}{\varepsilon^4} \log^2\left(\frac{n}{\varepsilon}\right)\right)$ value oracle queries. We note that Theorem 1.3 is a folklore result that improves over the best previously formally published result of [4], who described an algorithm using $O(\frac{n}{\varepsilon} \ln \frac{n}{\varepsilon})$ value oracle queries.

1.2 Additional Related Work

The literature on submodular maximization is rich and has a long history. We mention here only a few of the most relevant works. The classical result of Nemhauser et al. [38] states that the simple discrete greedy algorithm provides an approximation of $(1 - 1/e)$ for maximizing a monotone submodular function subject to a cardinality constraint. This result is known to be tight by the work of Nemhauser et al. [37]. Feige [15] proved the latter holds even when the objective function is restricted to being a coverage function. Calinescu et al. [8] presented the continuous greedy algorithm, which enabled one to achieve the same tight $(1 - 1/e)$ guarantee for the more general matroid constraint.

However, when one considers submodular objectives which are not monotone, less is known. An approximation of 0.309 was given by Vondrák [41] for the general matroid independence constraint, which was later improved to 0.325 by Oveis Gharan and Vondrák [22] using a simulated annealing technique. Extending the continuous greedy algorithm of [8] to general non-negative submodular objectives, Feldman et al. [19] obtained an improved approximation of $1/e - o(1)$ for the same problem.

When considering the special case of a cardinality constraint and a submodular objective which is not necessarily monotone, Buchbinder et al. [7] presented a $1/e$-approximation algorithm, called "random greedy" whose running time is as fast as the discrete greedy algorithm of Nemhauser et al. [38]. Furthermore, [7] also described a slower polynomial time $(1/e + 0.004)$-approximation algorithm, demonstrating that $1/e$ is not the right approximation ratio for the problem. On the hardness side, it is known that no polynomial time algorithm can have an approximation ratio better than 0.491 [22].

Paper Organization. Section 2 gives general preliminaries. Section 3 describes our results for general and partition matroids (Theorems 1.1 and 1.2). Finally, Sections 4 and 5 prove Theorem 1.4. The proof of the folklore result given by Theorem 1.3 can be found in Appendix A.

2 Preliminaries

Given a non-negative submodular function $f : 2^{\mathcal{N}} \to \mathbb{R}^+$, a set $S \subseteq \mathcal{N}$ and an element $u \in \mathcal{N}$, we denote by $f(u \mid S) = f(S \cup \{u\}) - f(S)$ the marginal contribution of u to S. The following similar lemmata of [17] and [7] are used in many of our proofs.

Lemma 2.1 (Lemma 2.2 of [17]). *Let $f : 2^{\mathcal{N}} \to \mathbb{R}$ be submodular. Denote by $A(p)$ a random subset of A where each element appears with probability p (not necessarily independently). Then, $\mathbb{E}[f(A(p))] \geq (1-p)f(\varnothing) + p \cdot f(A)$.*

Lemma 2.2 (Lemma 2.2 of [7]). *Let $f : 2^{\mathcal{N}} \to \mathbb{R}^+$ be non-negative and submodular. Denote by $A(p)$ a random subset of A where each element appears with probability at most p (not necessarily independently). Then, $E[f(A(p))] \geq (1-p)f(\varnothing)$.*

3 General Matroid Constraint

In this section we describe algorithms for the problem $\max\{f(S) : S \in \mathcal{I}\}$, where $f : 2^{\mathcal{N}} \to \mathbb{R}^+$ is a non-negative monotone submodular function and $M = (\mathcal{N}, \mathcal{I})$ is a matroid. Throughout the section we use k to denote the rank of M and assume $f(u) \leq f(OPT)$ for every $u \in \mathcal{N}$. The last assumption can be justified by observing that every element having $f(u) > f(OPT)$ must be a self-loop, and thus, all such elements can be removed from $\mathcal{M}$ in linear time.

3.1 Problem Specific Preliminaries

Given a non-negative submodular function $f : 2^{\mathcal{N}} \to \mathbb{R}^+$, its *multilinear* extension is a function $F : [0,1]^{\mathcal{N}} \to \mathbb{R}^+$ defined by $F(x) = \mathbb{E}[f(\mathtt{R}(x))]$, where $\mathtt{R}(x)$ is a random set containing every element $u \in \mathcal{N}$ with probability x_u, independently. We denote by $\partial_u F(x)$ the derivative of F at point x with respect to the coordinate corresponding to u. The multilinear extension has been extensively used for maximizing submodular functions subject to a matroid constrained, starting with the continuous greedy algorithm of Calinescu et al. [8]. All algorithms based on the multilinear extension approximate its value at various points using sampling. Unfortunately, this sampling is often responsible for the quite poor time complexity of these algorithms. For this reason, we use a more cautious approach to sampling in this work.

Let $b : \mathcal{N} \to \mathbb{R}^+$ be a non-negative function such that $\sum_{u \in S} b(u) \leq f(OPT)$ for every independent set $S \in \mathcal{I}$. Let m_b be a number of samples which is sufficent to approximate $\partial_u F(x)$ up to a multiplicative error of δ and an additive error of $\delta \cdot b(u)$ with high probability[4], for every given choice of $u \in \mathcal{N}$ and $x \in [0,1]^{\mathcal{N}}$. Badanidiyuru and Vondrák [4] describe a version of the continuous greedy algorithm which, together with swap rounding [10], provides an approximation ratio of $(1 - e^{-1} - \delta)$ using $O(m_b n \delta^{-2} \ln(\frac{n}{\delta}))$ value oracle queries and $O(n\delta^{-2}\ln(\frac{n}{\delta}) + k^2\delta^{-1})$ independence oracle queries. Badanidiyuru and Vondrák [4] assume $b(u) = f(OPT)/k$ for every $u \in \mathcal{N}$,[5] and bound m_b using the following lemma.

Lemma 3.1 (Lemma 2.3 of [4]). *Let $X_1, X_2, \ldots, X_m$ be independent random variables such that for each $1 \leq i \leq m$, $X_i \in [0,1]$. Let $X = \frac{1}{m} \cdot \sum_{i=1}^{m} X_i$ and $\mu = E[X]$. Then*

$$Pr[X > (1+\alpha)\mu + \beta] \leq e^{-m\alpha\beta/3} \ ,$$
$$Pr[X < (1-\alpha)\mu - \beta] \leq e^{-m\alpha\beta/2} \ .$$

The assumption $f(u) \leq f(OPT)$ for every $u \in \mathcal{N}$ allowed [4] to prove, using the above lemma, that m_b can be set to $k \ln n/\delta^2$. Assume now $\sum_{u \in S} f(u) \leq c \cdot f(OPT)$ for every independent set S and some value c, and let us define $b(u) = f(u)/c$. Clearly, b obeys the required condition.

[4] By "high probability" we mean that the complementary event occurs with a polynomially small probability in n.

[5] In fact, [4] proves explicitly only this case, but the proof can be easily extended to every function b obeying the condition defined above.

Moreover, the above lemma can now be used to show that m_b can be set to $c \ln n/\delta^2$. Thus, we get the following corollary.

Corollary 3.2. *If* $\max_{S\in\mathcal{I}} \sum_{u\in S} f(u) \leq c \cdot f(OPT)$ *for some value* c*, then for every* $\delta > 0$ *there exists a* $(1 - e^{-1} - \delta)$*-approximation algorithm for* $\max\{f(S) \mid S \in \mathcal{I}\}$ *using* $O(cn\delta^{-4} \ln^2(\frac{n}{\delta}))$ *value oracle queries and* $O(n\delta^{-2} \ln(\frac{n}{\delta}) + k^2\delta^{-1})$ *independence oracle queries.*

Some of the algorithms we describe need access to a quick constant approximation of $f(OPT)$. The following lemma provides such an approximation.

Lemma 3.3. *There exists a* $(1/3)$*-approximation algorithm for* $\max\{f(S) \mid S \in \mathcal{I}\}$ *using* $O(n \ln k)$ *value and independence oracle queries.*

The algorithm described by the above lemma is strongly based on the thresholding algorithm of [4], and thus, we defer the proof of the lemma to Appendix B.

3.2 Intuition and Techniques

Corollary 3.2 tells us that there exists a fast algorithm for the problem $\{f(S) \mid S \in \mathcal{I}\}$ when $\max_{S\in\mathcal{I}} \sum_{u\in S} f(u)$ is not much larger than $f(OPT)$. Thus, we need to show how to deal with the case of large $\max_{S\in\mathcal{I}} \sum_{u\in S} f(u)$. One interesting candidate algorithm for this case is the residual random greedy algorithm suggests by [7]. This algorithm works in k iterations. In each iteration, given that S is the current solution of the algorithm, it finds finds a set S' maximizing $\{f(S') \mid S \cup S' \in \mathcal{I}\}$. Then, it selects a random element $u \in S'$, and adds it to its solution S.

Buchbinder et al. [7] only managed to show that their residual random greedy is a 1/4-approximation algorithm. However, it is not difficult to check that this algorithm behaves much better as long as $\max_{S\cup S'\in\mathcal{I}} \sum_{u\in S'} f(u \mid S)$ is large. More specifically, the expected increase in the value of S is large compared to the expected decrease in the value of the best independent set containing S. This suggests the following natural approach. Execute the residual random greedy as long as $\max_{S\cup S'\in\mathcal{I}} \sum_{u\in S'} f(u \mid S)$ is large. Once $\max_{S\cup S'\in\mathcal{I}} \sum_{u\in S'} f(u \mid S)$ becomes small, apply the measured continuous greedy of [4] to the residual problem.

For technical reasons, we also use the observation that the solution produced by the residual random greedy tends to be very small because the value of S increases fast, in expectation. This observation allows us to ignore ("fail") cases in which the goal of small $\max_{S\cup S'\in\mathcal{I}} \sum_{u\in S'} f(u \mid S)$ is not obtained quickly enough.

3.3 Main Algorithm

In this section we explain how to combine our variant of the residual random greedy and the measured continous greedy of [4] into an algorithm for $\max\{f(S) \mid S \in \mathcal{I}\}$ having all the properties guaranteed by Theorem 1.1. The following lemma states the properties of our variant of the residual continous greedy that we need. In Section 3.4, we describe this variant (which appears as Algorithm 2) and prove Lemma 3.4. In the following, we use $\mathcal{M}/S$ to denote the matroid obtained from $\mathcal{M}$ by contracting a set $S \subseteq \mathcal{N}$.

Lemma 3.4. *There exists an algorithm that given a non-negative monotone submodular function* $f : 2^{\mathcal{N}} \to \mathbb{R}^+$*, a matroid* $\mathcal{M} = (\mathcal{N}, \mathcal{I})$ *of rank* k *and three parameters* $\delta \in (0, 1)$*,* $B \geq 0$ *and an integer* $0 \leq I \leq k/2$*, has the following properties.*

(i) The algorithm uses $O(In\delta^{-1} \ln(k/\delta))$ *independence oracle queries and* $O(Ik + n\delta^{-1} \ln(k/\delta))$ *value oracle queries.*

(ii) The algorithm declares failure with probability at most $k(BI)^{-1}$.

(iii) If the algorithm does not fail, it outputs a set S obeying:

- *For every independent set S' of $\mathcal{M}/S$, $\sum_{u\in S'} f(u \mid S) \leq (1-\delta)^{-2}(3B+\delta)\cdot f(OPT)$.*
- *Let OPT' be an independent set of $\mathcal{M}/S$ maximizing $f(OPT')$. Then, $\mathbb{E}[f(OPT')] \geq [1-B^{-1}(2+k/I)]\cdot f(OPT)$, where the expectation is conditioned on the event that the algorithm does not fail.*

Algorithm 1 is our final algorithm for $\max\{f(S) \mid S \in \mathcal{I}\}$. The algorithm gets two parameters: $\varepsilon > 0$ and $\lambda \in [1,k]$. The last parameter controls a tradeoff between the number of value and independence oracle queries used by the algorithm. In the rest of this section we show that Algorithm 1 has all the properties required by Theorem 1.1.

Algorithm 1: Combined Algorithm$(f, \mathcal{M}, \varepsilon, \lambda)$

```
Call the algorithm guaranteed by Lemma 3.4 with the parameters δ = 1/2, B = 20kλ^{-1}ε^{-1}
  and I = ⌈λ/3⌉). Let S denote the output set.
if the algorithm of Lemma 3.4 did not declare failure then
    Call the continuous greedy algorithm guaranteed by Corollary 3.2 on the matroid M/S
      and the objective f(· | S) with the parameters c = 240kλ^{-1}ε^{-1} + 2 and δ = ε/4. Let S'
      denote the output set.
    return S ∪ S'.
else
    return ∅.
```

Remark: For $k = 1$, the problem $\max\{f(S) \mid S \in \mathcal{I}\}$ can be solved optimally using $O(n)$ oracle queries. Thus, we assume throughout this section $k \geq 2$. Notice that this assumption implies $I \leq k/2$, as required by Lemma 3.4. Additionally, Theorem 1.1 is void for $\varepsilon \geq 1 - e^{-1}$, thus, we also assume $\varepsilon \in (0, 1-e^{-1})$.

We begin the analysis of Algorithm 1 by bounding the number of oracle queries it uses.

Observation 3.5. *Algorithm 1 uses at most $O(k\lambda + kn\lambda^{-1}\varepsilon^{-5}\ln^2(\frac{n}{\varepsilon}))$ value oracle queries and $O(k^2\varepsilon^{-1} + \lambda n\varepsilon^{-2}\ln(\frac{n}{\varepsilon}))$ independence oracle queries.*

Proof. The observation follows by adding up the guarantees on the number of oracle queries given by Corollary 3.2 and Lemma 3.4. □

Next, let us lower bound the approximation ratio of Algorithm 1. Let G be the event that the algorithm guaranteed by Lemma 3.4 did not declare failure.

Lemma 3.6. *Conditioned on G, Algorithm 1 is $(1-e^{-1}-\varepsilon/2)$-competitive.*

Proof. By Lemma 3.4, conditioned on G, there exists a random set OPT' (depending on S only) which is always independent in $\mathcal{M}/S$ and obeys:

$$\begin{aligned}\mathbb{E}[f(OPT') \mid G] &\geq \left(1 - \frac{2+k/I}{B}\right)\cdot f(OPT)\\ &\geq \left(1 - \frac{2+3k\lambda^{-1}}{20k\lambda^{-1}\varepsilon^{-1}}\right)\cdot f(OPT) \geq \left(1-\frac{\varepsilon}{4}\right)\cdot f(OPT)\ .\end{aligned}$$

Moreover, Lemma 3.4 also guarantees that for every independent set S' of $\mathcal{M}/S$:

$$\sum_{u \in S'} f(u \,|S) \leq 4(3B + 0.5) \cdot f(OPT) = (240k\lambda^{-1}\varepsilon^{-1} + 2) \cdot f(OPT) = c \cdot f(OPT) \ .$$

Hence, by Corollary 3.2, given a set S, the expected quality of the set produced by the continuous greedy algorithm is at least:

$$\mathbb{E}[f(S' \mid S)] \geq \left(1 - \frac{1}{e} - \frac{\varepsilon}{4}\right) \cdot f(OPT' \mid S) \geq \left(1 - \frac{1}{e} - \frac{\varepsilon}{4}\right) \cdot f(OPT') - f(S) \ .$$

Taking now the expectation over all the sets S, we get:

$$\begin{aligned}
\mathbb{E}[f(S \cup S') \mid G] &= \mathbb{E}[f(S' \mid S) \mid G] + \mathbb{E}[f(S) \mid G] \\
&\geq \mathbb{E}\left[\left(1 - \frac{1}{e} - \frac{\varepsilon}{4}\right) \cdot f(OPT') - f(S) \,\middle|\, G\right] + \mathbb{E}[f(S) \mid G] \\
&= \left(1 - \frac{1}{e} - \frac{\varepsilon}{4}\right) \cdot \mathbb{E}[f(OPT') \mid G] \geq \left(1 - \frac{1}{e} - \frac{\varepsilon}{2}\right) \cdot f(OPT) \ . \qquad \square
\end{aligned}$$

Corollary 3.7. *Algorithm 1 is a $(1 - e^{-1} - \varepsilon)$-approximation algorithm.*

Proof. Let A denote the output of Algorithm 1. Lemma 3.6 states that:

$$\mathbb{E}[f(A) \mid G] \geq \left(1 - \frac{1}{e} - \frac{\varepsilon}{2}\right) \cdot f(OPT) \ .$$

On the other hand, by Lemma 3.4, it is possible to lower bound $\Pr[G]$ by:

$$\Pr[G] \geq 1 - \frac{k}{BI} \geq 1 - \frac{k}{[20k\lambda^{-1}\varepsilon^{-1}] \cdot [\lambda/3]} \geq 1 - \frac{3\varepsilon}{20} \ .$$

The corollary now follows by observing that:

$$\mathbb{E}[f(A)] \geq \Pr[G] \cdot \mathbb{E}[f(A) \mid G] \geq \left(1 - \frac{3\varepsilon}{20}\right) \cdot \left(1 - \frac{1}{e} - \frac{\varepsilon}{2}\right) \cdot f(OPT) \geq \left(1 - \frac{1}{e} - \varepsilon\right) \cdot f(OPT) \ . \square$$

The above corollary completes the proof of Theorem 1.1.

3.4 A variant of the residual random greedy algorithm

In this section we describe an algorithm proving Lemma 3.4. The algorithm we describe is related to the residual random greedy algorithm of [7], with two main modifications. First, the algorithm is sped up using ideas from [4]. Second, the algorithm stops when the total marginal value of all the elements in every independent set becomes small enough. Recall that the last property implies that the measured continuous greedy of [4] can be used efficiently to complete the solution.

The description of our algorithm (give as Algorithm 2) assumes $\mathcal{N}$ contains a known set D of k dummy elements having the following properties:

- $f(S) = f(S \setminus D)$ for every set $S \subseteq \mathcal{N}$.
- A set $S \subseteq \mathcal{N}$ is independent if and only if $S \setminus D$ is independent and $|S| \leq k$.

This assumption can be guaranteed by artificially adding such a set D to the ground set, modifying the value and independence oracles accordingly and removing the elements of D from the solution produced by the algorithm.

Algorithm 2 performs up to I iterations. In each iteration, the algorithm finds an (almost) maximum weight independent set M in the residual matroid (*i.e.*, the matroid resulting from $\mathcal{M}$ by contracting the current solution S), where the weight of an element is defined as its marginal contribution to S. If M has a high enough weight, then a random element from it is added to the current solution S and the algorithm continues to the next iteration. Otherwise, the algorithm terminates.

In order to find M using few value oracle queries, the algorithm maintains a global variable w_u for every element $u \in \mathcal{N}$. The variable w_u is always an upper bound on $f(u \mid S)$.

Algorithm 2: Random Lazy Greedy$(f, \mathcal{M}, \delta, B, I)$

```
   // Initialization
1  Use the algorithm guaranteed by Lemma 3.3 to calculate a value opt obeying:
   f(OPT) ≤ opt ≤ 3 · f(OPT).
2  Let S_0 ← ∅.
3  Let W ← max_{u∈N} f(u).
4  foreach u ∈ N do Let w_u ← W.
   // Main Loop
5  for i = 1 to I do
6      Let M_i ← LinearGreedy().
7      if (1 − δ) · Σ_{u∈M_i} w_u ≥ B · opt then
8          Add to M_i enough dummy elements to make S_{i−1} ∪ M_i a base.
9          Let u_i be a uniformly random element of M_i.
10         Let S_i ← S_{i−1} ∪ {u_i}.
11     else return S_{i−1}.
12 Declare failure.
13 Function LinearGreedy()
14     Let M ← ∅.
15     for (w ← W; w > δW/k; w ← w(1 − δ)) do
16         foreach u ∈ N do
17             if w_u = w and S_{i−1} ∪ M ∪ {u} ∈ I then
18                 if f(u | S_{i−1}) ≤ (1 − δ)w_u then Update w_u ← w_u(1 − δ).
19                 else Add u to M.
20     return M.
```

We begin the analysis of Algorithm 2 by showing it has the complexity required by Lemma 3.4.

Observation 3.8. *Algorithm 2 makes $O(In\delta^{-1}\ln(k/\delta))$ independence oracle queries and $O(Ik + n\delta^{-1}\ln(k/\delta))$ value oracle queries.*

Proof. The algorithm guaranteed by Lemma 3.3 uses only $O(n \ln k)$ oracle queries, which is upper bounded by both guarantees of this observation, and thus, can be ignored. The first part of the observation now follows by multiplying the following values:

- Every iteration of the internal loop of `LinearGreedy` uses a single independence oracle query, and this loop repeats n times.
- The number of iterations performed by the external loop of `LinearGreedy` is:

$$\lceil \ln_{1-\delta}(\delta/k) \rceil \leq 1 - \frac{\ln(k/\delta)}{\ln(1-\delta)} \leq 1 + \frac{\ln(k/\delta)}{\delta} \ .$$

- The main loop of the algorithm repeats at most I times.

The second part of the observation holds since every time a value oracle query is made by Algorithm 2, one of two things must happen: either an element is added to M or w_u is decreased for some element $u \in \mathcal{N}$. Observe that at most Ik elements can be added to M, and the number of times w_u can be decreased for every element $u \in \mathcal{N}$ is at most:

$$\lceil \ln_{1-\delta}(\delta/k) \rceil \leq 1 + \frac{\ln(k/\delta)}{\delta} \ . \qquad \square$$

To prove the other properties guaranteed by Lemma 3.4, we first need some notation. Let i_ℓ be the (random) largest index for which S_{i_ℓ} is defined by the algorithm. For ease of notation, we define for every $0 \leq i \leq I$ the value $r(i) = \min\{i, i_\ell\}$. Notice that $S_{r(i)}$ is always defined, even when S_i is not assigned in a given execution of Algorithm 2. The following lemma lower bounds the expected value of $S_{r(i)}$.

Lemma 3.9. *For every $0 \leq i \leq I$, $\mathbb{E}[f(S_{r(i)})] \geq B \cdot (\mathbb{E}[r(i)]/k) \cdot f(OPT)$.*

Proof. We prove the lemma by induction on i. For $i = 0$:

$$\mathbb{E}[f(S_{r(0)})] \geq 0 = \mathbb{E}[r(0)] \ .$$

Next, assume the claim is true for $i-1 \geq 0$, and let us prove it for i. Fix an event A_i specifying the random decisions made by Algorithms 2 before iteration i. All the probabilities and expectations from this point till we unfix A_i are implicitly conditioned on A_i. Notice that once A_i is fixed the sets S_{i-1} and M_i become deterministic and so is the question whether $i \leq i_\ell$. Based on the last question, we have two cases. If $i \leq i_\ell$, then the definition of Algorithm 2 guarantees $\sum_{u \in M_i} f(u \mid S_{i-1}) \geq (1-\delta) \cdot \sum_{u \in M_i} w_u \geq B \cdot \texttt{opt} \geq B \cdot f(OPT)$. Since u_i is a random element from M_i, we get:

$$\begin{aligned}\mathbb{E}[f(S_{r(i)})] &= f(S_{r(i-1)}) + \mathbb{E}[f(u_i \mid S_{i-1})] \geq f(S_{r(i-1)}) + \frac{\sum_{u \in M_i} f(u \mid S_{i-1})}{|M_i|} \\ &\geq f(S_{r(i-1)}) + \frac{B \cdot f(OPT)}{k} = f(S_{r(i-1)}) + \frac{B \cdot [r(i) - r(i-1)]}{k} \cdot f(OPT) \ .\end{aligned}$$

Consider now the case $r(i) < i$. In this case,

$$\mathbb{E}[f(S_{r(i)})] = f(S_{r(i-1)}) = f(S_{r(i-1)}) + \frac{B \cdot [r(i) - r(i-1)]}{k} \cdot f(OPT) \ .$$

In conclusion, the inequality $\mathbb{E}[f(S_{r(i)})] \geq f(S_{r(i-1)}) + Bk^{-1} \cdot [r(i) - r(i-1)] \cdot f(OPT)$ holds in both cases. Moreover, since this inequality holds conditioned on every given event A_i, it holds also unconditionally. Therefore, unfixing the event A_i, we get:

$$\begin{aligned}\mathbb{E}[f(S_{r(i)})] &= \mathbb{E}[f(S_{r(i-1)})] + \frac{B \cdot \mathbb{E}[r(i) - r(i-1)]}{k} \cdot f(OPT) \\ &\geq \frac{B \cdot \mathbb{E}[r(i-1)]}{k} \cdot f(OPT) + \frac{B \cdot \mathbb{E}[r(i) - r(i-1)]}{k} \cdot f(OPT) = \frac{B \cdot \mathbb{E}[r(i)]}{k} \cdot f(OPT) \ . \qquad \square\end{aligned}$$

The above lemma implies the following very useful corollary.

Corollary 3.10. *For every* $0 \leq i \leq I$, $\mathbb{E}[r(i)] \leq k/B$.

Proof. Notice that $S_{r(i)}$ is always a feasible solution. Thus, $\mathbb{E}[f(S_{r(i)})] \leq f(OPT)$. Using the lower bound on $\mathbb{E}[f(S_{r(i)})]$ given by Lemma 3.9, we get:

$$\frac{\mathbb{E}[r(i)] \cdot B}{k} \cdot f(OPT) \leq f(OPT) \Rightarrow \mathbb{E}[r(i)] \leq \frac{k}{B} \ . \qquad \square$$

We are now ready to prove the upper bound on the failure probability of Algorithm 2 given by Lemma 3.4. Let G denote the event that Algorithm 2 succeeds (*i.e.*, it does not declare failure).

Observation 3.11. $\Pr[\bar{G}] \leq k(BI)^{-1}$.

Proof. Notice that Algorithm 2 fails exactly when $i_\ell = I$. Hence, by Markov's inequality:

$$\Pr[\bar{G}] = \Pr[i_\ell = I] = \Pr[r(I) = I] \leq \frac{\mathbb{E}[r(I)]}{I} \leq \frac{k}{BI} \ . \qquad \square$$

Our final objective in this section is to prove Algorithm 2 obeys item (iii) of Lemma 3.4. The following lemma proves the first part of this item.

Lemma 3.12. *Conditioned on* G, *every independent set* $S' \subseteq \mathcal{N} \setminus S_{r(I)}$ *of the matroid* $\mathcal{M}/S_{r(I)}$ *must have:* $\sum_{u \in S'} f(u \mid S_{r(I)}) \leq (1-\delta)^{-2}(3B+\delta) \cdot f(OPT)$.

Proof. In this proof, given two sets $A, B \subseteq \mathcal{N}$, we use the notation $f(A : B) = \sum_{u \in A} f(u \mid B)$ to the denote the total marginal contribution to B of A's elements. Since we are conditioned on G, Algorithm 2 stopped at some iteration $i = r(I) + 1$ after observing $(1-\delta) \cdot f(M_i : S_{i-1}) < B \cdot \texttt{opt} \leq 3B \cdot f(OPT)$. Let OPT' be the set maximizing $f(\cdot : S_{i-1})$ among the independent subsets of $\mathcal{M}/S_{i-1}$. To complete the proof, it is enough to show that $f(M_i : S_{i-1}) \geq (1-\delta) \cdot f(OPT' : S_{i-1}) - \delta \cdot f(OPT)$.

The function `LinearGreedy` constructs M_i element by element. For every $1 \leq j \leq |M_i|$, let $M_{i,j}$ be the set M_i after j elements are added to it. For consistency, we also define $M_{i,0} = \varnothing$. Let OPT'_j be a base maximizing $f(\cdot : S_{i-1})$ among the bases of $\mathcal{M}/S_{i-1}$ containing $M_{i,j}$. Clearly, $f(OPT' : S_{i-1}) = f(OPT'_0 : S_{i-1})$.

Fix an arbitrary $1 \leq j \leq |M_i|$, and let $v_j = M_{i,j} \setminus M_{i,j-1}$ be the j^{th} element added to M_i. If $v_j \in OPT'_{j-1}$ then we define $v'_j = v_j$. Otherwise, let v'_j be an arbitrary element of $\mathcal{N} \setminus M_{i,j}$ belonging to the (single) cycle of $OPT'_{j-1} \cup \{v_j\}$. Notice that $(OPT'_j \setminus \{v'_j\}) \cup \{v_j\}$ is always a base of $\mathcal{M}/S_{i-1}$ containing $M_{i,j}$, and thus, $f(OPT'_j : S_{i-1}) \geq f(OPT'_{j-1} \setminus \{v'_j\} \cup \{v_j\} : S_{i-1})$. Since $M_{i,j-1} \cup \{v'_j\}$ is independent in $\mathcal{M}/S_{i-1}$, at the time point when the algorithm added v_j to M_i, the following held:

$$f(v_j \mid S_{i-1}) \geq w_{v_j}(1-\delta) = w(1-\delta) \geq w_{v'_j}(1-\delta) \geq f(v'_j \mid S_{i-1}) \cdot (1-\delta) \ ,$$

which implies:

$$f(OPT'_j : S_{i-1}) \geq f(OPT'_{j-1} \setminus \{v'_j\} \cup \{v_j\} : S_{i-1}) \geq f(OPT'_{j-1} : S_{i-1}) - \delta \cdot f(v'_j \mid S_{i-1}) \ .$$

Adding up the above inequality for every $1 \leq j \leq |M_i|$ results in:

$$f(OPT'_{|M_i|} : S_{i-1}) \geq f(OPT'_0 : S_{i-1}) - \delta \cdot \sum_{j=1}^{|M_i|} f(v'_j \mid S_{i-1}) \geq (1-\delta) \cdot f(OPT' : S_{i-1}) \ .$$

Finally, every element of $u \in OPT'_{|M_i|} \setminus M_i$, must have $f(u \mid S) \leq \delta W/k \leq \delta \cdot f(OPT)/k$. Thus,

$$\begin{aligned} f(OPT'_{|M_i|} : S_{i-1}) &\leq f(M_i : S_{i-1}) + |OPT'_{|M_i|} \setminus M_i| \cdot \frac{\delta}{k} \cdot f(OPT) \\ &\leq f(M_i : S_{i-1}) + \delta \cdot f(OPT) \ . \end{aligned}$$ □

To prove the second part of item (iii) of Lemma 3.4, we need the following lemma from [6], which can be found (with a different notation) as Corollary 39.12a in [39].

Lemma 3.13. *If A and B are two bases of a matroid $\mathcal{M} = (\mathcal{N}, \mathcal{I})$, then there exists a bijection $\phi : A \setminus B \to B \setminus A$ such for every $u \in B \setminus A$, $A \cup \{u\} \setminus \{\phi(u)\} \in \mathcal{I}$.*

Using the above lemma we can now define for every $0 \leq i \leq I$, a random variable OPT_i via the following recursive definition.

- OPT_0 is an arbitrary base of $\mathcal{M}$ obtained from OPT by adding enough dummy elements.
- For $i > 0$, OPT_i depends on i_ℓ. If $i > i_\ell$, then OPT_i is an undefined set (its value is never used in the proofs below). Otherwise, if $i \leq i_\ell$, then let $\phi : M_i \to OPT_{i-1}$ be a one to one function having the following properties:
  - For every $u \in M_i$, $S_{i-1} \cup (OPT_{i-1} \setminus \{\phi(u)\}) \cup \{u\} \in \mathcal{I}$.
  - For every $u \in OPT_{i-1} \cap M_i$, $\phi(u) = u$.

  Then, $OPT_i = OPT_{i-1} \setminus \{\phi(u_i)\}$.

The existence of a function ϕ having the properties given in the above definition follow from Lemma 3.13 since both $OPT_{i-1} \cup S_{i-1}$ and $M_i \cup S_{i-1}$ are bases of $\mathcal{M}$ (the fact that $OPT_{i-1} \cup S_{i-1}$ is a base can be easily verified by induction). When there are multiple possible choices for ϕ, we assume one is picked based on the contents of the sets OPT_{i-1} and S_{i-1} alone.

The following lemma and corollary show that $OPT_{r(I)}$ is a random set having the properties required by the second part of item (iii) of Lemma 3.4.

Lemma 3.14. *For every $0 \leq i \leq I$, $\mathbb{E}[f(OPT_{r(i)})] \geq f(OPT) - 2 \cdot \mathbb{E}[r(i)] \cdot f(OPT)/k$.*

Proof. We prove the lemma by induction on i. For $i = 0$:

$$\mathbb{E}[f(OPT_{r(i)})] = f(OPT_0) = f(OPT) - \frac{2 \cdot 0}{k} \cdot f(OPT) \ .$$

Next, assume the claim holds for $i-1 \geq 0$, and let us prove it for i. Like in the proof of Lemma 3.9, we fix an event A_i specifying the random decisions made by Algorithms 2 before iteration i. All the probabilities and expectations from this point till we unfix A_i are implicitly conditioned on A_i. Notice that once A_i is fixed the sets S_{i-1}, M_i and OPT_{i-1} become deterministic and so is the question whether $i \leq i_\ell$. Based on the last question, we have two cases. If $i \leq i_\ell$, then every element of $OPT_{r(i)-1}$ belongs to $OPT_{r(i)}$ with probability $1 - 1/|OPT_{r(i)-1}| = 1 - (k - r(i) + 1)^{-1}$. Thus, by Lemma 2.1,

$$\begin{aligned} \mathbb{E}[f(OPT_r(i))] &\geq \left(1 - \frac{1}{k - r(i) + 1}\right) \cdot f(OPT_{r(i-1)}) \geq \left(1 - \frac{2}{k}\right) \cdot f(OPT_{r(i-1)}) \\ &\geq f(OPT_{r(i-1)}) - \frac{2}{k} \cdot f(OPT) = f(OPT_{r(i-1)}) - \frac{2[r(i) - r(i-1)]}{k} \cdot f(OPT) \ , \end{aligned}$$

where the second inequality holds since $r(i) \leq i \leq I \leq k/2$ and the third inequality holds since $OPT_{r(i-1)} \subseteq OPT_0$. Consider now the case $i > i_\ell$. In this case:

$$\mathbb{E}[f(OPT_{r(i)})] = f(OPT_{r(i-1)}) = f(OPT_{r(i-1)}) - \frac{2[r(i) - r(i-1)]}{k} \cdot f(OPT) \ .$$

In conclusion, the inequality $\mathbb{E}[f(OPT_{r(i)})] \geq f(OPT_{r(i-1)}) - 2k^{-1}[r(i) - r(i-1)] \cdot f(OPT)$ holds in both cases. Moreover, since this inequality holds conditioned on every given event A_i, it holds also unconditionally. Therefore, unfixing the event A_i, we get:

$$\begin{aligned}\mathbb{E}[f(OPT_{r(i)})] &\geq \mathbb{E}[f(OPT_{r(i-1)})] - \frac{2 \cdot \mathbb{E}[r(i) - r(i-1)]}{k} \cdot f(OPT) \\ &\geq f(OPT) - \frac{2 \cdot \mathbb{E}[r(i-1)]}{k} \cdot f(OPT) - \frac{2 \cdot \mathbb{E}[r(i) - r(i-1)]}{k} \cdot f(OPT) \\ &= f(OPT) - \frac{2 \cdot \mathbb{E}[r(i)]}{k} \cdot f(OPT) \ .\end{aligned}$$

□

Corollary 3.15. *For every $0 \leq i \leq I$, $\mathbb{E}[f(OPT_{r(i)}) \mid G] \geq [1 - B^{-1}(2 + k/I)] \cdot f(OPT)$.*

Proof. By the law of total expectation:

$$\mathbb{E}[f(OPT_{r(i)}) \mid G] \geq \Pr[G] \cdot \mathbb{E}[f(OPT_{r(i)}) \mid G] = \mathbb{E}[f(OPT_{r(i)})] - \Pr[\bar{G}] \cdot \mathbb{E}[f(OPT_{r(i)}) \mid \bar{G}] \ .$$

The term $\mathbb{E}[f(OPT_{r(i)}) \mid \bar{G}]$ can be upper bounded by $f(OPT)$ because $OPT_{r(i)}$ is always a subset of OPT (possibly, plus dummy elements). Combining this observation with Observation 3.11 and Lemma 3.14 gives:

$$\begin{aligned}\mathbb{E}[f(OPT_{r(i)}) \mid G] &\geq \left[f(OPT) - \frac{2 \cdot \mathbb{E}[r(i)]}{k} \cdot f(OPT)\right] - \frac{k}{BI} \cdot f(OPT) \\ &\geq \left[1 - \frac{2 + k/I}{B}\right] \cdot f(OPT) \ ,\end{aligned}$$

where the last inequality follows from Corollary 3.10. □

3.5 Generalized Partition Matroids

In this section we prove Theorem 1.2, which states that for generalized partition matroids it is possible to improve over the result given by Theorem 1.1 for general matroids. In this context, there is no longer an independence oracle, thus, we are only interested in the number of value oracle queries used by our algorithm (and guaranteeing that the time complexity is bounded by the same expression). Throughout this section, $\mathcal{M}$ is a generalized partition matroid and $h \leq k$ is the number of partitions in $\mathcal{M}$. We denote by $\mathcal{N}_j \subseteq \mathcal{N}$, the set of elements in the j^{th} partition of $\mathcal{M}$ and by k_j the maximum number of elements that can be taken from this partition (*i.e.*, $\sum_{j=1}^{h} k_j = k$ and a set $S \subseteq \mathcal{N}$ is independent if and only if $|S \cap \mathcal{N}_j| \leq k_j$ for every $1 \leq j \leq h$).

Swap rounding is an algorithm suggested by [10] for rounding fractional points in the matroid polytope $\mathcal{P}(\mathcal{M})$ into an (integral) independent set. Badanidiyuru and Jondrák [4] observed that swap rounding has a time complexity of $O(bk^2)$ when the fractional point is a convex combination of b independent sets. The following observation states that this bound can be improved for generalized partition matroids.

Observation 3.16. *Given a generalized partition matroid and a fractional point $x \in \mathcal{P}(\mathcal{M})$ which is a convex combination of b independent sets, swap rounding can be used to round x using $O(bk)$ time and no value oracle queries.*

Proof. It is possible to represent independent sets $S \in \mathcal{I}$ in such a way that given an index $1 \leq j \leq h$ one can find an element $u \in S \cap \mathcal{N}_j$ in $O(1)$ time (if such an element exists). The pseudo-code of swap rounding given by [10] requires only $O(bk)$ time when the sets composing the fractional point

are given in a representation having the above property. Moreover, the standard representation of an independent set as a list of items can be converted into a representation having the above property in $O(k)$ time, hence, all b sets can be converted into such a representation in $O(bk)$ time. □

Plugging the above improved time complexity into the result of [4] yields the following improved version of Corollary 3.2.

Corollary 3.17. *If $\max_{S \in \mathcal{I}} \sum_{u \in S} f(u) \leq c \cdot f(OPT)$ for some value c, then for every $\delta > 0$ there exists a $(1 - e^{-1} - \delta)$-approximation algorithm for maximizing f subject to a generalized partition matroid $\mathcal{M}$ using $O(cn\delta^{-4} \ln^2(\frac{n}{\delta}))$ value oracle queries and a time complexity bounded by the same expression.*

To get an improved result for generalized partition matroids, we also need to improve the implementation of the function `LinearGreedy` of Algorithm 2. We observe that for such matroids one can handle each partition separately in the function `LinearGreedy`. The separation allows us to remove the element-wise check whether a given element can be added to the current solution, and replace it with a partition-wise check whether the current solution already has the maximum allowed number of partition elements. Additionally, we replace the element specific variables w_u with sets $T_{j,w}$ containing all the elements $u \in \mathcal{N}_j$ that logically have $w_u = w$. This change allows us to avoid scanning all the elements of $\mathcal{N}_j$ in order to find the elements $u \in \mathcal{N}_j$ having $w_u = w$. Finally, for further acceleration, we introduce for each partition a list $\mathcal{T}_j$ of the non-empty sets $T_{j,w}$.

The improved implementation of `LinearGreedy` is given as Algorithm 3. The initialization part should be executed once before the first call to `LinearGreedy`.

Observation 3.18. *Algorithm 2 with the new implementation of* `LinearGreedy` *given by Algorithm 3 uses $O(Ik + n\delta^{-1} \ln(k/\delta))$ value oracle queries, and has a time complexity bounded by the same expression.*

Proof. Aside from the n value oracle queries used to calculate W and the $O(n \ln k)$ oracle queries used by the algorithm guaranteed by Lemma 3.3, every access to f is followed by one of two events: either an element is added to M or the "logical" w_u of an element u is reduced. The total number of elements that can be added to M is Ik, and the total number of values a single "logical" w_u can have is

$$\lceil \ln_{1-\delta}(\delta/k) \rceil \leq 1 - \frac{\ln\left(\frac{k}{\delta}\right)}{\ln(1-\delta)} \leq 1 + \frac{\ln\left(\frac{k}{\delta}\right)}{\delta} \ .$$

This completes the proof of the first part of the observation.

Regarding the time complexity, notice that the main part of Algorithm 2 uses $O(n + Ik)$ time (excluding the calls to `LinearGreedy`) and the initialization step introduced in Algorithm 3 uses $O(n + k \ln_{1-\delta}(\delta/k)) = O(n\delta^{-1} \ln(k/\delta))$ time. Thus, we only need to bound the time complexity of the new implementation of `LinearGreedy`.

Each iteration of the loop starting on Line 12 of Algorithm 3 takes $O(1)$ time (notice that $\mathcal{T}_j$ can be updated in $O(1)$ time if $\mathcal{T}_j$ is represented as a double linked list). Moreover, this loop always make at least one iteration, and each iteration (except for maybe one per execution of `PartitionGreedy`) access f. Hence, we can bound the time required for `LinearGreedy` by $O(h) = O(k)$ plus the number of value oracle queries it uses. The observation now follows since `LinearGreedy` is called only I times by Algorithm 2. □

Corollary 3.19. *Algorithm 1 with the new implementation of* `LinearGreedy` *given by Algorithm 3 makes at most $O(k\lambda + kn\lambda^{-1}\varepsilon^{-5} \ln^2(\frac{n}{\varepsilon}))$ value oracle queries.*

Algorithm 3: New Implementation of `LinearGreedy`

```
   // Initialization
1  for j = 1 to k do
2      for (w ← W; w > δW/k; w ← w(1 − δ)) do T_{j,w} ← ∅.
3      foreach u ∈ N_j do Add u to T_{j,W}.
4      Let T_j ← {T_{j,W}}.
5  Function LinearGreedy()
6      Let M ← ∅.
7      for j = 1 to h do Update M ← M ∪ PartitionGreedy(j)
8      return M.
9  Function PartitionGreedy(j)
10     Let M_j ← ∅.
11     foreach T_{j,w} ∈ T_j in decreasing weight order do
12         foreach u ∈ T_{j,w} do
13             if |M_j| = k_j then return M_j.
14             if f(u | S_{i−1}) ≤ w(1 − δ) then
15                 Remove u from T_{j,w} and add it to T_{j,w(1−δ)} (if such a set exists).
16                 Update T_j by removing T_{j,w} if it became empty and adding T_{j,w(1−δ)} if it was
                   empty before.
17             else Add u to M_j.
18     return M_j.
```

Proof. The observation follows by adding up the guarantees on the number of value oracle queries given by Corollary 3.17 and Observation 3.18. □

Theorem 1.2 now follows immediately by the following choice of λ:[6]

$$\lambda = \begin{cases} k & \text{if } k \leq \sqrt{n\varepsilon^{-5}}\ln(\frac{n}{\varepsilon}) \ , \\ \sqrt{n\varepsilon^{-5}}\ln(\frac{n}{\varepsilon}) & \text{otherwise} \ . \end{cases}$$

4 Random Sampling Algorithms

In this section we consider an algorithm for the problem $\max\{f(S) : |S| \leq k\}$ based on random sampling. The algorithm has k iterations and two parameters $p \in (0, 1]$ and $1 \leq s \leq \lceil pn \rceil$. In each iteration the algorithm picks a uniformly random sample M of the ground set containing $\lceil pn \rceil$ elements of $\mathcal{N}$. The elements of M are then assumed to be ordered according to their marginal contribution to the current solution, and a random element out of the top s elements of M is added to the solution. If s is not integral, then each one of the top $\lfloor s \rfloor$ elements of M is added with probability $1/s$ and the $\lceil s \rceil$ element is added with the remaining probability. A formal description of the algorithm is given as Algorithm 4. It is important to observe that the sample M can contain elements that already belong to the solution.

Observation 4.1. *Algorithm 4 uses $O(k + nkp)$ value oracle queries.*

[6]For $n \leq 2$ this choice might not be in the valid range $[1, k]$, however, for a constant n the problem can be optimally solved using a constant number of value oracle queries.

Algorithm 4: Random Sampling Algorithm(f, k, p, s)

Initialize: $S_0 \leftarrow \varnothing$.
for $i = 1$ **to** k **do**
  Let M_i be a uniformly random set containing $\lceil pn \rceil$ elements of $\mathcal{N}$.
  Let d_i be a uniformly random value from the range $(0, s]$.
  Let u_i be the element of M_i with the $\lceil d_i \rceil$-th largest marginal contribution to S_{i-1}.
  if $f(u_i \mid S_{i-1}) \geq 0$ **then** $S_i \leftarrow S_{i-1} \cup \{u_i\}$.
return S_k.

Proof. Algorithm 4 performs k iterations. Each iteration of Algorithm 4 requires $O(|M_i|) = O(1 + pn)$ value oracle queries. □

By setting the parameters of Algorithm 4 to $s = 1$ and $p = \ln \varepsilon^{-1}/k$, we get a folklore algorithm satisfying the properties guaranteed by Theorem 1.3 for $\varepsilon \in (e^{-k}, 1 - e^{-1})$ (for $\varepsilon \in (0, e^{-k}]$ the standard greedy algorithm of [38] fulfills the requirements of the theorem, and for $\varepsilon \geq 1 - e^{-1}$ the theorem is void). For completeness, we prove this folklore result in Appendix A.

In the rest of this section, we use Algorithm 4 to prove the following theorem. Theorem 1.4 follows by combining this theorem with the result proved in Section 5.

Theorem 4.2. *There exists an algorithm that given a general non-negative submodular function $f : 2^{\mathcal{N}} \to \mathbb{R}^+$, and parameters $k \geq 1$ and $\varepsilon > 0$, finds a solution $S \subseteq \mathcal{N}$ of size $|S| \leq k$ where: $f(S) \geq (1/e - \varepsilon) \cdot \max\{f(T) : T \subseteq \mathcal{N}, |T| \leq k\}$ and the algorithm performs $O(n\varepsilon^{-2} \ln \varepsilon^{-1})$ value oracle queries.*

Let δ be the (single) $\delta > 0$ for which $8\delta^{-2} \cdot \ln(2\delta^{-1})) = k$. If $\varepsilon \leq \delta$, then the random greedy algorithm of [7] can be used to get (e^{-1})-approximation using $O(nk) = O(n\delta^{-2} \ln \delta^{-1}) = O(n\varepsilon^{-2} \ln \varepsilon^{-1})$ value oracle queries. On the other hand, if $\varepsilon \geq e^{-1}$, then Theorem 4.2 is void. Thus, the interesting case, which we assume from now on, is $\varepsilon \in (\delta, e^{-1})$. We set the parameters of Algorithm 4 as follows: $s = k\lceil pn \rceil / n$ and $p = 8k^{-1}\varepsilon^{-2} \cdot \ln(2\varepsilon^{-1})$. Notice that $p \in (0, 1]$ since $\varepsilon \geq \delta$ and $1 \leq s \leq \lceil pn \rceil$ since $\varepsilon \leq e^{-1}$. Plugging our chosen value of p into Observation 4.1 yields the time complexity given in Theorem 4.2.

Let A_i be an event determining all the random decisions of the algorithm up to iteration i (excluding). In the first part of the proof, we fix an iteration $1 \leq i \leq k$ and an event A_i. All the probabilities and expectations in this part of the proof are implicitly conditioned in A_i. Notice that S_{i-1} is a deterministic set when conditioned on A_i. We denote by $v_1, v_2, \ldots, v_k$ the k elements with the maximum marginal contribution to S_{i-1}, sorted in a non-increasing marginal contribution order, and let X_j be an indicator for the event $u_i = v_j$.

Lemma 4.3. $\mathbb{E}[\sum_{j=1}^{k} X_j] > 1 - \varepsilon$.

Proof. Let E be the event that $|M_i \cap \{v_1, v_2, \ldots, v_k\}| \geq (1 - \varepsilon/2)s$. Observe that when E occurs there is a range of size at least $(1 - \varepsilon/2)s$ of possible values for d_i that make $\sum_{j=1}^{k} X_j$ equal to 1. Hence,

$$\mathbb{E}\left[\sum_{j=1}^{k} X_j \;\middle|\; E\right] \geq \frac{(1 - \varepsilon/2)s}{s} = 1 - \varepsilon/2 \ .$$

The random variable $|M_i \cap \{v_1, v_2, \ldots, v_k\}|$ has an hypergeometric distribution, and thus, obeys the Chernoff bounds (see, *e.g.*, Theorem 1.17 of [14]). Notice that the expectation of this random

variable is s, hence,

$$\Pr[\bar{E}] = \Pr[|M_i \cap \{v_1, v_2, \ldots, v_k\}| \leq (1-\varepsilon/2)s] \leq e^{-\frac{(\varepsilon/2)^2 \cdot s}{2}} \leq e^{-\frac{\varepsilon^2 kp}{8}} = \varepsilon/2 \ .$$

Combining the two above inequalities, we get:

$$\mathbb{E}\left[\sum_{j=1}^{k} X_j\right] \geq \mathbb{E}\left[\sum_{j=1}^{k} X_j \;\middle|\; E\right] \cdot (1 - \Pr[\bar{E}]) \geq (1-\varepsilon/2)^2 > 1 - \varepsilon \ . \qquad \square$$

Given two elements v_{j_1} and v_{j_2} with $j_1 < j_2$, we expect v_{j_1} to have at least as high a probability to be u_i as v_{j_2}. This is proved formally by the following lemma.

Lemma 4.4. *$\mathbb{E}[X_j]$ is a non-increasing function of j.*

Proof. Fix an arbitrary pair of indexes $1 \leq j_1 < j_2 \leq k$. We have to prove that $\mathbb{E}[X_{j_1}] \geq \mathbb{E}[X_{j_2}]$. For every set $S \subseteq \mathcal{N}$, let $\sigma(S)$ be the following set:

$$\sigma(S) = \begin{cases} S & \text{if } v_{j_1}, v_{j_2} \in S \text{ or } v_{j_1}, v_{j_2} \notin S \ , \\ S \cup \{v_{j_2}\} \setminus \{v_{j_1}\} & \text{if } v_{j_1} \in S \text{ and } v_{j_2} \notin S \ , \\ S \cup \{v_{j_1}\} \setminus \{v_{j_2}\} & \text{if } v_{j_2} \in S \text{ and } v_{j_1} \notin S \ . \end{cases}$$

Since $|S| = |\sigma(S)|$, we get $\Pr[M_i = S] = \Pr[M_i = \sigma(S)]$ for every set $S \subseteq \mathcal{N}$. Moreover, it is not difficult to verify that by the definition of Algorithm 4, $\mathbb{E}[X_{j_1} \mid M_i = S] \geq \mathbb{E}[X_{j_2} \mid M_i = \sigma(S)]$. Combining all these observations yields:

$$\mathbb{E}[X_{j_1}] = \sum_{S \subseteq \mathcal{N}} \Pr[M_i = S] \cdot \mathbb{E}[X_{j_1} \mid M_i = S] \geq \sum_{S \subseteq \mathcal{N}} \Pr[M_i = \sigma(S)] \cdot \mathbb{E}[X_{j_2} \mid M_i = \sigma(S)] = \mathbb{E}[X_{j_2}] \ ,$$

where the last equality uses the observation that σ is a bijection. $\square$

Using the two above lemmata, it is now possible to lower bound the expected gain of Algorithm 4 in iteration i.

Lemma 4.5. $\mathbb{E}[f(S_i) - f(S_{i-1})] \geq (1-\varepsilon) \cdot \frac{f(OPT \cup S_{i-1}) - f(S_{i-1})}{k}$.

Proof. Observe that:

$$\begin{aligned} \mathbb{E}[f(S_i) - f(S_{i-1})] = \mathbb{E}[\max\{f(u_i \mid S_{i-1}), 0\}] &\geq \sum_{j=1}^{k} [\mathbb{E}[X_j] \cdot \max\{f(v_j \mid S_{i-1}), 0\}] \\ &\geq \frac{\sum_{j=1}^{k} \mathbb{E}[X_j] \cdot \sum_{j=1}^{k} \max\{f(v_j \mid S_{i-1}), 0\}}{k} \ . \end{aligned}$$

where the last inequality holds by Chebyshev's sum inequality since $\max\{f(v_j \mid S_{i-1}), 0\}$ is non-increasing in j by definition and $\mathbb{E}[X_j]$ is non-increasing in j by Lemma 4.4. By the definition of the v_j's and the submodularity of the objective:

$$\sum_{j=1}^{k} \max\{f(v_j \mid S_{i-1}), 0\} \geq \sum_{u \in OPT} f(u \mid S_{i-1}) \geq f(OPT \cup S_{i-1}) - f(S_{i-1}) \ .$$

To complete the proof of the lemma recall that, by Lemma 4.3, $\mathbb{E}[X_j] \geq 1 - \varepsilon$. $\square$

At this point we unfix the event A_i. The probabilities and expectations in the rest of this section are no longer implicitly conditioned on A_i.

Corollary 4.6. *For every* $0 \leq i \leq k$, $\mathbb{E}[f(S_i)] \geq (i/k) \cdot [(1-1/k)^{i-1} - \varepsilon] \cdot f(OPT)$.

Proof. First, notice that since Lemma 4.5 holds for every given event A_i, it holds in expectation also unconditionally. More formally, we get for every $1 \leq i \leq k$,

$$\mathbb{E}[f(S_i) - f(S_{i-1})] \geq (1-\varepsilon) \cdot \frac{\mathbb{E}[f(OPT \cup S_{i-1})] - \mathbb{E}[f(S_{i-1})]}{k} \ .$$

Let us lower bound $\mathbb{E}[f(OPT \cup S_{i-1})]$. Algorithm 4 adds each element to its solution with probability at most: $(\lceil pn \rceil / n)/s = 1/k$. Hence, each element belongs to S_{i-1} with probability at most $1-(1-1/k)^{i-1}$. Let $h(S) = h(S \cup OPT)$. Since h is a non-negative submodular function, we get by Lemma 2.2,

$$\mathbb{E}[f(OPT \cup S_i)] = \mathbb{E}[h(S_i)] \geq (1-1/k)^i \cdot h(\varnothing) = (1-1/k)^i \cdot f(OPT) \ .$$

Combining the two above inequalities yields,

$$\begin{aligned}\mathbb{E}[f(S_i) - f(S_{i-1})] &\geq (1-\varepsilon) \cdot \frac{(1-1/k)^{i-1} \cdot f(OPT) - \mathbb{E}[f(S_{i-1})]}{k} \\ &\geq \frac{[(1-1/k)^{i-1} - \varepsilon] \cdot f(OPT) - \mathbb{E}[f(S_{i-1})]}{k} \ .\end{aligned}$$

We are now ready to prove the corollary by induction on i. For $i = 0$, the corollary holds since $f(S_0) \geq 0 = (0/k) \cdot [(1-1/k)^{-1} - \varepsilon] \cdot f(OPT)$. Assume the corollary holds for $i-1 \geq 0$, let us prove it for i.

$$\begin{aligned}\mathbb{E}[f(S_i)] &\geq \mathbb{E}[f(S_{i-1})] + \frac{\left[\left(1-\frac{1}{k}\right)^{i-1} - \varepsilon\right] \cdot f(OPT) - \mathbb{E}[f(S_{i-1})]}{k} \\ &= (1-1/k) \cdot \mathbb{E}[f(S_{i-1})] + \frac{[(1-1/k)^{i-1} - \varepsilon] \cdot f(OPT)}{k} \\ &\geq (1-1/k) \cdot \frac{i-1}{k} \cdot [(1-1/k)^{i-2} - \varepsilon] \cdot f(OPT) + \frac{[(1-1/k)^{i-1} - \varepsilon] \cdot f(OPT)}{k} \\ &\geq \frac{i}{k} \cdot [(1-1/k)^{i-1} - \varepsilon] \cdot f(OPT) \ .\end{aligned}$$

□

Plugging $i = k$ into the above lemma yields:

$$\mathbb{E}[f(S_k)] \geq [(1-1/k)^{k-1} - \varepsilon] \cdot f(OPT) \geq (e^{-1} - \varepsilon) \cdot f(OPT) \ ,$$

which completes the proof of the approximation ratio guaranteed by Theorem 4.2.

5 Cardinality Constraint via Thresholding

In this section we describe an algorithm for the problem $\max\{f(S) : |S| \leq k\}$ based on a combination of the Random Greedy of [7] and the thresholding algorithm of [4]. We prove that this algorithm obeys all the requirements of the following theorem. Theorem 1.4 follows by combining this theorem with the result proved in Section 4.

Theorem 5.1. *There exists an algorithm that given a general non-negative submodular function $f : 2^{\mathcal{N}} \to \mathbb{R}^+$, and parameters $k \geq 1$ and $\varepsilon > 0$, finds a solution $S \subseteq \mathcal{N}$ of size $|S| \leq k$ where: $f(S) \geq (1/e - \varepsilon) \cdot \max\{f(T) : T \subseteq \mathcal{N}, |T| \leq k\}$ and the algorithm performs $O(k\sqrt{n\varepsilon^{-1}\ln(k/\varepsilon)} + n\varepsilon^{-1}\ln(k/\varepsilon))$ value oracle queries.*

All algorithms considered in this section assume the existence of a set $D \subseteq \mathcal{N}$ of at least $2k$ dummy elements having a weight of 0. This assumption can be justified by explicitly adding such a set of elements to the ground set. We may also assume $\varepsilon < e^{-1}$, since the theorem is void otherwise. The first algorithm we consider is Algorithm 5. This algorithm accepts an error parameter $\delta \in (0, e^{-1})$ which we set later. The function `FillM` described in Algorithm 5 has a somewhat non-standard semantics. Namely, at the first time it is called it executes from its beginning till reaching the command **yield**. At every additional call, `FillM` starts executing from the place where it stopped on the last call, and continues till reaching the command **yield** again.

On a more intuitive level, Algorithm 5 constructs a solution using k iterations. In each iteration the function `FillM` is used to fill the set M with the k elements having the largest marginal contributions (up to an approximation error). Then, a uniformly random element of M is added to the solution, and elements of M whose marginal contribution decreased significantly following the addition are removed from M.

Algorithm 5: Random Lazy Greedy Simple(f, k, δ)

```
   // Initialization
 1 M, S_0 ← ∅.
 2 Let w, W ← max_{u∈N} f(u).
   // Main Loop
 3 for i = 1 to k do
 4 |   Call FillM (M).
 5 |   Uniformly pick a random element u_i from M.
 6 |   Let S_i ← S_{i−1} ∪ {u_i}.
 7 |   foreach element u ∈ M do
 8 |   |   if f(u | S_i) ≤ w(1 − δ) then Remove u from M.
 9 return S_k.
10 Function FillM (M)
11 |   for (w = W; w > δW/k; w ← w(1 − δ)) do
12 |   |   foreach u ∈ N do
13 |   |   |   if f(u | S) > w(1 − δ) then
14 |   |   |   |   Add u to M.
15 |   |   |   |   if |M| = k then yield.
16 |   do forever
17 |   |   Add k − |M| dummy elements of D \ M to M.
18 |   |   yield.
```

Let us begin by analyzing the approximation ratio of Algorithm 5. We need some notation. Let M_i be the set M at the moment the algorithm picks u_i from it. For two sets $A, B \subseteq \mathcal{N}$, let $f(A : B) = \sum_{u \in A} f(u \mid B)$. Finally, let $O_i \subseteq \mathcal{N}$ be the (random) subset of size at most k maximizing $f(O_i : S_{i-1})$.

Lemma 5.2. *For every* $1 \leq i \leq k$, $f(M_i : S_{i-1}) \geq (1-\delta) \cdot f(O_i : S_{i-1}) - \delta \cdot f(OPT)$.

Proof. Let A_i be an event determining all the random choices of the algorithm in the first $i-1$ iterations. We fix an arbitrary such event A_i, and prove the lemma conditioned on this event. Notice that if the lemma holds conditioned an arbitrary event A_i, then it also holds unconditionally. Observe also that once we fix A_i, the sets M_i, O_i and S_{i-1} all become deterministic.

Let w_i be the value of w at the moment Algorithm 5 chooses u_i. There are two cases to consider. Assume first $w_i > \delta W/k$. Since the marginals of elements only decrease as the solution increases, every element $u \in \mathcal{N}$ with $f(u \mid S_{i-1}) > w_i$ must be in M_i. On the other hand, since elements with low marginals are removed from M_i, we also have $f(u \mid S_{i-1}) > w_i(1-\delta)$ for every element $u \in M_i$. Let $O'_i = \{u \in O_i \mid f(u \mid S_{i-1}) > w_i\}$. By the above discussion $O'_i \subseteq M_i$. Thus,

$$\begin{aligned} f(O_i : S_{i-1}) &= f(O'_i : S_{i-1}) + f(O_i \setminus O'_i : S_{i-1}) \leq f(O'_i : S_{i-1}) + w_i \cdot |O_i \setminus O'_i| \\ &\leq f(O'_i : S_{i-1}) + w_i \cdot |M_i \setminus O'_i| \leq f(O'_i : S_{i-1}) + \frac{f(M_i \setminus O'_i : S_{i-1})}{1-\delta} \leq \frac{f(M_i : S_{i-1})}{1-\delta} \ , \end{aligned}$$

where the second inequality holds since $|M_i| = k \geq |O_i|$ and $O'_i \subseteq O_i \cap M_i$. The last inequality holds since $f(u \mid S_{i-1}) \geq 0$ for every $u \in O'_i$. This completes the proof of the lemma for the case $w_i > \delta W/k$. Assume, now, $w_i \leq \delta W/k$. In this case every element $u \in \mathcal{N}$ having $f(u \mid S_{i-1}) > \delta W/k$ must be in M_i. Let $O'_i = \{u \in O_i \mid f(u \mid S_{i-1}) > \delta W/k\}$. Again $O'_i \subseteq M_i$, and thus,

$$\begin{aligned} f(O_i : S_{i-1}) &= f(O'_i : S_{i-1}) + f(O_i \setminus O'_i : S_{i-1}) \\ &\leq f(O'_i : S_{i-1}) + \frac{\delta W}{k} \cdot |O_i \setminus O'_i| \leq f(M_i : S_{i-1}) + \delta W \ . \end{aligned}$$

The lemma now follows by observing that $W \leq f(OPT)$. □

The above lemma can be used to derive a lower bound on the expected improvement in the solution of Algorithm 5 in a given iteration.

Lemma 5.3. *For every* $1 \leq i \leq k$, $\mathbb{E}[f(S_i)] \geq \frac{[(1-\delta)(1-1/k)^{i-1} - \delta] \cdot f(OPT) + (k-1) \cdot \mathbb{E}[f(S_{i-1})]}{k}$.

Proof. In every given iteration, Algorithms 5 adds every element $u \in \mathcal{N}$ to its solution with probability at most $1/k$. Hence, for every $u \in \mathcal{N}$, $\Pr[u \in S_i] \leq 1 - (1-1/k)^i$, which implies $\mathbb{E}[f(OPT \cup S_i)] \geq (1-1/k)^i \cdot f(OPT)$ by Lemma 2.2. Hence, by the definition of O_i and the submodularity of f:

$$\begin{aligned} \mathbb{E}[f(O_i : S_{i-1})] &\geq \mathbb{E}[f(OPT : S_{i-1})] \\ &\geq \mathbb{E}[f(OPT \cup S_{i-1}) - f(S_{i-1})] \geq (1-1/k)^{i-1} \cdot f(OPT) - \mathbb{E}[f(S_{i-1})] \ . \end{aligned}$$

Recall that a uniformly random element of M_i is added to S_{i-1} to form S_i. This observation, together with Lemma 5.2, give:

$$\begin{aligned} \mathbb{E}[f(S_i) - f(S_{i-1})] &= \frac{\mathbb{E}[f(M_i : S_{i-1})]}{k} \geq \frac{(1-\delta) \cdot \mathbb{E}[f(O_i : S_{i-1})] - \delta \cdot f(OPT)}{k} \\ &\geq \frac{(1-\delta) \cdot \left\{(1-1/k)^{i-1} \cdot f(OPT) - \mathbb{E}[f(S_{i-1})]\right\} - \delta \cdot f(OPT)}{k} \\ &\geq \frac{[(1-\delta)(1-1/k)^{i-1} - \delta] \cdot f(OPT) - \mathbb{E}[f(S_{i-1})]}{k} \ . \end{aligned}$$

□

We are now ready to prove the approximation ratio of Algorithm 5.

Corollary 5.4. *Algorithm 5 is a $(e^{-1}-2\delta)$-approximation algorithm.*

Proof. We first prove by induction on i that for $0 \leq i \leq k$: $\mathbb{E}[f(S_i)] \geq \frac{i}{k} \cdot [(1-\delta)(1-1/k)^{i-1} - \delta] \cdot f(OPT)$. For $i = 0$, the claim is trivial since: $\mathbb{E}[f(S_0)] \geq 0 = \frac{0}{k} \cdot [(1-\delta)(1-1/k)^{0-1} - \delta] \cdot f(OPT)$. Next, assume the claim holds for $i - 1 \geq 0$, and let us prove it for i. By Lemma 5.3,

$$\begin{aligned}
\mathbb{E}[f(S_i)] &\geq \frac{[(1-\delta)(1-1/k)^{i-1} - \delta] \cdot f(OPT) + (k-1) \cdot \mathbb{E}[f(S_{i-1})]}{k} \\
&\geq \frac{[(1-\delta)(1-1/k)^{i-1} - \delta] \cdot f(OPT) + (k-1) \cdot \frac{i-1}{k} \cdot [(1-\delta)(1-1/k)^{i-2} - \delta] \cdot f(OPT)}{k} \\
&\geq \frac{i}{k} \cdot [(1-\delta)(1-1/k)^{i-1} - \delta] \cdot f(OPT) \enspace .
\end{aligned}$$

For $i = k$, we get:

$$\begin{aligned}
\mathbb{E}[f(S_k)] &\geq \frac{k}{k} \cdot [(1-\delta)(1-1/k)^{k-1} - \delta] \cdot f(OPT) \\
&\geq [(1-\delta)e^{-1} - \delta] \cdot f(OPT) \geq [e^{-1} - 2\delta] \cdot f(OPT) \enspace .
\end{aligned}$$

□

Choosing $\delta = \varepsilon/2$, the above corollary yields the approximation ratio guaranteed by Theorem 5.1. Let us now analyze the number of value oracle queries made by Algorithm 5.

Lemma 5.5. *Algorithm 5 uses $O(k^2 + n\delta^{-1}\ln(k/\delta)) = O(k^2 + n\varepsilon^{-1}\ln(k/\varepsilon))$ value oracle queries.*

Proof. The main body of Algorithm 5 uses $O(k)$ value oracle queries per iteration. Hence, it uses $O(k^2)$ value oracle queries in total. The function `FillM` uses $O(n)$ value oracle queries for every value w takes. The lemma follows by observing that the total number of values w can take is at most:

$$\lceil \ln_{1-\delta}(\delta/k) \rceil \leq 1 + \frac{\ln(k/\delta)}{-\ln(1-\delta)} \leq 1 + \frac{\ln(k/\delta)}{\delta} \enspace .$$

□

The $O(k^2)$ term in the guarantee of Lemma 5.5 stems from the fact that Algorithm 5 scans M at the end of each iteration for elements whose marginal contribution became too small. Algorithm 6 is a variant of Algorithm 5 which attempts to reduce the number of times M is scanned by first selecting a random element from M, and then scanning M only if the selected random element happens to have a small marginal contribution.

Observation 5.6. *In every given iteration, Algorithm 6 adds every element to its solution with probability at most $1/k$.*

Proof. Fix an iteration $1 \leq i \leq k$, let M'_i and w'_i be the set M at the moment the algorithm selects u'_i and the value of w at that moment. Finally, let a_i be the number of dummy elements and elements with a marginal larger than $(1-\delta)w_i$ in M'_i. The observation clearly holds for every element counted by a_i, since such an element cannot get into $\hat{M}$ on this iteration. For an element $u \in \mathcal{N}$ which is not counted by a_i to enter the solution in this iteration, two events have to happen. First, the algorithm should not select $u_i \leftarrow u'_i$, which happens with probability $(k - a_i)/k$. Second, u has to be selected from $\hat{M}$, which happens with probability at most: $1/|\hat{M}| = (k - a_i)^{-1}$ (given that the first event happened). □

Using Observation 5.6, it is possible to apply to Algorithm 6 the same analysis used above to lower bound the approximation ratio of Algorithm 5. For the analysis to work, we need to redefine some notation:

Algorithm 6: Random Lazy Greedy Improved(f, k, δ)

```
   // Initialization
1  Let M, S_0 ← ∅.
2  Let w, W ← max_{u∈N} f(u).
   // Main Loop
3  Call FillM (M).
4  for i = 1 to k do
5      Uniformly pick a random element u'_i from M.
6      if u'_i is a dummy element or f(u'_i | S) > (1 − δ)w then u_i ← u'_i.
7      else
8          foreach u ∈ M do
9              if u is not a dummy element and f(u | S) ≤ w(1 − δ) then Remove u from M.
10         Call FillM (M), and let M̂ be the set of elements added to M.
11         Uniformly pick a random element u_i from M̂.
12     Let S_i ← S_{i−1} ∪ {u_i}.
13 return S_k.
14 Function FillM (M)
15     for (w = W; w > δW/k; w ← w(1 − δ)) do
16         foreach u ∈ N do
17             if f(u | S) > w(1 − δ) then
18                 Add u to M.
19                 if |M| = k then yield.
20     do forever
21         Add k − |M| dummy elements of D \ M to M.
22         yield.
```

- M_i is a random set determined only by the random decisions of the algorithm before iteration i. Given these random decisions, M_i is the set of elements that have a positive probability (in fact $1/k$) to become u_i.
- w_i is a random value determined only by the random decisions of the algorithm before iteration i. Given these random decisions, w_i is the (unique) value that w will take if `FillM` is called during this iteration.

The following lemma completes the proof of Theorem 5.1.

Lemma 5.7. *Algorithm 6 uses, in expectation, $O(k\sqrt{n\delta^{-1}\ln(k/\delta)} + n\delta^{-1}\ln(k/\delta))$ value oracle queries.*

Proof. The function `FillM` uses $O(n)$ value oracle queries for every value w takes. Thus, it uses in total $O(n\delta^{-1}\ln(k/\delta))$ queries since the total number of values w can take is at most:

$$\lceil \ln_{1-\delta}(\delta/k) \rceil \leq 1 + \frac{\ln(k/\delta)}{-\ln(1-\delta)} \leq 1 + \frac{\ln(k/\delta)}{\delta} \ .$$

The rest of the proof bounds the expected number of value oracle queries made by the main part of Algorithm 6. For every $1 \leq i \leq k$, let X_i be the (random) number of non-dummy elements

in M at the beginning of iteration i whose marginal is $w(1-\delta)$ or less (and thus, will force the algorithm to make value oracle queries if selected as u'_i). Clearly, the main part of Algorithm 6 makes, in expectation, $O(k) \cdot \sum_{i=1}^{k} \mathbb{E}[X_i/k] = O(1) \cdot \sum_{i=1}^{k} \mathbb{E}[X_i]$ value oracle queries. On the other hand, the expected number of elements added to M in iteration i can be lower bounded by:

$$\sum_{j=0}^{k} \left[\Pr[X_i = j] \cdot \frac{j}{k} \cdot j \right] = \frac{\mathbb{E}[X_i^2]}{k} \geq \frac{(\mathbb{E}[X_i])^2}{k} \ .$$

Since the main part of Algorithm 6 never removes dummy elements from M, `FillM` might add to M up to $2k$ dummy elements and up to $O(n\delta^{-1}\ln(k/\delta))$ other elements. Thus, the following must hold:

$$\sum_{i=1}^{k} \frac{(\mathbb{E}[X_i])^2}{k} = O(n\delta^{-1}\ln(k/\delta)) \Rightarrow \sum_{i=1}^{k} \mathbb{E}[X_i] = O(k\sqrt{n\delta^{-1}\ln(k/\delta)}) \ . \qquad \square$$

6 Conclusion

We presented fast algorithms for maximizing submodular functions subject to various constraints. Our algorithm for a general matroid constraint has the interesting property that the number of value oracle queries it uses can be reduced at the cost of more independence oracle queries, and vice versa.

As far as we know, such a property did not appear in any previously known algorithm for this problem (or other related problems). Thus, it can be interesting to determine whether this unusual property represents the real nature of the problem's complexity, or is an artifact of our algorithm.

References

[1] A. A. Ageev and M. I. Sviridenko. An 0.828 approximation algorithm for the uncapacitated facility location problem. *Discrete Appl. Math.*, 93:149–156, July 1999.

[2] Per Austrin, Siavosh Benabbas, and Konstantinos Georgiou. Better balance by being biased: A 0.8776-approximation for max bisection. In *SODA*, pages 277–294, 2013.

[3] Francis Bach. Learning with submodular functions: A convex optimization perspective. *Foundations and Trends in Machine Learning*, 6(2-3):145–373, 2013.

[4] Ashwinkumar Badanidiyuru and Jan Vondrák. Fast algorithms for maximizing submodular functions. In *SODA*, pages 1497–1514, 2014.

[5] Y. Y. Boykov and M. P. Jolly. Interactive graph cuts for optimal boundary & region segmentation of objects in N-D images. In *ICCV*, volume 1, pages 105–112, 2001.

[6] Richard A. Brualdi. Comments on bases in dependence structures. *Bull. of the Australian Math. Soc.*, 1(02):161–167, 1969.

[7] Niv Buchbinder, Moran Feldman, Joseph (Seffi) Naor, and Roy Schwartz. Submodular maximization with cardinality constraints. In *SODA*, pages 1433–1452, 2014.

[8] Gruia Calinescu, Chandra Chekuri, Martin Pal, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM Journal on Computing*, 40(6):1740–1766, 2011.

[9] Chandra Chekuri and Sanjeev Khanna. A polynomial time approximation scheme for the multiple knapsack problem. *SIAM J. Comput.*, 35(3):713–728, September 2005.

[10] Chandra Chekuri, Jan Vondrák, and Rico Zenklusen. Dependent randomized rounding via exchange properties of combinatorial structures. In *FOCS*, pages 575–584, 2010.

[11] Reuven Cohen, Liran Katzir, and Danny Raz. An efficient approximation for the generalized assignment problem. *Information Processing Letters*, 100(4):162–166, 2006.

[12] G. Cornuejols, M. L. Fisher, and G. L. Nemhauser. Location of bank accounts to optimize float: an analytic study of exact and approximate algorithms. *Management Sciences*, 23:789–810, 1977.

[13] G. Cornuejols, M. L. Fisher, and G. L. Nemhauser. On the uncapacitated location problem. *Annals of Discrete Mathematics*, 1:163–177, 1977.

[14] Benjamin Doerr. Analyzing randomized search heuristics: Tools from probability theory. In Anne Auger and Benjamin Doerr, editors, *Theory of Randomized Search Heuristics*, chapter 1. World Scientific Publishing, 2011.

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

[16] Uriel Feige and Michel X. Goemans. Aproximating the value of two prover proof systems, with applications to max 2sat and max dicut. In *ISTCS*, pages 182–189, 1995.

[17] Uriel Feige, Vahab S. Mirrokni, and Jan Vondrák. Maximizing non-monotone submodular functions. *SIAM Journal on Computing*, 40(4):1133–1153, 2011.

[18] Uriel Feige and Jan Vondrák. Approximation algorithms for allocation problems: Improving the factor of $1 - 1/e$. In *FOCS*, pages 667–676, 2006.

[19] Moran Feldman, Joseph (Seffi) Naor, and Roy Schwartz. A unified continuous greedy algorithm for submodular maximization. In *FOCS*, 2011.

[20] Lisa Fleischer, Michel X. Goemans, Vahab S. Mirrokni, and Maxim Sviridenko. Tight approximation algorithms for maximum general assignment problems. In *SODA*, pages 611–620, 2006.

[21] Alan M. Frieze and Mark Jerrum. Improved approximation algorithms for max k-cut and max bisection. In *IPCO*, pages 1–13, 1995.

[22] Shayan Oveis Gharan and Jan Vondrák. Submodular maximization by simulated annealing. In *SODA*, pages 1098–1117, 2011.

[23] Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6):1115–1145, 1995.

[24] Eran Halperin and Uri Zwick. Combinatorial approximation algorithms for the maximum directed cut problem. In *SODA*, pages 1–7, 2001.

[25] Jason Hartline, Vahab Mirrokni, and Mukund Sundararajan. Optimal marketing strategies over social networks. In *WWW*, pages 189–198, 2008.

[26] Johan Hàstad. Some optimal inapproximability results. *J. ACM*, 48:798–859, July 2001.

[27] S. Jegelka and J. Bilmes. Submodularity beyond submodular energies: Coupling edges in graph cuts. *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 0:1897–1904, 2011.

[28] Richard M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[29] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *SIGKDD*, pages 137–146, 2003.

[30] Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O'Donnell. Optimal inapproximability results for max-cut and other 2-variable csps? *SIAM J. Comput.*, 37:319–357, April 2007.

[31] S. Khuller, A. Moss, and J. Naor. The budgeted maximum coverage problem. *Information Processing Letters*, 70(1):39–45, 1999.

[32] Andreas Krause, AjitSingh, and Carlos Guestrin. Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies. *J. Mach. Learn. Res.*, 9:235–284, January 2008.

[33] Andreas Krause and Carlos Guestrin. Near-optimal nonmyopic value of information in graphical models. In *UAI*, page 5, 2005.

[34] Andreas Krause, Jure Leskovec, Carlos Guestrin, Jeanne VanBriesen, and Christos Faloutsos. Efficient sensor placement optimization for securing large water distribution networks. *Journal of Water Resources Planning and Management*, 134(6):516–526, November 2008.

[35] Hui Lin and Jeff Bilmes. Multi-document summarization via budgeted maximization of submodular functions. In *North American chapter of the Association for Computational Linguistics/Human Language Technology Conference (NAACL/HLT-2010)*, Los Angeles, CA, June 2010.

[36] Hui Lin and Jeff Bilmes. A class of submodular functions for document summarization. In *HLT*, pages 510–520, 2011.

[37] G. L. Nemhauser and L. A. Wolsey. Best algorithms for approximating the maximum of a submodular set function. *Mathematics of Operations Research*, 3(3):177–188, 1978.

[38] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functionsi. *Mathematical Programming*, 14:265–294, 1978.

[39] A. Schrijver. *Combinatorial Optimization: Polyhedra and Eciency*. Springer, 2003.

[40] Luca Trevisan, Gregory B. Sorkin, Madhu Sudan, and David P. Williamson. Gadgets, approximation, and linear programming. *SIAM J. Comput.*, 29:2074–2097, April 2000.

[41] Jan Vondrák. Symmetry and approximability of submodular maximization problems. *SIAM J. Comput.*, 42(1):265–304, 2013.

A Proof of Theorem 1.3

In this section we prove the folklore result given by Theorem 1.3. Notice that for $\varepsilon \in (0, e^{-k}]$ the the standard greedy algorithm of [38] fulfills all the requirements the theorem, and for $\varepsilon \geq 1 - e^{-1}$ the theorem is void. Thus, from this point on we assume $\varepsilon \in (e^{-k}, 1 - e^{-1})$. The algorithm we use to prove Theorem 1.3 is Algorithm 4 with the parameters $s = 1$ and $p = \ln \varepsilon^{-1}/k$. Notice that $p \in (0, 1]$ since $\varepsilon > e^{-k}$. We restate Algorithm 4 with these parameter values as Algorithm 7. Since the objective function f is assumed to be monotone in Theorem 1.3, the restatement can safely omit the check on Line 6 of Algorithm 4.

Algorithm 7: Random Sampling Algorithm for Monotone Objectives(f, k)

1 Initialize: $S_0 \leftarrow \varnothing$.
2 **for** $i = 1$ **to** k **do**
3 Let M_i be a uniformly random set containing $\lceil \frac{n \cdot \ln \varepsilon^{-1}}{k} \rceil$ elements of $\mathcal{N}$.
4 Let u_i be the element of M_i with the largest marginal contribution to S_{i-1}.
5 $S_i \leftarrow S_{i-1} \cup \{u_i\}$.
6 **return** S_k.

First, the following observation follows by plugging our chosen value for p into Observation 4.1.

Observation A.1. *Algorithm 7 uses $O(n \ln \varepsilon^{-1})$ value oracle queries.*

The following lemma lower bounds the expected improvement in the solution of Algorithm 7 in every iteration.

Lemma A.2. *For every $1 \leq i \leq k$, $\mathbb{E}[f(u_i \mid S_{i-1})] \geq (1 - \varepsilon) \cdot \frac{f(OPT) - \mathbb{E}[f(S_{i-1})]}{k}$.*

Proof. Let A_i be an event specifying the random decisions of Algorithm 7 up to iteration i (excluding). If the lemma holds conditioned on every given event A_i, then it holds also unconditionally. Hence, in the rest of the proof we fix an event A_i and prove the lemma conditioned on this event. All the probabilities and expectations in the proof are implicitly conditioned on A_i. Notice that S_{i-1} is a deterministic set when conditioned on A_i.

Let $v_1, v_2, \ldots, v_k$ be the k elements with the largest marginal contributions to S_{i-1}, sorted in a non-increasing marginal contribution order. Additionally, let X_j be an indicator for the event $M_i \cap \{v_1, v_2, \ldots, v_j\} \neq \varnothing$. Using this notation, it is possible to lower bound $f(u_i \mid S_{i-1})$ as follows.

$$f(u_i \mid S_{i-1}) \geq X_k \cdot f(v_k \mid S_{i-1}) + \sum_{j=1}^{k-1} X_j \cdot [f(v_j \mid S_{i-1}) - f(v_{j+1} \mid S_{i-1})] \ .$$

On the other hand, for every $1 \leq j \leq k$, we can lower bound $\mathbb{E}[X_j]$ as follows. If $j + \lceil pn \rceil > n$, then $\mathbb{E}[X_j] = 1 \geq 1 - (1 - p)^j$. Otherwise,

$$\mathbb{E}[X_j] = 1 - \frac{\binom{n-j}{\lceil pn \rceil}}{\binom{n}{\lceil pn \rceil}} = 1 - \prod_{r=0}^{j-1} \frac{n - \lceil pn \rceil - r}{n - r} \geq 1 - \prod_{r=0}^{j-1} \frac{n - pn}{n} = 1 - (1 - p)^j \ .$$

Combining the two above observations with the linearity of the expectation, we get:

$$\begin{aligned}\mathbb{E}[f(u_i \mid S_{i-1})] &\geq \mathbb{E}[X_k] \cdot f(v_k \mid S_{i-1}) + \sum_{j=1}^{k-1} \mathbb{E}[X_j] \cdot [f(v_j \mid S_{i-1}) - f(v_{j+1} \mid S_{i-1})] \\ &\geq [1-(1-p)^k] \cdot f(v_k \mid S_{i-1}) + \sum_{j=1}^{k-1} [1-(1-p)^j] \cdot [f(v_j \mid S_{i-1}) - f(v_{j+1} \mid S_{i-1})] \\ &= p \cdot \sum_{j=1}^{k} (1-p)^{j-1} f(v_j \mid S_{i-1}) \ ,\end{aligned}$$

where the second inequality holds since $f(v_j \mid S_{i-1}) - f(v_{j+1} \mid S_{i-1})$ is always non-negative.

Consider the sum on the rightmost hand side of the above inequality. Every term of this sum is a multiplication of two non-increasing functions of j: $(1-p)^{j-1}$ and $f(v_j \mid S_{i-1})$. This allows us to use Chebyshev's sum inequality to bound this sum as follows:

$$\begin{aligned}\mathbb{E}[f(u_i \mid S_{i-1})] &\geq p \cdot \frac{\sum_{j=1}^{k} f(v_j \mid S_{i-1}) \cdot \sum_{j=1}^{k} (1-p)^{j-1}}{k} = (1-(1-p)^k) \cdot \frac{\sum_{j=1}^{k} f(v_j \mid S_{i-1})}{k} \\ &\geq (1-e^{-kp}) \cdot \frac{\sum_{j=1}^{k} f(v_j \mid S_{i-1})}{k} = (1-\varepsilon) \cdot \frac{\sum_{j=1}^{k} f(v_j \mid S_{i-1})}{k} \ .\end{aligned}$$

The lemma now follows by observing that by the definition of the v_j's and the submodularity and monotonicity of f,

$$\sum_{j=1}^{k} f(v_j \mid S_{i-1}) \geq \sum_{u \in OPT} f(u \mid S_{i-1}) \geq f(OPT \cup S_{i-1}) - f(S_{i-1}) \geq f(OPT) - f(S_{i-1}) \ . \qquad \square$$

Corollary A.3. *For every* $0 \leq i \leq k$, $\mathbb{E}[f(S_i)] \geq \left[1 - e^{-\frac{i \cdot (1-\varepsilon)}{k}}\right] \cdot f(OPT)$.

Proof. Let us denote $\alpha = k^{-1}(1-\varepsilon)$. Then, by Lemma A.2, for every $1 \leq i \leq k$,

$$\mathbb{E}[f(S_i) - f(S_{i-1})] = \mathbb{E}[f(u_i \mid S_{i-1})] \geq \alpha [f(OPT) - \mathbb{E}[f(S_{i-1})]] \ .$$

Rearranging, we get:

$$f(OPT) - \mathbb{E}[f(S_i)] \leq (1-\alpha) \cdot [f(OPT) - \mathbb{E}[f(S_{i-1})]] \ .$$

Combining the above inequalities gives:

$$f(OPT) - \mathbb{E}[f(S_i)] \leq (1-\alpha)^i \cdot [f(OPT) - \mathbb{E}[f(S_0)]] \leq (1-\alpha)^i \cdot f(OPT) \ .$$

Rearranging once more, yields:

$$\mathbb{E}[f(S_i)] \geq \left[1 - (1-\alpha)^i\right] \cdot f(OPT) \geq \left[1 - e^{-i \cdot \alpha}\right] \cdot f(OPT) \ . \qquad \square$$

The above corollary implies that $\mathbb{E}[f(S_k)] \geq (1 - e^{\varepsilon - 1}) \cdot f(OPT)$. Theorem 1.3 follows by combining this inequality with the following lemma.

Lemma A.4. $1 - e^{\varepsilon - 1} \geq 1 - e^{-1} - \varepsilon$.

Proof. Notice that:

$$1 - e^{\varepsilon - 1} \geq 1 - e^{-1} - \varepsilon \Leftrightarrow e^{\varepsilon - 1} \leq e^{-1} + \varepsilon \Leftrightarrow \varepsilon - 1 \leq \ln(e^{-1} + \varepsilon) \ .$$

By the definition of ln:

$$\ln(e^{-1} + \varepsilon) = \int_1^{e^{-1}+\varepsilon} \frac{dx}{x} = \int_1^{e^{-1}} \frac{dx}{x} + \int_{e^{-1}}^{e^{-1}+\varepsilon} \frac{dx}{x} \geq \ln e^{-1} + \varepsilon \cdot \frac{1}{e^{-1} + \varepsilon} \geq -1 + \varepsilon \ ,$$

where the last inequality holds since $\varepsilon < 1 - e^{-1}$. □

B Proof of Lemma 3.3

In this section we prove Lemma 3.3, *i.e.*, we describe a (1/3)-approximation algorithm for the problem $\max\{f(S) \mid S \in \mathcal{I}\}$ using $O(n \ln k)$ value and independence oracle queries. The algorithm we describe (given as Algorithm 8) is a close variant of an algorithm suggested by [4] for the case of a cardinality constraint. We assume in the analysis of the algorithm that $\varepsilon \in (0, 1)$.

Algorithm 8: Thresholding Greedy$(f, \mathcal{M}, \varepsilon)$

```
Let S ← ∅.
Let W, w ← max_{u∈N} f(u).
for (w ← W; w > εW/k; w ← w(1 − ε)) do
    foreach u ∈ N do
        if S ∪ {u} ∈ I and f(u | S) ≥ w then Add u to S.
return S.
```

Observation B.1. *Algorithm 8 outputs an independent set and uses $O(n\varepsilon^{-1} \ln(k/\varepsilon))$ value and independence oracle queries.*

Proof. The first part of the observation holds since Algorithm 8 does not add an element u to S unless $S \cup \{u\} \in \mathcal{I}$ before the addition. The second part of the observation follows by multiplying three values:

- Each iteration of the internal loop makes $O(1)$ queries to each oracle.
- The internal loop repeats n times.
- The number of iterations performed by the external loop is:

$$\lceil \ln_{1-\varepsilon}(\varepsilon/k) \rceil \leq 1 - \frac{\ln(k/\varepsilon)}{\ln(1-\varepsilon)} \leq 1 + \frac{\ln(k/\varepsilon)}{\varepsilon} \ .$$

□

Next, let us analyze the approximation ratio of Algorithm 8. Let ℓ be the size of the solution produced by the algorithm, and let S_i be the set S after i elements were added to it. For consistency, we also define $S_0 = \varnothing$. For every $0 \leq i \leq \ell$, let OPT_i be the maximum value independent set containing S_i. The following lemma lower bounds the gain of S_i (as a function of i) in terms of the loss of OPT_i (again, as a function of i).

Lemma B.2. *For every $1 \leq i \leq \ell$, $(1 - \varepsilon) \cdot [f(OPT_{i-1}) - f(OPT_i)] \leq f(S_i) - f(S_{i-1})$.*

Proof. Let $u_i = S_i \setminus S_{i-1}$ be the i^{th} element added by Algorithm 8, and let w_i denote the value of w in the iteration when u_i is picked by the algorithm. By the definition of the algorithm, $f(u \mid S_{i-1}) \geq w_i$. Let $u_i^* = \arg\max_{u \in OPT_{i-1} \setminus S_{i-1}} f(u \mid S_{i-1})$ be an element with the maximal marginal contribution in $OPT_{i-1} \setminus S_{i-1}$. By the definition of OPT_{i-1}, u_i^* can be added to S_{i-1}. Since u_i^* was not added before u_i, $f(u_u^* \mid S_{i-1}) \leq w_i/(1-\varepsilon)$.

If $u_i \in OPT_i$, then $f(OPT_{i-1}) = f(OPT_i)$ and the lemma clearly holds since $f(S_i) - f(S_{i-1}) \geq 0$. Otherwise, consider the set OPT_i' obtained by adding u_i to OPT_{i-1} and removing an element of $OPT_i \setminus S_i$ from the cycle created (if no cycle is created, we remove no element). Clearly OPT_i' is an independent set containing S_i. Thus,

$$f(OPT_{i-1}) - f(OPT_i) \leq f(OPT_{i-1}) - f(OPT_i') \leq \max_{u \in OPT_{i-1} \setminus S_{i-1}} f(u \mid OPT_{i-1} \setminus \{u\})$$

$$\leq \max_{u \in OPT_{i-1} \setminus S_{i-1}} f(u \mid S_{i-1}) = f(u_i^* \mid S_{i-1}) \leq \frac{w_i}{1-\varepsilon} \leq \frac{f(u_i \mid S_{i-1})}{1-\varepsilon} = \frac{f(S_i) - f(S_{i-1})}{1-\varepsilon} \ . \qquad \square$$

To get an interesting result from the last lemma, we need to show that $f(OPT_\ell)$ is not too large, *i.e.*, $f(OPT_i)$ decreases significantly as a function of i.

Lemma B.3. $f(OPT_\ell) \leq f(S_\ell) + \varepsilon \cdot f(OPT)$.

Proof. Every element of $OPT_\ell \setminus S_\ell$ can be added to S_ℓ (since S_ℓ is a subset of the independent set OPT_ℓ). Since none of them is added by Algorithm 8, we must have $f(u \mid S_\ell) \leq \varepsilon W/k$ for every $u \in OPT_\ell \setminus S_\ell$. Hence,

$$f(OPT_\ell) - f(S_\ell) \leq \sum_{u \in OPT_\ell \setminus S_\ell} f(u \mid S_\ell) \leq \sum_{u \in OPT_\ell \setminus S_\ell} \frac{\varepsilon W}{k} \leq \varepsilon W \leq \varepsilon \cdot f(OPT) \ ,$$

where the last inequality follows from the assumption that $f(u) \leq f(OPT)$ for every $u \in \mathcal{N}$. $\square$

Combining the above lemmata imply the following corollary.

Corollary B.4. *Algorithm 8 is a* $(1/2 - \varepsilon)$*-approximation algorithm for* $\max\{f(S) \mid S \in \mathcal{I}\}$.

Proof. Lemma B.2 implies:

$$(1-\varepsilon) \cdot [f(OPT_0) - f(OPT_\ell)] = (1-\varepsilon) \cdot \sum_{i=1}^{\ell} [f(OPT_{i-1}) - f(OPT_i)]$$

$$\leq \sum_{i=1}^{\ell} [f(S_i) - f(S_{i-1})] = f(S_\ell) - f(S_0) \leq f(S_\ell) \ .$$

Rearranging and using Lemma B.3 and the observation $f(OPT_0) = f(OPT)$, we get:

$$f(OPT) \leq f(OPT_\ell) + \frac{f(S_\ell)}{1-\varepsilon} \leq [f(S_\ell) + \varepsilon \cdot f(OPT)] + \frac{f(S_\ell)}{1-\varepsilon} = \varepsilon \cdot f(OPT) + \frac{2-\varepsilon}{1-\varepsilon} \cdot f(S_\ell) \ .$$

Hence,

$$f(S_\ell) \geq \frac{(1-\varepsilon)^2}{2-\varepsilon} \cdot f(OPT) \geq \frac{1-2\varepsilon}{2} \cdot f(OPT) = (1/2 - \varepsilon) \cdot f(OPT) \ . \qquad \square$$

Lemma 3.3 now follows by choosing $\varepsilon = 1/6$.