Submodular Maximization in Clean Linear Time

Wenxin Li
The Ohio State University
li.7328@osu.edu

Moran Feldman
University of Haifa
moranfe@cs.haifa.ac.il

Ehsan Kazemi
Google
ehsankazemi@google.com

Amin Karbasi
Yale University, Google Research
amin.karbasi@yale.edu.

Abstract

In this paper, we provide the first deterministic algorithm that achieves $1/2$-approximation for monotone submodular maximization subject to a knapsack constraint, while making a number of queries that scales only linearly with the size of the ground set n. Moreover, our result automatically paves the way for developing a linear-time deterministic algorithm that achieves the tight $1 - 1/e$ approximation guarantee for monotone submodular maximization under a cardinality (size) constraint. To complement our positive results, we also show strong information-theoretic lower bounds. More specifically, we show that when the maximum cardinality allowed for a solution is constant, no deterministic or randomized algorithm making a sub-linear number of function evaluations can guarantee any constant approximation ratio. Furthermore, when the constraint allows the selection of a constant fraction of the ground set, we show that any algorithm making fewer than $\Omega(n/\log(n))$ function evaluations cannot perform better than an algorithm that simply outputs a uniformly random subset of the ground set of the right size. We extend our results to the general case of maximizing a monotone submodular function subject to the intersection of a p-set system and multiple knapsack constraints. Finally, we evaluate the performance of our algorithms on multiple real-life applications, including movie recommendation, location summarization, Twitter text summarization, and video summarization.

1 Introduction

In the era of big data, one of the main challenges that we daily face is to make an informed decision while taking into account all the relevant information. The study of how to make such near-optimal decisions from a massive pool of possibilities is at the heart of combinatorial optimization. It is well-known that without further structures, finding a near-optimal solution for a combinatorial optimization problem is notoriously hard, similar to finding a global minimum of a general non-convex function. Nevertheless, the problems of practical interest, such as the ones we encounter in machine learning applications, are often much more well-behaved which makes them amenable to (exact or approximate) optimization techniques.

One celebrated combinatorial structure is submodularity, also known as diminishing returns, stating that the added value of an element (e.g., image, sensor, etc.) to a context (data set of images, sensor network, etc.) decreases as the context in which it is considered grows. Submodular functions have attracted significant interest from the machine learning community, and have led to the development of algorithms with near-optimal solutions for a wide range of applications, including outbreak detection [52], graph cuts in computer vision [40], image and video summarization [22, 32, 59, 70], active learning [19, 27, 28], compressed sensing and structured sparsity [3, 16], fairness in machine

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

learning [13, 41], recommendation [60], human brain parcellation [64], model training [57, 58], and learning causal structures [66, 75]. For recent surveys on the applications of submodular functions in machine learning and artificial intelligence, we refer the interested reader to [7, 69].

In their pioneering work, Nemhauser et al. [62] showed that a greedy algorithm achieves a tight $1 - 1/e$ approximation for *maximizing a monotone submodular function subject to cardinality constraint* (SMC). Running the greedy algorithm is too expensive as it requires $O(nk)$ function evaluations, where n is the size of the ground set and k is the maximum cardinality allowed by the constraint. Since then, there has been a significant effort to develop faster algorithms [12, 55, 72] and extend the approximation guarantee to more complex constraints (such as knapsack and p-set system) [4, 32].

In this work, we propose the first deterministic linear time algorithm for *maximizing a monotone submodular function subject to a knapsack constraints* (SMK), achieving $(1/2 - \varepsilon)$-approximation using $O_\varepsilon(n)$ function evaluations. We note that some nearly-linear time algorithms have been previously proposed for this problem. Ene and Nguyen [18] showed such an algorithm achieving $(1 - 1/e - \varepsilon)$-approximation, but the number of function evaluation used by their algorithm scales poorly with ε (namely, $O((1/\varepsilon)^{1/\varepsilon})$), which makes it impractical even for moderate values of ε. This has motivated Yaroslavtsev et al. [73] to design an algorithm called Greedy+ that achieve $(1/2 - \varepsilon)$-approximation and has a much more practical nearly-linear time complexity. Nevertheless, our algorithm manages to combine the approximation guarantee of Greedy+ with a clean linear time. For the special case of a cardinality constraint, i.e., when the elements of the ground set have the same cost, our algorithm automatically reduces to a *deterministic* linear-time algorithm that guarantees $(1 - 1/e - \varepsilon)$ approximation via $O_\varepsilon(n)$ function evaluations. This special result has also been independently obtained by [46, 47]. Previously, the state-of-the-art *deterministic* algorithm only managed to achieve this approximation ratio using $O_\varepsilon(n \log \log n)$ function evaluations [36] (albeit in their new arXiv version, [38] claimed a linear-time algorithm, but with a worse dependence on ε compared to our work).

Since the linear time complexity of our algorithm is the minimum required to read all the elements of the ground set, it is difficult to believe that any algorithm can improve over it while guaranteeing a reasonable approximation ratio. However, note that a single function evaluation can provide information about many elements of the ground set, and therefore, it is not a priori clear that an algorithm making a sub-linear number of function evaluations may not exist (even if its total time complexity is linear). Nevertheless, we are able to prove that this is indeed the case in at least two important regimes. First, we show that when the maximum cardinality allowed for the solution is constant, no algorithm making a sub-linear number of function evaluations can guarantee any constant approximation ratio for SMC. Second, when the constraint allows the selection of a constant fraction of the ground set, we show that any algorithm making fewer than $\Omega(n/\log(n))$ function evaluations cannot perform better than an algorithm that simply outputs a uniformly random subset of the ground set of the right size. Our technique also leads to a similar inapproximability result for *unconstrained submodular maximization* (USM). These lower bounds, along with the corresponding linear-time algorithms, nearly complete our understanding of the minimum query complexity of SMC, SMK, and USM problems. It is important to note that our inapproximability results are based on information theoretic arguments rather than relying on any complexity assumptions.

We consider the more general setting where we *maximize a monotone submodular function subject to a p-set system and d knapsack constraints* (SMKS). We study the trade-off between the time complexity and approximation ratio for this problem. In particular, our results improve over the state-of-the-art approximation for nearly-linear time algorithms [4]. Finally, we evaluate the performance of our algorithms using real-life and synthetic experiments, including movie recommendation, location summarization, vertex cover, twitter text summarization and video summarization.

1.1 Additional Related Work

The work of Badanidiyuru and Vondrák [4] is probably the first work whose main motivation was to find algorithms with guaranteed approximation ratios for submodular maximization problems that provably enjoy faster time complexities. This work introduced the thresholding technique, which became a central component of basically every fast deterministic implementation of the greedy algorithm considered to date, including our algorithms. In particular, using this technique, Badanidiyuru and Vondrák [4] described an $O_\varepsilon(n \log n)$ time algorithm for maximizing a monotone submodular function subject to a cardinality constraint.

Sviridenko [67] was the first to obtain $(1-1/e)$-approximation for maximizing a monotone submodular function subject to a knapsack constraint. Their original algorithm was very slow (having a time complexity of $O(n^5)$), but it can be improved to run in $\tilde{O}(n^4)$ time using the thresholding technique of [4]. The algorithm of Sviridenko [67] is based on making 3 guesses. Recently, Feldman et al. [24] showed that the optimal approximation ratio of $1-1/e$ can be obtained using only 2 guesses, leading to $\tilde{O}(n^3)$ time. They also showed that a single guess suffices to guarantee a slightly worse approximation ratio of 0.6174 and requires only $\tilde{O}(n^2)$ time [24]. There exists also an impractical nearly-linear time algorithm for the problem due to [18]. Another fast algorithm was suggested by [4], but an error was later found in their analysis (see [18] for details). We also note that multiple recent works aimed to bound the performance of a natural (and historically important) greedy algorithm [24, 49, 68]. A brief review on non-monotone submodular maximization subject to cardinality and knapsack constraints can be found in Appendix A.

When considering fast algorithms, it is important to consider also (semi-)streaming algorithms since these algorithms are usually naturally fast. Among the first semi-streaming algorithms for submodular maximization was the work of Badanidiyuru et al. [5], who obtained $(1/2-\varepsilon)$-approximation for SMC. While this work was later improved by Kazemi et al. [42] and extended to non-monotone functions [1], it was also proved that no semi-streaming algorithm can provide a better than $1/2$-approximation for this problem [23]. The more general case of a knapsack constraint was first considered in the semi-streaming setting by Yu et al. [74]. Improved algorithms for this problem were later found by [37, 39] for single pass and by [38, 73] for multiple passes.

Maximizing a monotone submodular function subject to a p-set system constraint was first studied by Fisher et al. [25], who showed that a natural greedy algorithm obtains $(p+1)^{-1}$-approximation for this problem. Many years later, Badanidiyuru and Vondrák [4] used their thresholding technique to get a nearly-linear time algorithm guaranteeing $(p+2d+1+\varepsilon)^{-1}$-approximation even when there are d knapsack constraints (in addition to the p-set system constraint). Recently, Badanidiyuru et al. [6] presented a new technique that allows them to improve over the last approximation ratio. They explicitly consider the implications of their ideas only for the regime where $d \leq p$, and in this regime they achieve $[2(p+1+\varepsilon)]^{-1}$-approximation and $(p+1+\varepsilon)^{-1}$-approximation in $\tilde{O}(nr/\varepsilon)$ and $\tilde{O}(n^3r^2/\varepsilon)$ time, respectively, where $r \leq n$ is the rank of p-set system. We note, however, that some of the ideas of [6] can also be applied to the regime $d > p$, leading to $(p+d+1)^{-1}$-approximation in $\tilde{O}(n^3)$ time.[1] One can observe that the improvement in the approximation ratio obtained by the new results of [6] comes at the cost of an increased time complexity. Our results in Section 5 further study this trade-off between the time complexity and approximation ratio and improve over the state-of-the-art approximation for nearly-linear time algorithms.

2 Notations and Preliminaries

A **set function** $f\colon 2^{\mathcal{N}} \to \mathbb{R}$ over a ground set $\mathcal{N}$ is a function that assigns a numeric value to every subset of $\mathcal{N}$. Given such a function, an element $u \in \mathcal{N}$ and a set $S \subseteq \mathcal{N}$, we denote by $f(u \mid S) \triangleq f(S \cup \{u\}) - f(S)$ the marginal contribution of u to S with respect to f. The set function f is submodular if $f(u \mid S) \geq f(u \mid T)$ for every two sets $S \subseteq T \subseteq \mathcal{N}$ and element $u \in \mathcal{N} \setminus T$, and it is monotone if $f(u \mid S) \geq 0$ for every element $u \in \mathcal{N}$ and set $S \subseteq \mathcal{N}$. All the set functions considered in this paper are also non-negative. Additionally, given two sets $S, T \in \mathcal{N}$, we define the shorthand $f(T \mid S) \triangleq f(S \cup T) - f(S)$. Furthermore, given a set S and element u, we often use $S + u$ as a shorthand for $S \cup \{u\}$.

We assume throughout the paper that algorithms are able to access the objective function f only through a value oracle that given a set $S \subseteq \mathcal{N}$ returns $f(S)$. This (standard) assumption allows us to prove inapproximability results that are based on information theoretic arguments, and therefore, are independent of any complexity assumptions.

A **set system** $\mathcal{M}$ is an ordered pair $(\mathcal{N}, \mathcal{I})$, where $\mathcal{N}$ is a ground set and $\mathcal{I}$ is a non-empty subset of $2^{\mathcal{N}}$ that obeys the following property: if $S \subseteq T \subseteq \mathcal{N}$ and $T \in \mathcal{I}$, then S also belongs to $\mathcal{I}$. It is customary to refer to the sets of $\mathcal{I}$ as independent (and to sets that do not belong to $\mathcal{I}$ as dependent). A base of a set $S \subseteq \mathcal{N}$ is a subset of S that is independent and is not included in any other set having

[1] To be more exact, to get the $(p+d+1)^{-1}$-approximation one has to apply the idea used in the proof of Theorem 8 of [6] to the $(p+2d+1+\varepsilon)^{-1}$-approximation algorithm due to [4].

these properties. The set system $\mathcal{M} = (\mathcal{N}, \mathcal{I})$ is called p-set system for an integer $p \geq 1$ if for every $S \subseteq \mathcal{N}$ the ratio between the sizes of the largest base of S and the smallest base of S is at most p.

3 Knapsack Constraint

In this section we consider the *Submodular Maximization subject to a Knapsack Constraint problem* (SMK). In this problem we are given a non-negative monotone submodular function $f\colon 2^{\mathcal{N}} \to \mathbb{R}_{\geq 0}$, a non-negative cost function $c\colon \mathcal{N} \to \mathbb{R}_{\geq 0}$ and a budget $B > 0$. We say that a set $S \subseteq \mathcal{N}$ is *feasible* if it obeys $c(S) \leq B$ (where $c(S) \triangleq \sum_{u \in S} c(u)$), and the objective of the problem is to find a set maximizing f among all feasible sets $S \subseteq \mathcal{N}$. For simplicity, we assume below that $c(u) \in (0, B]$ for every element $u \in \mathcal{N}$ and that $B = 1$. These assumptions are without loss of generality because (i) every element of $\mathcal{N}$ whose cost exceeds B can be simply removed, (ii) every element of $\mathcal{N}$ whose cost is 0 can be safely assumed to be part of the optimal solution (f is monotone), and (iii) one can scale the costs to make the budget B equal to 1.

When the knapsack constraint of SMK happens to be simply a cardinality constraint, we get a problem known as *Submodular Maximization subject to a Cardinality Constraint* (SMC). Formally, in this problem we have an integer parameter $1 \leq k \leq n$, and we are allowed to output any set whose size is at most k. One can observe that SMC is a special case of SMK in which $c(u) = 1/k$ for every element $u \in \mathcal{N}$.

The first step of our proposed solution to solve SMK and SMC problems is getting an estimate Γ of the value of an optimal solution (up to a constant factor). We show how this can be done in Appendix B. Then, in Section 3.1 we present a variant of the threshold-greedy algorithm of Badanidiyuru and Vondrák [4] that, unlike the original algorithm of [4], uses the estimate Γ to reduce the number of thresholds that need to be considered. In Section 3.2, by combining the threshold-greedy variant from Section 3.1 with a post-processing step, we show how to achieve the state-of-the-art algorithm for SMK. The theoretical guarantee of this algorithm is given by Theorem 3.1.

Theorem 3.1. *For every $\varepsilon > 0$, there exists a deterministic $(1/2 - \varepsilon)$-approximation algorithm for Submodular Maximization subject to a Knapsack Constraint (SMK) that uses $O(n\varepsilon^{-1} \log \varepsilon^{-1})$ time.*

Complementary to our result for SMK, our proposed variant of the threshold-greedy algorithm leads to a deterministic linear algorithm for SMC, which is given by the next theorem.

Theorem 3.2. *For every $\varepsilon > 0$, there exists a deterministic $(1 - 1/e - \varepsilon)$-approximation algorithm for Submodular Maximization subject to a Cardinality Constraint (SMC) using $O(n/\varepsilon)$ time.*

3.1 Fast Threshold Greedy Algorithm

In this section, we present Algorithm 1 as a variant of the threshold-greedy algorithm from [4]. Algorithm 1 uses Γ, our estimate of the optimal value (Γ is calculated by Algorithm 4 which is given in Appendix B). Algorithm 4 enables us to reduce the number of thresholds that needed to be considered. Specifically, Algorithm 1 considers an exponentially decreasing series of thresholds between $8\alpha \cdot \Gamma$ and Γ/e, where $\alpha \geq 1$ is a parameter of the algorithm. For every threshold considered, the algorithm adds to its solution every element such that (i) adding the element to the solution does not violate feasibility, and (ii) the density of the element with respect to the current solution exceeds that threshold. In addition to the parameter α, Algorithm 1 gets a quality control parameter $\varepsilon \in (0, 1)$.

The time complexity of Algorithm 1 is given by the next observation.

Observation 3.3. *The time complexity of Algorithm 1 is $O(n\varepsilon^{-1} \log \alpha)$.*

Due to space constraints, the proof of this observation and the other proofs omitted from this section can be found in Appendix C.

Let ℓ be the value of h when Algorithm 1 terminated. We analyze the approximation ratio of Algorithm 1 by describing two lower bounds on the performance of this algorithm. The following lemma gives the simpler among these bounds. Its proof is based on the observation that the elements of $T \setminus S_\ell$ were not added by Algorithm 1 to its solution even during the outer loop's last iteration.

Algorithm 1: FAST THRESHOLD GREEDY(ε, α)

Let Γ (an estimate of the value of an optimal solution) be the output of Algorithm 4.
Let $\tau \leftarrow 8\alpha \cdot \Gamma$, $h \leftarrow 0$ and $S_0 \leftarrow \varnothing$.
while $\tau > (1-\varepsilon)\Gamma/e$ **do**
  for *every element* $u \in \mathcal{N}$ **do**
    if $u \notin S_h$, $c(u) + c(S_h) \leq 1$ *and* $\frac{f(u|S_h)}{c(u)} \geq \tau$ **then**
      Let $u_{h+1} \leftarrow u$ and $S_{h+1} \leftarrow S_h + u_{h+1}$.
      Increase h by 1.
  Update $\tau \leftarrow (1-\varepsilon)\tau$.
return S_h.

Lemma 3.4. *For every set $T \subseteq \mathcal{N}$, if $\max_{u \in T} c(u) \leq 1 - c(S_\ell)$, then $f(S_\ell) \geq f(T) - \frac{c(T)}{e} \cdot f(OPT)$. In particular, $\max_{u \in OPT} c(u) \leq 1 - c(S_\ell)$ implies $f(S_\ell) \geq (1 - 1/e) \cdot f(OPT)$.*

The following lemma gives the other lower bound we need on the performance of Algorithm 1. The proof of this lemma is based on the standard argument in greedy-like algorithms, namely, the observation that adding all the elements of T to a solution increases the value of the solution to at least T, and therefore, at least one element of T is a good choice that the algorithm can choose.

Lemma 3.5. *For every set $\varnothing \neq T \subseteq \mathcal{N}$ and integer $0 \leq h < \ell$, if $\max_{u \in T} c(u) \leq 1 - c(S_h)$, then $\frac{f(S_{h+1}) - f(S_h)}{c(u_{h+1})} \geq \min\{(1-\varepsilon) \cdot \frac{f(T|S_h)}{c(T)}, \alpha \cdot f(OPT)\}$.*

Using the last two lemmata we can prove the following theorem, which implies Theorem 3.2. In a nutshell, the proof of this theorem uses Lemma 3.4 when the output of Algorithm 1 has the maximum allowed cardinality, and Lemma 3.5 otherwise.

Theorem 3.6. *If we choose $\alpha = 1$, then, for every $\varepsilon \in (0, 1)$, Algorithm 1 runs in $O(n/\varepsilon)$ time and guarantees $(1 - 1/e - \varepsilon)$-approximation for* SMC.

3.2 General Knapsack Constraint in Clean Linear Time

In this section we prove Theorem 3.1. We should point out that Algorithm 1 by itself does not guarantee any constant approximation guarantee for general knapsack constraints.[2] However, this can be fixed by using the post-processing step described by Algorithm 2. Recall that Algorithm 1 maintains a solution S_k that grows as the algorithm progresses. The post-processing step takes $O(\varepsilon^{-1} \log \varepsilon^{-1})$ snapshots of this solution at various times during the execution of Algorithm 1, and then tries to augment each one of these snapshots with a single element.

Algorithm 2: FAST THRESHOLD GREEDY + POST-PROCESSING(ε)

Execute Algorithm 1 with $\alpha = \varepsilon^{-1}$. Store all sets $S_0, S_1, \ldots, S_\ell$ produced by this execution.
for $i = 0$ **to** $\lfloor \log_{1+\varepsilon} \varepsilon^{-1} \rfloor$ **do**
  Let $S^{(i)}$ be the set S_k for the maximal k value for which $c(S_k) \leq \varepsilon(1+\varepsilon)^i$.
  if *there exists an element $u \in \mathcal{N}$ such that $c(u) \leq 1 - c(S^{(i)})$* **then**
    Let $u^{(i)}$ be an element maximizing $f(u^{(i)} \mid S^{(i)})$ among all the elements obeying the condition on the previous line.
    Let $S^{(i)+} \leftarrow S^{(i)} + u^{(i)}$.
  else Let $S^{(i)+} \leftarrow S^{(i)}$.
return *the set maximizing f in* $\{S_\ell\} \cup \{\{u\} \mid u \in \mathcal{N}\} \cup \{S^{(i)+} \mid 0 \leq i \leq \lfloor \log_{1+\varepsilon} \varepsilon^{-1} \rfloor\}$.

The time complexity of Algorithm 2 is given by the next observation.

[2] To see this, consider an input instance consisting of an element u of size $\delta > 0$, an additional element w of size 1 and a function $f(S) = 2\delta \cdot |S \cap \{u\}| + |S \cap \{w\}|$. For small enough ε and δ values, Algorithm 1 will pick the solution $\{u\}$ of value 2δ instead of the much more valuable solution $\{w\}$ whose value is 1.

Observation 3.7. *Algorithm 2 runs in $O(n\varepsilon^{-1}\log\varepsilon^{-1})$ time.*

The rest of this section is devoted to analyzing the approximation ratio of Algorithm 2. Consider the inequality

$$f(S^{(i)+}) < 1/2 \cdot f(OPT) \qquad \forall\, 0 \leq i \leq \log_{1+\varepsilon}\varepsilon^{-1} \ . \tag{1}$$

If this inequality does not hold for some integer i, then Algorithm 2 clearly obtains at least $1/2$-approximation. Therefore, it remains to consider the case in which Inequality (1) applies for every integer $0 \leq i \leq \log_{1+\varepsilon}\varepsilon^{-1}$. One can observe that this inequality implies that OPT is non-empty because $f(S^{(0)}) \geq f(\varnothing)$ by the monotonicity of f, and therefore, we can define r to be an element of OPT maximizing $c(r)$. Additionally, we assume below that r is not the only element of OPT—if this assumption does not hold, then Algorithm 2 trivially returns an optimal solution.

Lemma 3.8. *If $c(r) \geq 1-\varepsilon$, then $f(S_\ell) \geq (1/2-\varepsilon)\cdot f(OPT)$.*

If $f(\{r\}) \geq 1/2 \cdot f(OPT)$, then the last lemma follows immediately since $\{r\}$ is one of the sets considered for the output of Algorithm 2 on Line 8. Otherwise, we get that $OPT \setminus \{r\}$ is a set with a lot of value taking a very small part of the budget allowed, and the proof of Lemma 3.8 uses Lemmata 3.4 and 3.5 to argue that $f(S_\ell)$ is always large when such a set exists.

So far we have proved that the approximation ratio of Algorithm 2 is at least $1/2-\varepsilon$ when either Inequality (1) is violated or $c(r) \geq 1-\varepsilon$. The following lemma proves that the same approximation guarantee applies also when neither of these cases applies, which completes the proof of Theorem 3.1. The proof of this lemma defines i_r to be the maximal integer such that $\varepsilon(1+\varepsilon)^{i_r} \leq 1-c(r)$, and h_r to be the index for which $S_{h_r} = S^{(i_r)}$. If $h_r = \ell$, then the proof uses Lemma 3.4 to argue that $f(S_{h_r})$ is large. Otherwise, the proof uses Lemma 3.5 to argue that $f(S_{h_r+1})$ is large.

Lemma 3.9. *If $c(r) \leq 1-\varepsilon$, then Inequality* (1) *implies $f(S_\ell) \geq (1/2-\varepsilon)\cdot f(OPT)$.*

4 Information-Theoretic Inapproximability Results

In this section we present our inapproximability results. The first result that we show is inapproximability for SMC (see Section 3 for the definition of this problem) showing that, when the number k of elements allowed in the solution is constant, one cannot obtain a constant approximation ratio for this problem using $o(n)$ value oracle queries.

Theorem 4.1. *Any (possibly randomized) algorithm guaranteeing α-approximation ($\alpha \in (0,1]$) for `Submodular Maximization subject to a Cardinality Constraint (SMC)` must use $\Omega(\alpha n/k)$ value oracle queries. In particular, this implies that the algorithm must make $\Omega(n)$ value oracle queries when α and k are constants.*

See Appendix D for the proof of Theorem 4.1. Let us now consider the approximability of SMC in a different regime allowing larger values of k. Specifically, we consider the regime in which the ratio between n and k is some constant β. In this regime one can obtain β-approximation by simply outputting a uniformly random subset of $\mathcal{N}$ of size k (which requires no value oracle queries); thus, this regime cannot admit an extremely pessimistic inapproximability result like the one given by Theorem 4.1. Nevertheless, the following theorem shows that improving over the above-mentioned "easy" approximation guarantee requires a nearly-linear query complexity.

Theorem 4.2. *For any rational constant $\beta \in (0,1)$ and constant $\varepsilon > 0$, every (possibly randomized) algorithm guaranteeing approximation ratio of $\beta+\varepsilon$ for instances of `Submodular Maximization subject to a Cardinality Constraint` obeying $k = \beta n$ must use $\Omega(\frac{n}{\log n})$ value oracle queries. Moreover, this is true even when the function f of is guaranteed to be a linear function.*

We prove in Section E.1 a restricted version of Theorem 4.2 that applies only to deterministic algorithms. Then, Section E.2 extends the proof to randomized algorithms, which implies Theorem 4.2. Table 1 summarizes our knowledge about the number of value oracle queries necessary for obtaining various approximation guarantees for SMC.

Using the technique of Theorem 4.2, we can also prove another interesting result. In the *Unconstrained Submodular Maximization problem* (USM), the input consists of a non-negative (not necessarily monotone) submodular function $f\colon 2^{\mathcal{N}} \to \mathbb{R}_{\geq 0}$. The objective is to output a set $S \subseteq \mathcal{N}$ maximizing f. Our result for USM is given by the next theorem (proved in Appendix F). This theorem

Table 1: Number of value oracle queries that are required and sufficient to obtain any given approximation ratio for `Submodular Maximization subject to a Cardinality Constraint` (`SMC`). The variable ε should be understood as an arbitrarily small constant.

Regime	**Approximation Ratio Range**	**Query Complexity**
Constant k	$[\varepsilon, 1-1/e-\varepsilon]$	Can be obtained using $O(n)$ queries by Theorem 3.2 or Mirzasoleiman et al. [55], and requires $\Omega(n)$ queries by Theorem 4.1.
	$[1-1/e, 1]$	Exact solution can be obtained using $O(n^k)$ queries via brute force; $\Omega(n)$ queries are necessary by Theorem 4.1.
$k=\beta n$	$[0, \beta]$	Can be obtained by outputting a uniformly random subset of the right size (0 queries).
	$[\beta+\varepsilon, 1-1/e-\varepsilon]$	Can be obtained using $O(n)$ queries by Theorem 3.2 or Mirzasoleiman et al. [55]; requires $\Omega(\frac{n}{\log n})$ queries by Theorem 4.2.
	$1-1/e$	Obtained by Greedy using $O(nk)$ queries [62]; requires $\Omega(\frac{n}{\log n})$ queries by Theorem 4.2.
	$[1-1/e+\varepsilon, 1]$	Requires exponentially many queries [61].

Table 2: Number of value oracle queries required and sufficient to obtain any given approximation ratio for `Unconstrained Submodular Maximization` (`USM`). The variable ε should be understood as an arbitrarily small constant.

Approximation Ratio Range	**Required Query Complexity**	**Sufficient Query Complexity**
$[0, 1/4]$	0	0 An approximation ratio of $1/4$ can be obtained by outputting a uniformly random subset of $\mathcal{N}$ [20].
$[1/4+\varepsilon, 1/2]$	$\Omega(\frac{n}{\log n})$ (Theorem 4.3)	$O(n)$ Using the Double Greedy algorithm of Buchbinder et al. [11].
$[1/2+\varepsilon, 1]$	Requires an exponential number of queries [20].	–

nearly completes our understanding of the minimum query complexity necessary to obtain various approximation ratios for `USM`. See Table 2 for more detail.

Theorem 4.3. *For any constant $\varepsilon > 0$, any algorithm for `Unconstrained Submodular Maximization` that guarantees an approximation ratio of $1/4+\varepsilon$ must use $\Omega(\frac{n}{\log n})$ value oracle queries.*

The necessity of the $\log n$ term in Theorems 4.2 and 4.3 is discussed in Appendix G. Specifically, we show in this appendix an algorithm that can be used to solve (exactly) the hard problem underlying these theorems using $O(n/\log n)$ oracle queries. This algorithm is not efficient in terms of its time complexity, but it shows that one cannot prove a lower bound stronger than $\Omega(n/\log n)$ for this problem based on information theoretic arguments alone.

5 Set System and Multiple Knapsacks Constraints

In this section we consider the *Knapsacks and a Set System Constraint problem* (`SMKS`). In this problem we are given a non-negative monotone submodular function $f\colon 2^{\mathcal{N}} \to \mathbb{R}_{\geq 0}$, a p-system $\mathcal{M} = (\mathcal{N}, \mathcal{I})$, $d \geq 1$ non-negative cost functions $c_i\colon \mathcal{N} \to \mathbb{R}_{\geq 0}$ (one function for every integer $1 \leq i \leq d$) and d positive values $B_1, B_2, \ldots, B_d$ as Knapsack budgets. We say that a set $S \subseteq \mathcal{N}$ is *feasible* if it is independent in $\mathcal{M}$ and also obeys $c_i(S) \leq B_i$ for every $1 \leq i \leq d$ (where $c_i(S) \triangleq \sum_{u \in S} c_i(u)$). The objective of the problem is to find a set maximizing f among all feasible sets $S \subseteq \mathcal{N}$.

Algorithm 3 is the basic version of our proposed solution for `SMKS` (in Appendix I.1 we explain how to make this basic algorithm nearly-linear). Algorithm 3 gets two parameters. The first of

these parameters is an integer $\lambda \geq 1$. Elements that have a value larger than λ^{-1} with respect to at least one function c_i are considered big elements, and are stored in the set B of the algorithm. The other elements of $\mathcal{N}$ are considered small elements. The algorithm never considers any solution that includes both big and small elements. Instead, it creates one candidate solution S_B from the big elements, and one candidate solution from the small elements, and then outputs the better among the two (technically, in some cases the algorithm outputs directly the candidate solution based on the small elements without comparing it to S_B). The candidate solution S_B is constructed using a procedure called `BigElementsAlg` that gets the set B as input and outputs a feasible set whose value is at least $\alpha \cdot f(OPT \cap B)$, where α is a some value in $(0, 1]$ and OPT is an arbitrary optimal solution. At this point we ignore the implementation of `BigElementsAlg`, and leave the value of α unspecified. These gaps are filled in Section H.3.

Most of Algorithm 3 is devoted to constructing the candidate solution out of small elements, which we refer to as the "small elements solution". In the construction of this solution, Algorithm 3 uses its second parameter $\rho \geq 0$ that intuitively should represent the density of the small elements of OPT. The algorithm initializes the small elements solution to be empty, and then iteratively adds to it the element with the largest marginal contribution among the small elements that have two properties: (i) their addition to the solution does not make it dependent in $\mathcal{M}$, and (ii) their density (the ratio between their marginal contribution and cost according to the linear constraints) is at least ρ. This process of growing the small elements solution can end in two ways. One option is that the process ends because no additional elements can be added to the solution (in other words, no element has the two properties stated above). In this case the better among S_B and the small elements solution obtained S_k is returned. The other way in which the process of growing the small elements solution can end is when it starts violating at least one linear constraint. When this happens, Algorithm 3 uses a procedure called `SetExtract` (see Algorithm 7 in Section H.1) to get a subset of the small elements solution that is feasible and also has a good value, and this subset is returned.

Algorithm 3 assumes access to an estimate ρ of the density of the small elements of an optimal solution for the problem. In Section H.2 we explain how the dependence of the algorithm on ρ can be dropped without increasing the time complexity of the algorithm by too much. Recall that Section 1.1 demonstrated a trade-off between the time complexity and approximation guarantee of state-of-the-art algorithms for `SMKS`. The next theorem further studies this trade-off, and in particular, improves over the state-of-the-art approximation for nearly-linear time algorithms. The $\tilde{O}$ notation in this theorem suppresses factors that are poly-logarithmic in n, d and ε^{-1}. Due to space constraints, the proof of Theorem 5.1 is differed to Appendix H.

Theorem 5.1. *For every $\varepsilon > 0$, there exist algorithms that achieve $[(1 + O(\varepsilon))(p + 1 + \frac{7}{4}d)]^{-1}$-approximation and $(p + 1.5556 + \frac{13}{9}d + \varepsilon)^{-1}$ for `Submodular Maximization subject to Knapsacks and a Set System Constraint` in $\tilde{O}(nd + n/\varepsilon)$ and $\tilde{O}(n^2 + nd)$ time, respectively.*

Algorithm 3: `Basic Algorithm`(λ, ρ)

```
// Build the set of big elements, and find a solution based on them.
```
Let $B \leftarrow \{u \in \mathcal{N} \mid \exists_{1 \leq i \leq d}\, c_i(u) > \lambda^{-1}\}$.
Let S_B be the output set of `BigElementsAlg`(B).

```
// Construct a solution from the small elements.
```
Let $S_0 \leftarrow \varnothing$, $k \leftarrow 0$.
while *there exists an element $u \in \mathcal{N} \setminus (S_k \cup B)$ such that $S_k + u \in \mathcal{I}$ and $f(u \mid S_k) \geq \rho \cdot \sum_{i=1}^{d} c_i(u)$* **do**
  Let v_{k+1} be an element maximizing $f(u \mid S_k)$ among all the elements obeying the condition of the loop.
  Let $S_{k+1} \leftarrow S_k + v_{k+1}$.
  if $\max_{1 \leq i \leq d} c_i(S_{k+1}) \leq 1$ **then** Increase k by 1.
  else return *the output set of `SetExtract`(λ, S_{k+1}).*
return *the better set among S_B and S_k.*

6 Experimental Results

We have studied the submodular maximization problem subject to different constraints. In this section, we compare our proposed algorithms with the state-of-the-art algorithms under the following constraints: (i) a carnality constraint (Section 6.1), (ii) a single knapsack constraint (Section 6.2), and (iii) combination of a p-system and d knapsack constraints (Appendix J.4).

6.1 Cardinally Constraint

Cardinality constraint is the most widely studied setup in the submodular maximization literature. We compare our algorithm, FASTTHRESHOLDGREEDY (abbreviated as FTG) with LAZYGREEDY and BOOSTRATIO [46] under cardinality constraint. LAZYGREEDY is an efficient implementation of the naïve GREEDY algorithm in which the diminishing returns property of submodular functions is used to avoid oracle queries that are known to provide little gain [55]. It is well-known that LAZYGREEDY leads to several orders of magnitude speedups over GREEDY in practice.

In our first experiment, we implement a movie recommender system by finding a diverse set of movies for a user. We adopt the approach of Lindgren et al. [54] to extract features for each movie by using ratings from the MovieLens dataset [31]. For a given set of movies $\mathcal{N}$ ($|\mathcal{N}| = n$), let vector v_i represents the attributes of the i-th movie. The similarity matrix $M_{n\times n}$ between movies is defined by $M_{ij} = e^{-\lambda\cdot\text{dist}(v_i,v_j)}$, where $\text{dist}(v_i, v_j)$ is the euclidean distance between vectors $v_i, v_j \in \mathcal{N}$. For this application, we used the following monotone and submodular function to quantify the diversity of a given set of movies S: $f(S) = \log\det(\mathbf{I} + \alpha M_S)$, where $\mathbf{I}$ is the identity matrix and M_S is a square sub-matrix of M consisting of the rows and columns corresponding to the movies in the set S. Our objective is to maximize f under a cardinality constraint k. In Figure 1a, we compare the utility of the algorithms on this instance. We observe that FASTTHRESHOLDGREEDY with $\varepsilon \in \{0.1, 0.2\}$ performs as good as LAZYGREEDY, and BOOSTRATIO performs slightly worse. In Figure 1b, we observe that the query complexity of FASTTHRESHOLDGREEDY is significantly less than that of LAZYGREEDY. It is interesting to note that, as is guaranteed by our theoretical results, the number of oracle calls for FASTTHRESHOLDGREEDY is (almost) not increasing with k. In our second experiment, which appears in Appendix J.1, we consider a location summarization application and show that similar conclusions apply also to this application.

STOCHASTICGREEDY is a fast but randomized approach for solving SMC [55]. In Appendix J.2 we present an experiment comparing FASTTHRESHOLDGREEDY with STOCHASTICGREEDY under the cardinality constraint in a vertex cover context. In general, this experiment shows that the utility of FASTTHRESHOLDGREEDY is significantly better than that of STOCHASTICGREEDY, and that FASTTHRESHOLDGREEDY sometimes also outperforms STOCHASTICGREEDY in terms of the query complexity. We also observe a high variability in the utility of solutions returned by STOCHASTICGREEDY, which diminishes the benefit from this algorithm.

6.2 Single Knapsack Constraint

In this section, we evaluate the performance of Algorithm 2 (referred to as FTGP) with that of DYNAMICMRT [37] and DENSITYGREEDY under a single knapsack constraint. DENSITYGREEDY greedily adds elements that maximize the ratio between their marginal gain and knapsack cost (but ignoring elements whose addition will result in a violation of the knapsack constraint). In our first experiment for this constraint, we again consider the movie recommendation application from Section 6.1. The cost of each movie is defined to be proportional to the absolute difference between the rating of that movie and 10 (the maximum rating in iMDB). In this application, the goal is to find a diverse set of movies while guaranteeing that the total rating of the picked movies is high enough [6]. From Figures 1c and 1d, we observe that Algorithm 2 significantly outperforms the other two algorithms with respect to both the utility and number of oracle calls metrics. For our second experiment under the knapsack constraint, which appears in Appendix J.3, we consider Twitter text summarization. In this experiment we again get that Algorithm 2 surpasses the baseline algorithms.

7 Conclusion and Societal Impact

An algorithm for maximizing a submodular function is conventionally measured in terms of two quantities: a) approximation guarantee, i.e., how well it performs with respect to the (exponential time) optimum algorithm, and b) query complexity, i.e., how many function evaluations it requires.

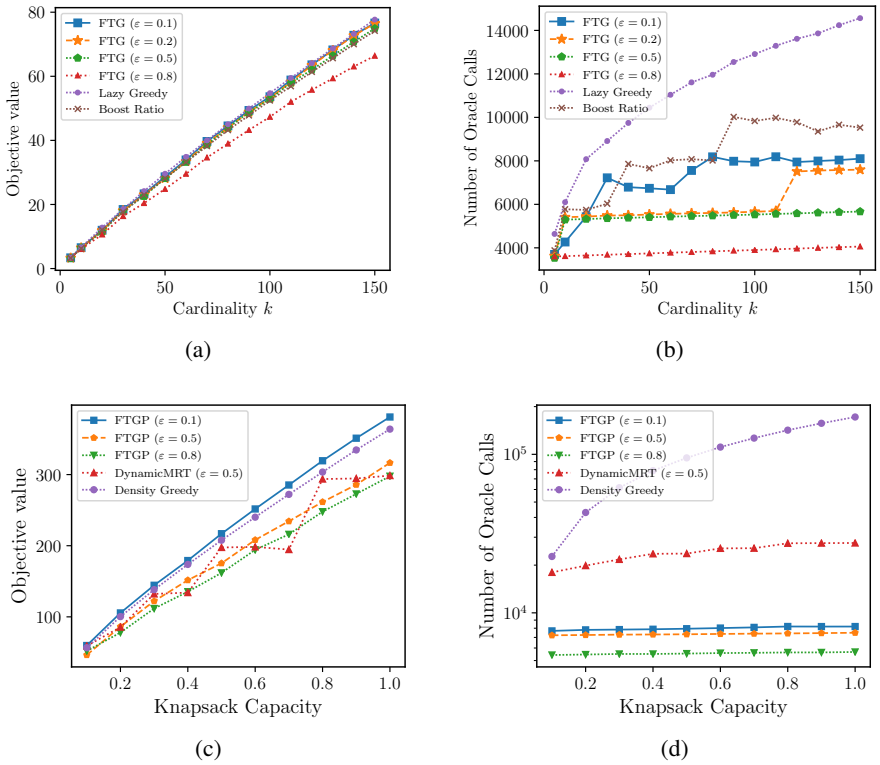


Figure 1: Movie Recommendation: (a) and (b) compare FastThresholdGreedy (referred to as FTG) with LazyGreedy and BoostRatio [46] under a cardinality constraint; (c) and (d) compare Algorithm 2 (referred to as FTGP) with DynamicMrt [37] and DensityGreedy under a single knapsack constraint.

In their seminal work, Nemhauser and Wolsey [61] and Nemhauser et al. [62] resolved part (a). In this paper, after 44 years and building on a large body of prior work, we nearly resolved part (b) and portrayed a nearly complete picture of the landscape. Specifically, we developed a clean linear-time algorithm for maximizing a monotone submodular function subject to a cardinality constraint (and more generally a knapsack constraint). We also provided information-theoretic lower bounds on the query complexity of both constrained and unconstrained (non-monotone) submodular maximization. Finally, we studied the trade-off between the time complexity and approximation ratio for maximizing a monotone submodular function subject to a p-system and d knapsack constraints. This work does not have any negative societal impact.

Like most algorithms in the literature, our algorithms assume the ability to exactly evaluate the objective function. In some applications exact evaluations is too costly, which has motivated works (such as [33]) that considered erroneous value oracles. Extending our work in this direction is an interesting question for future research.

Funding Transparency Statement

Funding in direct support of this work: The work of Moran Feldman was supported in part by Israel Science Foundation (ISF) grant no. 459/20. The work of Amin Karbasi was supported in part by the NSF (IIS-1845032), ONR (N00014- 19-1-2406), and the AI Institute for Learning-Enabled Optimization at Scale (TILOS).

References

[1] Naor Alaluf, Alina Ene, Moran Feldman, Huy L. Nguyen, and Andrew Suh. Optimal streaming algorithms for submodular maximization with cardinality constraints. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 168, pages 6:1–6:19, 2020.

[2] Georgios Amanatidis, Federico Fusco, Philip Lazos, Stefano Leonardi, and Rebecca Reiffenhäuser. Fast adaptive non-monotone submodular maximization subject to a knapsack constraint. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[3] Francis R Bach. Structured sparsity-inducing norms through submodular functions. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 118–126, 2010.

[4] Ashwinkumar Badanidiyuru and Jan Vondrák. Fast algorithms for maximizing submodular functions. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1497–1514, 2014.

[5] Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. Streaming submodular maximization: massive data summarization on the fly. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 671–680. ACM, 2014.

[6] Ashwinkumar Badanidiyuru, Amin Karbasi, Ehsan Kazemi, and Jan Vondrák. Submodular Maximization Through Barrier Functions. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[7] Jeff Bilmes. Submodularity in machine learning and artificial intelligence. *CoRR*, abs/2202.00132, 2022.

[8] Niv Buchbinder and Moran Feldman. Deterministic algorithms for submodular maximization problems. *ACM Trans. Algorithms*, 14(3):32:1–32:20, 2018.

[9] Niv Buchbinder and Moran Feldman. Constrained submodular maximization via a nonsymmetric technique. *Math. Oper. Res.*, 44(3):988–1005, 2019.

[10] Niv Buchbinder, Moran Feldman, Joseph Naor, and Roy Schwartz. Submodular maximization with cardinality constraints. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1433–1452, 2014.

[11] Niv Buchbinder, Moran Feldman, Joseph Naor, and Roy Schwartz. A tight linear time (1/2)-approximation for unconstrained submodular maximization. *SIAM J. Comput.*, 44(5):1384–1402, 2015.

[12] Niv Buchbinder, Moran Feldman, and Roy Schwartz. Comparing apples and oranges: Query trade-off in submodular maximization. *Math. Oper. Res.*, 42(2):308–329, 2017.

[13] L. Elisa Celis, Amit Deshpande, Tarun Kathuria, and Nisheeth K. Vishnoi. How to be Fair and Diverse? *CoRR*, abs/1610.07183, 2016.

[14] Vasek Chvátal. The tail of the hypergeometric distribution. *Discret. Math.*, 25(3):285–287, 1979.

[15] Sandra Eliza Fontes De Avila, Ana Paula Brandão Lopes, Antonio da Luz Jr, and Arnaldo de Albuquerque Araújo. Vsumm: A mechanism designed to produce static video summaries and a novel evaluation method. *Pattern Recognition Letters*, 32(1):56–68, 2011.

[16] Ethan R Elenberg, Rajiv Khanna, Alexandros G Dimakis, and Sahand Negahban. Restricted strong convexity implies weak submodularity. *The Annals of Statistics*, 46(6B):3539–3568, 2018.

[17] Alina Ene and Huy L. Nguyen. Constrained submodular maximization: Beyond 1/e. In *Foundations of Computer Science (FOCS)*, pages 248–257, 2016.

[18] Alina Ene and Huy L. Nguyen. A nearly-linear time algorithm for submodular maximization with a knapsack constraint. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 53:1–53:12, 2019.

[19] Hossein Esfandiari, Amin Karbasi, and Vahab Mirrokni. Adaptivity in adaptive submodularity. In *Conference on Learning Theory (COLT)*, pages 1823–1846. PMLR, 2021.

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

[21] Moran Feldman, Joseph Naor, and Roy Schwartz. A unified continuous greedy algorithm for submodular maximization. In *Foundations of Computer Science (FOCS)*, pages 570–579, 2011.

[22] Moran Feldman, Amin Karbasi, and Ehsan Kazemi. Do less, get more: Streaming submodular maximization with subsampling. *Advances in Neural Information Processing Systems (NeurIPS)*, 31, 2018.

[23] Moran Feldman, Ashkan Norouzi-Fard, Ola Svensson, and Rico Zenklusen. The one-way communication complexity of submodular maximization with applications to streaming and robustness. In *ACM Symposium on Theory of Computing (STOC)*, pages 1363–1374. ACM, 2020.

[24] Moran Feldman, Zeev Nutov, and Elad Shoham. Practical budgeted submodular maximization. *CoRR*, abs/2007.04937, 2020.

[25] Marshall L. Fisher, George L. Nemhauser, and Laurence A. Wolsey. An analysis of approximations for maximizing submodular set functions–II. *Mathematical Programming Study*, 8, 1978.

[26] Shayan Oveis Gharan and Jan Vondrák. Submodular maximization by simulated annealing. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1098–1116, 2011.

[27] Daniel Golovin and Andreas Krause. Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *Journal of Artificial Intelligence Research*, 42: 427–486, 2011.

[28] Andrew Guillory and Jeff Bilmes. Interactive Submodular Set Cover. In *International Conference on Machine Learning*, pages 415–422. Omnipress, 2010.

[29] Ran Haba, Ehsan Kazemi, Moran Feldman, and Amin Karbasi. Streaming submodular maximization under a k-set system constraint. In *International Conference on Machine Learning (ICML)*, pages 3939–3949. PMLR, 2020.

[30] Kai Han, Shuang Cui, Tianshuai Zhu, Enpei Zhang, Benwei Wu, Zhizhuo Yin, Tong Xu, Shaojie Tang, and He Huang. Approximation algorithms for submodular data summarization with a knapsack constraint. In *International Conference on Measurement and Modeling of Computer Systems*, pages 65–66, 2021.

[31] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *Acm Transactions on Interactive Intelligent Systems (TIIS)*, 5(4):1–19, 2015.

[32] Christopher Harshaw, Ehsan Kazemi, Moran Feldman, and Amin Karbasi. The power of subsampling in submodular maximization. *Mathematics of Operations Research*, 2021.

[33] Avinatan Hassidim and Yaron Singer. Submodular optimization under noise. In *Conference on Learning Theory (COLT)*, pages 1069–1122, 2017.

[34] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[35] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[36] Chien-Chung Huang and Naonori Kakimura. Multi-pass streaming algorithms for monotone submodular function maximization. *CoRR*, abs/1802.06212, 2018.

[37] Chien-Chung Huang and Naonori Kakimura. Improved streaming algorithms for maximizing monotone submodular functions under a knapsack constraint. *Algorithmica*, 83(3):879–902, 2021.

[38] Chien-Chung Huang and Naonori Kakimura. Multi-pass streaming algorithms for monotone submodular function maximization. *Theory Comput. Syst.*, 66(1):354–394, 2022.

[39] Chien-Chung Huang, Naonori Kakimura, and Yuichi Yoshida. Streaming algorithms for maximizing monotone submodular functions under a knapsack constraint. *Algorithmica*, 82(4): 1006–1032, 2020.

[40] Stefanie Jegelka and Jeff Bilmes. Submodularity beyond submodular energies: coupling edges in graph cuts. In *Computer Vision and Pattern Recognition (CVPR)*, pages 1897–1904. IEEE, 2011.

[41] Ehsan Kazemi, Morteza Zadimoghaddam, and Amin Karbasi. Scalable deletion-robust submodular maximization: Data summarization with privacy and fairness constraints. In *International conference on machine learning (ICML)*, pages 2544–2553. PMLR, 2018.

[42] Ehsan Kazemi, Marko Mitrovic, Morteza Zadimoghaddam, Silvio Lattanzi, and Amin Karbasi. Submodular streaming in all its glory: Tight approximation, minimum memory and low adaptive complexity. In *International Conference on Machine Learning (ICML)*, pages 3311–3320. PMLR, 2019.

[43] Ehsan Kazemi, Shervin Minaee, Moran Feldman, and Amin Karbasi. Regularized submodular maximization at scale. In *International Conference on Machine Learning (ICML)*, pages 5356–5366. PMLR, 2021.

[44] Andreas Krause and Daniel Golovin. Submodular Function Maximization. In *Tractability: Practical Approaches to Hard Problems*. Cambridge University Press, 2012.

[45] Alan Kuhnle. Interlaced greedy algorithm for maximization of submodular functions in nearly linear time. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2371–2381, 2019.

[46] Alan Kuhnle. Quick streaming algorithms for maximization of monotone submodular functions in linear time. In *AISTATS*, volume 130 of *Proceedings of Machine Learning Research*, pages 1360–1368. PMLR, 2021.

[47] Alan Kuhnle. Personal communication, 2022.

[48] Ariel Kulik, Hadas Shachnai, and Tami Tamir. Approximations for monotone and nonmonotone submodular maximization with knapsack constraints. *Math. Oper. Res.*, 38(4):729–739, 2013.

[49] Ariel Kulik, Roy Schwartz, and Hadas Shachnai. A faster tight approximation for submodular maximization subject to a knapsack constraint. *CoRR*, abs/2102.12879, 2021.

[50] Jon Lee, Vahab S. Mirrokni, Viswanath Nagarajan, and Maxim Sviridenko. Maximizing nonmonotone submodular functions under matroid or knapsack constraints. *SIAM J. Discret. Math.*, 23(4):2053–2078, 2010.

[51] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford Large Network Dataset Collection. `http://snap.stanford.edu/data`, June 2014.

[52] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. Cost-effective outbreak detection in networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD)*, pages 420–429. ACM, 2007.

[53] Vsevolod F. Lev and Raphael Yuster. On the size of dissociated bases. *The Electronic Journal of Combinatorics*, 18(1), 2011.

[54] Erik M Lindgren, Shanshan Wu, and Alexandros G Dimakis. Sparse and greedy: Sparsifying submodular facility location problems. In *NeuIPS Workshop on Optimization for Machine Learning*, 2015.

[55] Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, Amin Karbasi, Jan Vondrák, and Andreas Krause. Lazier than lazy greedy. In Blai Bonet and Sven Koenig, editors, *Conference on Artificial Intelligence (AAAI)*, pages 1812–1818. AAAI Press, 2015.

[56] Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, and Amin Karbasi. Fast constrained submodular maximization: Personalized data summarization. In *International Conference on Machine Learning (ICML)*, pages 1358–1367, 2016.

[57] Baharan Mirzasoleiman, Amin Karbasi, Rik Sarkar, and Andreas Krause. Distributed submodular maximization. *The Journal of Machine Learning Research*, 17(1):8330–8373, 2016.

[58] Baharan Mirzasoleiman, Jeff Bilmes, and Jure Leskovec. Coresets for data-efficient training of machine learning models. In *International Conference on Machine Learning (ICML)*, pages 6950–6960. PMLR, 2020.

[59] Marko Mitrovic, Ehsan Kazemi, Morteza Zadimoghaddam, and Amin Karbasi. Data summarization at scale: A two-stage submodular approach. In *International Conference on Machine Learning (ICML)*, pages 3596–3605. PMLR, 2018.

[60] Marko Mitrovic, Ehsan Kazemi, Moran Feldman, Andreas Krause, and Amin Karbasi. Adaptive sequence submodularity. *Advances in Neural Information Processing Systems (NeurIPS)*, 32, 2019.

[61] 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.

[62] 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.

[63] Shinsaku Sakaue. Guarantees of stochastic greedy algorithms for non-monotone submodular maximization with cardinality constraint. In *International Conference on Artificial Intelligence and Statistics, (AISTATS)*, pages 11–21, 2020.

[64] Mehraveh Salehi, Amin Karbasi, Dustin Scheinost, and R Todd Constable. A submodular approach to create individualized parcellations of the human brain. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 478–485. Springer, 2017.

[65] Matthew Skala. Hypergeometric tail inequalities: ending the insanity. *CoRR*, abs/1311.5939, 2013.

[66] Scott Sussex, Caroline Uhler, and Andreas Krause. Near-optimal multi-perturbation experimental design for causal structure learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 34, 2021.

[67] Maxim Sviridenko. A note on maximizing a submodular set function subject to a knapsack constraint. *Oper. Res. Lett.*, 32(1):41–43, 2004.

[68] Jing Tang, Xueyan Tang, Andrew Lim, Kai Han, Chongshou Li, and Junsong Yuan. Revisiting modified greedy algorithm for monotone submodular maximization with a knapsack constraint. *Proc. ACM Meas. Anal. Comput. Syst.*, 5(1):08:1–08:22, 2021.

[69] Ehsan Tohidi, Rouhollah Amiri, Mario Coutino, David Gesbert, Geert Leus, and Amin Karbasi. Submodularity in action: From machine learning to signal processing applications. *IEEE Signal Processing Magazine*, 37(5):120–133, 2020.

[70] Sebastian Tschiatschek, Rishabh K Iyer, Haochen Wei, and Jeff A Bilmes. Learning mixtures of submodular functions for image collection summarization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2014.

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

[72] Kai Wei, Rishabh Iyer, and Jeff Bilmes. Fast multi-stage submodular maximization. In *International conference on machine learning (ICML)*, pages 1494–1502. PMLR, 2014.

[73] Grigory Yaroslavtsev, Samson Zhou, and Dmitrii Avdiukhin. ”bring your own greedy”+max: Near-optimal 1/2-approximations for submodular knapsack. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 3263–3274, 2020.

[74] Qilian Yu, Easton Li Xu, and Shuguang Cui. Streaming algorithms for news and scientific literature recommendation: Monotone submodular maximization with a d-knapsack constraint. *IEEE Access*, 6:53736–53747, 2018.

[75] Yuxun Zhou and Costas J Spanos. Causal meets submodular: Subset selection with directed information. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2649–2657, 2016.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] There is no risk of negative societal impacts form our work as we only solve the a previously well-studied set of problems faster.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes] It is not applicable.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) It is not applicable.

A Non-monotone Submodular Maximization: Literature Review

Maximization of non-monotone submodular functions subject to a cardinality constraint is a special case of the maximization of such functions subject to matroid constraints, and many works [17, 21, 26, 50, 71] on the problem considered this more general setting, culminating in 0.385-approximation due to Buchbinder and Feldman [9]. As the above algorithms (especially the ones obtaining better approximation ratios) are quite involved and non-combinatorial, Buchbinder et al. [10] proposed the random greedy algorithm, which is a combinatorial algorithm achieving an approximation ratio of $1/e \approx 0.367$ using $O(nk)$ queries (matching the state-of-the-art approximation ratio for general matroid constraints at the time). Furthermore, by derandomizing random greedy, Buchbinder and Feldman [8] obtained a $1/e$-approximation deterministic algorithm. To improve the computational speed, Buchbinder et al. [12] proposed an $(1/e-\varepsilon)$-approximation random sampling algorithm that makes $O(\frac{n}{\varepsilon^2}\log\frac{1}{\varepsilon})$ queries, and another algorithm that achieves the same approximation guarantee using $O(\frac{n}{\varepsilon}\ln\frac{k}{\varepsilon})$ queries when $k = O(\sqrt{n})$. Sakaue [63] showed that, with slight modification, a well-known algorithm called "stochastic greedy" [12, 55] can achieve almost $1/4$-approximation in expectation in linear time. Additionally, based on the idea of interlacing two greedy procedures, Kuhnle [45] developed a $(1/4-\varepsilon)$-approximation deterministic algorithm requiring $O(\frac{n}{\varepsilon}\log\frac{k}{\varepsilon})$ queries.

The state-of-the-art algorithm for maximizing a non-monotone submodular function subject to a knapsack constraint is 0.385-approximation that can be obtained by combining the algorithm of Buchbinder and Feldman [9] for optimization of the multilinear extension with the rounding scheme of Kulik et al. [48]. Unfortunately, this state-of-the-art algorithm is randomized and quite slow, which has motivated works on deterministic and fast algorithms for the problem. Mirzasoleiman et al. [56] proposed a deterministic $(1/10-\varepsilon)$-approximation algorithm using $O(\frac{n^2\log n}{\varepsilon})$ queries; and the currently best deterministic algorithm is due to Han et al. [30], who achieve an approximation ratio of $(1/6-\varepsilon)$ via $O(\frac{n}{\varepsilon}\log\frac{k}{\varepsilon})$ queries. If one is willing to allow randomization, then a $1/(3+2\sqrt{2})$-approximation algorithm was propose by Amanatidis et al. [2], which requires $O(\frac{n}{\varepsilon}\log\frac{n}{\varepsilon})$ queries. This was later improved by Han et al. [30], who showed that an approximation ratio of $1/4$ is obtainable with $O(\frac{n}{\varepsilon}\log\frac{k}{\varepsilon})$ queries.

B Estimating the Value of an Optimal Solution

In this section our objective is to design an algorithm (given as Algorithm 4) that given an instance of SMK produces a value Γ that obeys $\Gamma \leq f(OPT) \leq s \cdot \Gamma$ for some constant $s \geq 1$, where OPT is an arbitrary optimal solution.

Algorithm 4: ESTIMATING $f(OPT)$

```
Let S ← ∅.
for every element u ∈ N do
    if f(u|S)/c(u) ≥ f(S) then
        Add u to S.
return f(S)/4.
```

Clearly Algorithm 4 can be implemented to work in $O(n)$ time. Therefore, we concentrate on bounding the quality of the estimate it produces. The following observation shows that Algorithm 4 does not underestimate OPT by more than a constant factor. Let $\tilde{S}$ denote the final value of the set S in Algorithm 4.

Observation B.1. $f(\tilde{S}) \geq f(OPT)/2$.

Proof. Consider an arbitrary element $u \in OPT \setminus \tilde{S}$. The fact that u was never added to S by Algorithm 4 implies that at the time in which u was processed by Algorithm 4 the following inequality held

$$\frac{f(u \mid S)}{c(u)} \leq f(S) \ .$$

Since elements are only ever added to S, by the monotonicity and submodularity of f, the last inequality implies

$$\frac{f(u \mid \tilde{S})}{c(u)} \leq f(\tilde{S}) \ .$$

Using the last inequality, we now get

$$f(OPT \mid \tilde{S}) \leq \sum_{u \in OPT \setminus \tilde{S}} f(u \mid \tilde{S}) \leq f(\tilde{S}) \cdot \sum_{u \in OPT \setminus \tilde{S}} c(u) = f(\tilde{S}) \cdot c(OPT \setminus \tilde{S}) \leq f(\tilde{S}) \ ,$$

where the first inequality follows from the submodularity of f, and the last inequality holds because OPT is a feasible solution and $OPT \setminus \tilde{S}$ is a subset of it. Plugging the definition of $f(OPT \mid \tilde{S})$ into the last inequality, and rearranging, yields

$$f(\tilde{S}) \geq \frac{f(OPT \cup \tilde{S})}{2} \geq \frac{f(OPT)}{2} \ ,$$

where the second inequality follows from the monotonicity of f. □

To prove that Algorithm 4 also does not over estimate $f(OPT)$, we need to define some additional notation. From this point on, it will be convenient to assume that the elements of $\mathcal{N}$ are ordered in the order in which Algorithm 4 considers them. Given a set $T \subseteq \mathcal{N}$, let $T^{\geq 1}$ be the minimal suffix of T (according to the above order) whose value according to c is at least 1 (unless $c(T) < 1$, in which case $T^{\geq 1} = T$). Clearly $f(\tilde{S}^{\geq 1}) \leq f(\tilde{S})$ by the monotonicity of f. However, it turns out that $f(\tilde{S}^{\geq 1})$ is never much smaller than $f(\tilde{S})$.

Lemma B.2. $f(\tilde{S}^{\geq 1}) \geq f(\tilde{S})/2$.

Proof. If $c(\tilde{S}^{\geq 1}) < 1$, then $\tilde{S}^{\geq 1} = \tilde{S}$, which makes the lemma trivial. Therefore, we may assume in the rest of the proof that $c(\tilde{S}^{\geq 1}) \geq 1$.

The fact that an element $u \in \tilde{S}^{\geq 1}$ was added to S by Algorithm 4 implies that if, we denote by S_u the set S immediately before u was processed, then we have the following inequality.

$$\frac{f(u \mid S_u)}{c(u)} \geq f(S_u) \ .$$

Since $\tilde{S}^{\geq 1}$ is a suffix of $\tilde{S}$, this inequality implies

$$f(\tilde{S}^{\geq 1} \mid \tilde{S} \setminus \tilde{S}^{\geq 1}) = \sum_{u \in \tilde{S}^{\geq 1}} f(u \mid S_u) \geq \sum_{u \in \tilde{S}^{\geq 1}} c(u) \cdot f(S_u) \geq c(\tilde{S}^{\geq 1}) \cdot f(\tilde{S} \setminus \tilde{S}^{\geq 1}) \geq f(\tilde{S} \setminus \tilde{S}^{\geq 1}) \ ,$$

where the penultimate inequality holds by the monotonicity of f, and the last inequality follows from our assumption that $c(\tilde{S}^{\geq 1}) \geq 1$.

Plugging the definition of $f(\tilde{S}^{\geq 1} \mid \tilde{S} \setminus \tilde{S}^{\geq 1})$ into the last inequality gives

$$f(\tilde{S}) \geq 2f(\tilde{S} \setminus \tilde{S}^{\geq 1}) \geq 2[f(\tilde{S}) - f(\tilde{S}^{\geq 1})] \ ,$$

where the second inequality holds by the submodularity (and non-negativity) of f. The lemma now follows by rearranging the last inequality. □

Corollary B.3. $f(\tilde{S}) \leq 4 \cdot f(OPT)$.

Proof. Since $\tilde{S}^{\geq 1}$ is the minimal suffix of $\tilde{S}$ whose size according to c is at least 1, if we denote by u the first element of this suffix, then both $\{u\}$ and $\tilde{S}^{\geq 1} - u$ are feasible solutions. Hence,

$$f(OPT) \geq \max\{f(\{u\}), f(\tilde{S}^{\geq 1} - u)\} \geq \frac{f(\{u\}) + f(\tilde{S}^{\geq 1} - u)}{2} \geq \frac{f(\tilde{S}^{\geq 1})}{2} \geq \frac{f(\tilde{S})}{4} \ ,$$

where the penultimate inequality follows from the submodularity of f, and the last inequality follows from Lemma B.2. The corollary now follows by multiplying the last inequality by 4. □

Recall now that the output of Algorithm 4 is $f(\tilde{S})/4$. Therefore, Observation B.1 and Corollary B.3 imply together the following proposition.

Proposition B.4. *Algorithm 4 runs in $O(n)$ time, and if we denote by Γ its output, then $\Gamma \leq f(OPT) \leq 8 \cdot \Gamma$.*

C Omitted Proofs of Section 3

Observation 3.3. *The time complexity of Algorithm 1 is* $O(n\varepsilon^{-1}\log\alpha)$.

Proof. Recall that Algorithm 4 runs in $O(n)$ time by Proposition B.4, and observe that every iteration of the outer loop of Algorithm 1 runs in $O(n)$ time. Therefore, to prove the lemma, it suffices to argue that the number of iterations made by this outer loop is $O(\varepsilon^{-1}\log\alpha)$. Formally, the number of iterations made by this loop is upper bounded by

$$2+\log_{(1-\varepsilon)^{-1}}\left(\frac{8\alpha\cdot\Gamma}{(1-\varepsilon)\Gamma/e}\right)=3-\frac{\ln(8\alpha\cdot e)}{\ln(1-\varepsilon)}\leq 3+\frac{4+\ln\alpha}{\varepsilon}=O(\varepsilon^{-1}\log\alpha)\ . \qquad \square$$

Lemma 3.4. *For every set* $T\subseteq\mathcal{N}$*, if* $\max_{u\in T}c(u)\leq 1-c(S_\ell)$*, then* $f(S_\ell)\geq f(T)-\frac{c(T)}{e}\cdot f(OPT)$*. In particular,* $\max_{u\in OPT}c(u)\leq 1-c(S_\ell)$ *implies* $f(S_\ell)\geq(1-1/e)\cdot f(OPT)$*.*

Proof. The second part of the lemma follows from the first part by plugging $T=OPT$ since OPT, as a feasible set, obeys $c(OPT)\leq 1$. Therefore, we concentrate below on proving the first part of the lemma.

Since elements are only ever added by Algorithm 1 to its solution set S_k, the condition of the lemma implies that whenever an element $u\in T\setminus S_\ell$ was considered on Line 5 of Algorithm 1, its density was too low compared to the value of the threshold τ at time. In particular, this is true for the threshold $\tau\leq\Gamma/e$ in the last iteration of the outer loop of Algorithm 1, and therefore,

$$\frac{f(u\mid S_u)}{c(u)}\leq\frac{\Gamma}{e}\qquad\forall\,u\in T\setminus S_\ell\ ,$$

where S_u represents here the set S_k immediately before u is processed by Algorithm 1 in the last iteration of its outer loop.

By the submodularity of f, the last inequality holds also when S_u is replaced with S_ℓ, and therefore,

$$f(T)\leq f(T\cup S_\ell)\leq f(S_\ell)+\sum_{u\in T\setminus S_\ell}f(u\mid S_\ell)\leq f(S_\ell)+\frac{\Gamma}{e}\cdot c(T\setminus S_\ell)\leq f(S_\ell)+\frac{c(T)\cdot\Gamma}{e}\ ,$$

where the first inequality follows from the monotonicity of f, and the second inequality follows from the submodularity of f. The lemma now follows from the last inequality by recalling that Proposition B.4 guarantees $\Gamma\leq f(OPT)$. $\square$

Lemma 3.5. *For every set* $\varnothing\neq T\subseteq\mathcal{N}$ *and integer* $0\leq h<\ell$*, if* $\max_{u\in T}c(u)\leq 1-c(S_h)$*, then* $\frac{f(S_{h+1})-f(S_h)}{c(u_{h+1})}\geq\min\{(1-\varepsilon)\cdot\frac{f(T|S_h)}{c(T)},\alpha\cdot f(OPT)\}$.

Proof. Let τ_{h+1} be the value of τ when Algorithm 1 selects u_{h+1}. If $\tau_{h+1}=8\alpha\cdot\Gamma\geq\alpha\cdot f(OPT)$, then the lemma follows immediately from the condition used on Line 5 of the algorithm. Therefore, we assume below that $\tau_{h+1}<8\alpha\cdot\Gamma$, which implies that no element of $T\setminus S_h$ was added to the solution of Algorithm 1 when the threshold τ was equal to $(1-\varepsilon)^{-1}\tau_{h+1}$. Therefore, since elements are only ever added by Algorithm 1 to its solution, the submodularity of f guarantees

$$\frac{f(u\mid S_h)}{c(u)}<\frac{\tau_{h+1}}{1-\varepsilon}\qquad\forall\,u\in T\setminus S_h\ .$$

We now observe that the lemma follows immediately from the monotonicity of f when $T\subseteq S_h$. Therefore, we need to consider only the case of $T\not\subseteq S_h$, and in this case the previous inequality implies

$$\begin{aligned}\frac{f(S_{h+1})-f(S_h)}{c(u_{h+1})}&=\frac{f(u_{h+1}\mid S_h)}{c(u_{h+1})}\geq\tau_{h+1}=\frac{\sum_{u\in T\setminus S_h}c(u)\cdot\tau_{h+1}}{c(T\setminus S_h)}\\&>(1-\varepsilon)\cdot\frac{\sum_{u\in T\setminus S_h}f(u\mid S_h)}{c(T\setminus S_h)}\geq(1-\varepsilon)\cdot\frac{f(T\mid S_h)}{c(T)}\ ,\end{aligned}$$

where the last inequality follows from the submodularity of f. $\square$

Theorem 3.6. *If we choose $\alpha = 1$, then, for every $\varepsilon \in (0,1)$, Algorithm 1 runs in $O(n/\varepsilon)$ time and guarantees $(1 - 1/e - \varepsilon)$-approximation for* SMC.

Proof. The time complexity of Algorithm 1 was already proved in Observation 3.3. Therefore, we concentrate here on analyzing the approximation guarantee of this algorithm. Let k be the maximum cardinality allowed by the cardinality constraint (*i.e.*, $c(u) = 1/k$ for every $u \in \mathcal{N}$). We need to consider two cases depending on the value of ℓ. If $\ell < k$, *i.e.*, Algorithm 1 outputs a solution that is smaller than the maximum cardinality allowed, then the condition $\min_{u \in OPT} c(u) \leq 1 - c(S_\ell)$ of Lemma 3.4 holds (because $c(S_\ell) = \ell/k \leq 1 - 1/k$), and therefore,

$$f(S_\ell) \geq (1 - 1/e) \cdot f(OPT) \ .$$

Assume now that $\ell = k$. In this case we need to use Lemma 3.5. Since the condition of this lemma holds for every integer $0 \leq h < \ell$ and $T = OPT$ (we may assume that $OPT \neq \varnothing$ because otherwise Algorithm 1 is clearly optimal), the lemma implies, for every such k,

$$\frac{f(S_{h+1}) - f(S_h)}{c(u_{h+1})} \geq \min\left\{(1-\varepsilon) \cdot \frac{f(OPT \mid S_h)}{c(OPT)}, f(OPT)\right\} \geq (1-\varepsilon) \cdot f(OPT \mid S_h) \ ,$$

where the second inequality holds because the feasibility of OPT implies $c(OPT) \leq 1$, and the submodularity and non-negativity of f imply $f(OPT \mid S_h) \leq f(OPT \mid \varnothing) \leq f(OPT)$. Rearranging the last inequality yields

$$\begin{aligned}&[1 - (1-\varepsilon) \cdot c(u_{h+1})] \cdot [f(OPT) - f(S_h)] \\ &\qquad \geq [1 - (1-\varepsilon) \cdot c(u_{h+1})] \cdot f(OPT) + (1-\varepsilon) \cdot c(u_{h+1}) \cdot f(OPT \cup S_h) - f(S_{h+1}) \ ,\end{aligned}$$

which by the monotonicity of f, the observation that $f(S_h) \leq f(OPT)$ because S_h is a feasible solution, and the inequality $1 - x \leq e^{-x}$ (which holds for every $x \in \mathbb{R}$) implies

$$e^{-(1-\varepsilon) \cdot c(u_{k+1})} \cdot [f(OPT) - f(S_k)] \geq f(OPT) - f(S_{k+1}) \ .$$

Finally, by combining the last inequality for all integers $0 \leq h < \ell$, we get

$$\begin{aligned}f(OPT) - f(S_\ell) &\leq e^{-(1-\varepsilon) \cdot \sum_{0 \leq h < \ell} c(u_{h+1})} \cdot [f(OPT) - f(S_0)] \\ &= e^{-(1-\varepsilon)} \cdot [f(OPT) - f(S_0)] \leq e^{-(1-\varepsilon)} \cdot f(OPT) \leq (e^{-1} + \varepsilon) \cdot f(OPT) \ ,\end{aligned}$$

where the equality holds since $\sum_{0 \leq h < \ell} c(u_{h+1}) = c(S_\ell) = 1$ by the definition of the case we consider, and the penultimate inequality follows from the non-negativity of f. The theorem now follows by rearranging the last inequality. □

Observation 3.7. *Algorithm 2 runs in $O(n\varepsilon^{-1} \log \varepsilon^{-1})$ time.*

Proof. Algorithm 1 runs in $O(n\varepsilon^{-1} \log \varepsilon^{-1})$ time by Observation 3.3. Additionally, every iteration of the loop of Algorithm 2 runs in $O(n + \ell) = O(n)$ time, where the equality holds because the fact that some element is added to the solution of Algorithm 1 whenever the index k increases guarantees that ℓ is at most n. Therefore, to prove the observation it suffices to show that the loop of Algorithm 2 iterates only $O(\varepsilon^{-1} \log \varepsilon^{-1})$ times. Formally, the number of iterations of this loop is upper bounded by

$$1 + \log_{1+\varepsilon} \varepsilon^{-1} = 1 + \frac{\ln \varepsilon^{-1}}{\ln(1+\varepsilon)} \leq 1 + \frac{\ln \varepsilon^{-1}}{\varepsilon/2} = O(\varepsilon^{-1} \ln \varepsilon^{-1}) \ .$$ □

Lemma 3.8. *If $c(r) \geq 1 - \varepsilon$, then $f(S_\ell) \geq (1/2 - \varepsilon) \cdot f(OPT)$.*

Proof. If $f(\{r\}) \geq 1/2 \cdot f(OPT)$, then the lemma follows immediately since $\{r\}$ is one of the sets considered for the output of Algorithm 2 on Line 8. Therefore, we may assume that $f(\{r\}) \leq 1/2 \cdot f(OPT)$, which implies $f(OPT - r) \geq f(OPT) - f(\{r\}) \geq 1/2 \cdot f(OPT)$ by the submodularity of f. Since $c(OPT) \leq 1$ because OPT is a feasible set, we get that $OPT - r$ is a set with a lot of value taking a very small part of the budget allowed (specifically, $c(OPT - r) = c(OPT) - c(r) \leq 1 - (1 - \varepsilon) = \varepsilon$). Below we show that the existence of such a set implies that $f(S_\ell)$ is large. We also assume below $f(S_\ell) \leq f(OPT - r)$ because otherwise the lemma follows immediately since $f(OPT - r) \geq 1/2 \cdot f(OPT)$.

Consider first the case in which $c(S_\ell) \leq 1-\varepsilon$. Since $\max_{u \in OPT-r} c(u) \leq c(OPT-r) \leq \varepsilon \leq 1 - c(S_\ell)$ in this case, Lemma 3.4 gives us immediately, for $T = OPT - r$,

$$f(S_\ell) \geq f(OPT-r) - \frac{c(OPT-r)}{e} \cdot f(OPT) \geq \frac{f(OPT)}{2} - \frac{\varepsilon \cdot f(OPT)}{e} \geq (1/2 - \varepsilon) \cdot f(OPT) \ .$$

It remains to consider the case of $c(S_\ell) \geq 1-\varepsilon$. Let $\bar{h}$ be the minimal h such that $c(S_h) \geq 1-\varepsilon$. For every integer $0 \leq h < \bar{h}$, we have $\max_{u \in OPT-r} \leq c(OPT-r) \leq \varepsilon \leq 1 - c(S_h)$, and therefore, applying Lemma 3.5 with $T = OPT - r$ yields

$$\begin{aligned}\frac{f(S_{h+1}) - f(S_h)}{c(u_{h+1})} &\geq \min\left\{(1-\varepsilon) \cdot \frac{f(OPT - r \mid S_h)}{c(OPT-r)}, \varepsilon^{-1} \cdot f(OPT)\right\} \\ &\geq \frac{1-\varepsilon}{\varepsilon} \cdot f(OPT - r \mid S_h) \ ,\end{aligned}$$

where the second inequality holds because $c(OPT-r) \leq \varepsilon$, and the monotonicity, submodularity and non-negativity of f imply $f(OPT - r \mid S_h) \leq f(OPT \mid S_h) \leq f(OPT \mid \varnothing) \leq f(OPT)$. Rearranging the last inequality yields

$$\begin{aligned}&\left[1 - \frac{1-\varepsilon}{\varepsilon} \cdot c(u_{h+1})\right] \cdot [f(OPT-r) - f(S_h)] \\ &\quad \geq \left[1 - \frac{1-\varepsilon}{\varepsilon} \cdot c(u_{h+1})\right] \cdot f(OPT-r) + \frac{1-\varepsilon}{\varepsilon} \cdot c(u_{h+1}) \cdot f((OPT-r) \cup S_h) - f(S_{h+1}) \ ,\end{aligned}$$

which by the monotonicity of f, our assumption that $f(S_h) \leq f(S_\ell) \leq f(OPT-r)$ and the inequality $1 - x \leq e^{-x}$ (which holds for every $x \in \mathbb{R}$) implies

$$e^{-\varepsilon^{-1}(1-\varepsilon) \cdot c(u_{h+1})} \cdot [f(OPT-r) - f(S_h)] \geq f(OPT-r) - f(S_{h+1}) \ .$$

Finally, by combining the last inequality for all integers $0 \leq h < \bar{h}$, we get

$$\begin{aligned}f(OPT-r) - f(S_{\bar{h}}) &\leq e^{-\varepsilon^{-1}(1-\varepsilon) \cdot \sum_{0 \leq h < \bar{h}} c(u_{h+1})} \cdot [f(OPT-r) - f(S_0)] \\ &\leq e^{-\varepsilon^{-1}(1-\varepsilon)^2} \cdot [f(OPT-r) - f(S_0)] \\ &\leq e^{-\varepsilon^{-1}(1-\varepsilon)^2} \cdot f(OPT-r) \leq 2\varepsilon \cdot f(OPT-r) \ ,\end{aligned}$$

where the second inequality holds since $\sum_{0 \leq h < \bar{h}} c(u_{h+1}) = c(S_{\bar{h}}) \geq 1 - \varepsilon$ by the definition of $\bar{h}$ and the penultimate inequality follows from the non-negativity of f.

Rearranging the last inequality, we now get, using the monotonicity of f,

$$f(S_\ell) \geq f(S_{\bar{h}}) \geq (1 - 2\varepsilon) \cdot f(OPT - r) \geq (1/2 - \varepsilon) \cdot f(OPT) \ . \qquad \square$$

Lemma 3.9. *If $c(r) \leq 1 - \varepsilon$, then Inequality* (1) *implies $f(S_\ell) \geq (1/2 - \varepsilon) \cdot f(OPT)$.*

Proof. Let i_r be the maximal integer such that $\varepsilon(1+\varepsilon)^{i_r} \leq 1 - c(r)$. Since we assume that $c(r) \leq 1 - \varepsilon$, i_r is non-negative. On the other hand, we also have $i_r \leq \log_{1+\varepsilon} \varepsilon^{-1}$ because otherwise $\varepsilon(1+\varepsilon)^{i_r} > 1 \geq 1 - c(r)$. Therefore, the set $S^{(i_r)}$ is well-defined, and one can observe that

$$f(S^{(i_r)} + r) = f(S^{(i_r)}) + f(r \mid S^{(i_r)}) \leq f(S^{(i_r)}) + f(u^{(i_r)} \mid S^{(i_r)}) = f(S^{(i_r)+}) < 1/2 \cdot f(OPT) \ , \tag{2}$$

where the first inequality follows from the way in which Algorithm 2 selects $u^{(i_r)}$ because $c(S^{(i_r)}) \leq \varepsilon(1+\varepsilon)^{i_r} \leq 1 - c(r)$, and the second inequality follows from Inequality (1).

Let h_r be the index for which $S_{h_r} = S^{(i_r)}$. If $h_r = \ell$, then $f(S_\ell) \geq (1 - 1/e) \cdot f(OPT)$ by Lemma 3.4 (for $T = OPT$) because $\max_{u \in OPT} c(u) = c(r) \leq 1 - c(S^{(i_r)}) = 1 - c(S_\ell)$. Therefore, we may assume $h_r < \ell$ in the rest of this proof. In particular, this implies that for every

$0 \leq h \leq h_r$, by plugging $T = OPT - r$ into Lemma 3.5, we get

$$\begin{aligned}
\frac{f(S_{h+1}) - f(S_h)}{c(u_{h+1})} &\geq \min\left\{(1-\varepsilon)\cdot\frac{f(OPT - r \mid S_h)}{c(OPT - r)}, \varepsilon^{-1}\cdot f(OPT)\right\} \\
&\geq \min\left\{(1-\varepsilon)\cdot\frac{f(OPT \mid S_h + r)}{c(OPT - r)}, \varepsilon^{-1}\cdot f(OPT)\right\} \\
&\geq \min\left\{(1-\varepsilon)\cdot\frac{f(OPT) - f(S_{h_r} + r)}{c(OPT - r)}, \varepsilon^{-1}\cdot f(OPT)\right\} \\
&\geq (1-\varepsilon)\cdot\frac{f(OPT) - f(S_{h_r} + r)}{1 - c(r)} \quad ,
\end{aligned}$$

where the second inequality follows from the submoduarity of f; the third inequality follows from f's monotonicity; and the last inequality holds since $f(S_{h_r} + r) \geq 0$, and $\max\{c(OPT - r), \varepsilon\} \leq 1 - c(r)$ by the condition of the lemma and the fact that OPT is a feasible solution. Plugging now Inequality (2) into the last inequality (and recalling that $S_{h_r} = S^{(i_r)}$) yields

$$\frac{f(S_{h+1}) - f(S_h)}{c(u_{h+1})} \geq (1-\varepsilon)\cdot\frac{1/2\cdot f(OPT)}{1 - c(r)} \quad .$$

Summing up the last inequality for all integers $0 \leq h \leq h_r$ now gives

$$\begin{aligned}
f(S_{h_r+1}) - f(S_0) &\geq \sum_{h=0}^{h_r}[f(S_{h+1}) - f(S_h)] \geq (1-\varepsilon)\cdot\frac{1/2\cdot f(OPT)}{1 - c(r)}\cdot\sum_{h=0}^{h_r} c(u_{h+1}) \\
&= (1-\varepsilon)\cdot\frac{1/2\cdot f(OPT)}{1 - c(r)}\cdot c(S_{h_r+1}) \geq (1-\varepsilon)\cdot\frac{1/2\cdot f(OPT)}{1 - c(r)}\cdot\varepsilon(1+\varepsilon)^{i_r} \\
&\geq (1-\varepsilon)\cdot\frac{1/2\cdot f(OPT)}{1 - c(r)}\cdot\frac{1 - c(r)}{1+\varepsilon} \geq (1/2 - \varepsilon)\cdot f(OPT) \quad ,
\end{aligned}$$

where the third inequality holds since the definition of $S^{(i_r)}$ implies that h_r is the maximal index for which $c(S_{h_r}) \leq \varepsilon(1+\varepsilon)^{i_r}$, and the penultimate inequality follows from the definition of i_r. □

D Inapproximability of SMC for Constant k Values

In this section we prove Theorem 4.1, which we repeat here for convinience.

Theorem 4.1. *Any (possibly randomized) algorithm guaranteeing α-approximation ($\alpha \in (0,1]$) for* `Submodular Maximization subject to a Cardinality Constraint (SMC)` *must use $\Omega(\alpha n/k)$ value oracle queries. In particular, this implies that the algorithm must make $\Omega(n)$ value oracle queries when α and k are constants.*

Proof. Let $t = 2k/\alpha$, and for every element $u \in \mathcal{N}$, consider the set function $f_u \colon \mathcal{N} \to \mathbb{R}_{\geq 0}$ defined, for every set $S \subseteq \mathcal{N}$, as

$$f_u(S) = \begin{cases} \min\{t, |S|\} & \text{if } u \notin S \quad , \\ t & \text{if } u \in S \quad . \end{cases}$$

Below we consider a distribution $\mathcal{D}$ over instances of SMC obtained by choosing u uniformly at random out of $\mathcal{N}$, and we show that this distribution is hard for every deterministic algorithm that uses $o(\alpha n/k)$ value oracle queries, which implies the theorem by Yao's principle.

Let ALG be any deterministic algorithm using $o(\alpha n/k)$ value oracle queries, and let $S_1, S_2, \ldots, S_\ell$ be the sets on which ALG queries the value oracle of the objective function when this function is chosen to be $\min\{t, |S|\}$. We assume without loss of generality that S_ℓ is the output set of ALG. Let E be the event that u does not belong to any of sets $S_1, S_2, \ldots, S_\ell$ that are of size less than t. Since ALG is deterministic, whenever the event E happens, ALG makes the same set of value oracle queries when given either $\min\{t, |S|\}$ or f_u as the objective function, and outputs $f_u(S_\ell) = \min\{t, |S|\} \leq |S_\ell| \leq k$ in both cases. Moreover, even when the event E does not happen, ALG still outputs a set of value at most t since f_u never returns larger value. Finally, observe that

the probability of the event E is at least $1 - \ell t/n$, and therefore, when the input instance is drawn from the distribution $\mathcal{D}$, the expected value of the output of ALG is at most

$$\frac{\ell t}{n} \cdot t + \left(1 - \frac{\ell t}{n}\right) \cdot k = \frac{\ell t^2}{n} + k - \frac{\ell t k}{n} \leq \frac{\ell t^2}{n} + k \ .$$

It remains to observe that the optimal solution for every instance in the support of the distribution $\mathcal{D}$ is $\{u\}$, whose value according to f_u is t. Therefore, the approximation ratio of ALG with respect to an instance of SMC drawn from $\mathcal{D}$ is no better than

$$\frac{\ell t^2/n + k}{t} = \frac{\ell t}{n} + \frac{k}{t} = \frac{2\ell k}{\alpha n} + \frac{\alpha}{2} = o(1) + \frac{\alpha}{2} \ ,$$

where the last equality holds since ALG makes $o(\alpha n/k)$ queries, and ℓ is the number of queries that it makes given a particular input. Hence, $\mathcal{D}$ is an hard distribution in the sense that ALG is not an α-approximation algorithm against an instance of SMC drawn from $\mathcal{D}$. □

E Inapproximability of SMC and Large k Values

We prove in Section E.1 a restricted version of Theorem 4.2 that applies only to deterministic algorithms. Then, Section E.2 extends the proof to randomized algorithms, which implies Theorem 4.2.

E.1 Inapproximability of SMC for Deterministic Algorithms and Large k Values

We begin this section by considering the following problem, termed `Set-Identification`, which has a rational parameter $\beta \in (0, 1)$. An algorithm for this problem has access to a ground set $\mathcal{N}$ of size n and a non-empty collection $\mathcal{C}$ of subsets of $\mathcal{N}$ of size $k = \beta n$ (formally, $\mathcal{C} \subseteq \{S \subseteq \mathcal{N} \mid |S| = k\}$). Each instance of `Set-Identification` also includes some hidden set $C^* \in \mathcal{C}$, which the algorithm can access only via an oracle that answers the following type of queries: given a set $S \subseteq \mathcal{N}$, the oracle returns the size of $S \cap C^*$. The objective of the algorithm is to output a set S of size k whose intersection with C^* is as large as possible.

The majority of this section is devoted to proving the following proposition regarding deterministic algorithms for `Set-Identification`. We then show that this proposition implies a version of Theorem 4.2 for deterministic algorithms.

Proposition E.1. *For every $\alpha \in (\beta, 1]$ and set $\mathcal{C}$, any deterministic α-approximation algorithm for `Set-Identification` must make $\frac{(k+1)\cdot[2(\alpha-\beta)^2+\ln\beta+(\beta^{-1}-1)\ln(1-\beta)-k^{-1}\ln n]+\ln|\mathcal{C}|}{\ln(k+1)}$ oracle queries.*

Let ALG be an arbitrary deterministic algorithm for `Set-Identification`. To prove Proposition E.1, we need to design an adversary that selects a hidden set C^* that is "bad" for ALG. Our adversary does not immediately commit to a particular set C^*. Instead, it maintains a list $\mathcal{L}$ of candidate sets that are consistent with all the oracle queries that ALG has performed so far. In other words, the set $\mathcal{L}$ originally includes all the sets of $\mathcal{C}$. Then, whenever ALG queries the oracle on some set S, the adversary returns some answer a and eliminates from $\mathcal{L}$ every set C for which $|S \cap C| \neq a$. To fully define the adversary, we still need to explain how it chooses the answer a for the oracle query. As the adversary wishes to keep the algorithm in the dark for as long as possible, it chooses the answer a to be the one that reduces the list $\mathcal{L}$ by the least amount.

Let $\mathcal{L}_i$ be the list $\mathcal{L}$ after the adversary answers i queries.

Observation E.2. *For every integer $i \geq 0$, $|\mathcal{L}_i| \geq |\mathcal{C}|/(k+1)^i$.*

Proof. Whenever the adversary has to answer a query regarding some set S, the answer corresponding to each set in the list $\mathcal{L}$ is some number in $\{0, 1, \ldots, k\}$. Therefore, at least one of these answers is guaranteed to reduce the list only by a factor of $|\{0, 1, \ldots, k\}| = k + 1$. Thus, after i queries have been answered, the size of the list $\mathcal{L}$ reduces by at most a factor of $(k + 1)^i$. To complete the proof of the observation, we recall that the original size of $\mathcal{L}$ is $|\mathcal{C}|$. □

Once ALG terminates and outputs some set S of size k, our adversary has to decide what set of $\mathcal{L}$ is the "real" hidden set C^*, which it does by simply choosing the set $C^* \in \mathcal{L}$ with the least intersection

with S. If the list $\mathcal{L}$ is still large at this point, then the adversary is guaranteed to be able to find in it a set with a low intersection with S. The following lemma quantifies this observation.

Lemma E.3. *For every set $S \subseteq \mathcal{N}$ of size k, a non-empty subset $\mathcal{L} \subseteq \mathcal{C}$ must include a set $C \in \mathcal{L}$ such that $|S \cap C| < \beta k + k\sqrt{(k^{-1}\ln n - \ln\beta - (\beta^{-1}-1)\ln(1-\beta) - \frac{1}{k+1}\ln|\mathcal{L}|)/2}$.*

Proof. Let $h = \beta k + k \cdot \sqrt{(k^{-1}\ln n - \ln\beta - (\beta^{-1}-1)\ln(1-\beta) - \frac{1}{k+1}\ln|\mathcal{L}|)/2}$. We would like to upper bound the probability of a uniformly random subset $R \subseteq \mathcal{N}$ of size k to intersect at least h elements of S. Since the expected size of the intersection of R and S is

$$\frac{|R| \cdot |S|}{|\mathcal{N}|} = \frac{k^2}{n} = \beta k \ ,$$

and the distribution of R is hyper-geometric, we get by Inequality (10) in [65] (developed based on the works of [14, 35]) that

$$\Pr[|R \cap S| \geq h] = \Pr[|R \cap S| \geq \mathbb{E}[|R \cap S|] + h - \beta k] \leq e^{-2((h-\beta k)/k)^2 \cdot k} = e^{-2(h-\beta k)^2/k} \ . \quad (3)$$

By Stirling's approximation, the total number of subsets of $\mathcal{N}$ whose size is k is

$$\begin{aligned}\binom{n}{k} &= \frac{n!}{k! \cdot (n-k)!} \leq \frac{en(n/e)^n}{e(k/e)^k \cdot e((n-k)/e)^{n-k}} \\ &= \frac{n}{e\beta^k \cdot (1-\beta)^{n-k}} \leq \frac{n}{\beta^k(1-\beta)^{n-k}} \ .\end{aligned}$$

Since Inequality (3) implies that at most a fraction of $e^{-2(h-\beta k)^2/k}$ out of these sets has intersection with S of size at least h, we get that the number of subsets of $\mathcal{N}$ of size k whose intersection with S is of size at least h can be upper bounded by

$$\begin{aligned}e^{-2(h-\beta k)^2/k} \cdot \binom{n}{k} &\leq \frac{ne^{-2(h-\beta k)^2/k}}{\beta^k(1-\beta)^{n-k}} \\ &= \frac{ne^{-(\ln n - k\ln\beta - (n-k)\cdot\ln(1-\beta) - \frac{k}{k+1}\ln|\mathcal{L}|)}}{\beta^k(1-\beta)^{n-k}} = |\mathcal{L}|^{\frac{k}{k+1}} < |\mathcal{L}| \ .\end{aligned}$$

The last bound implies that $|\mathcal{L}|$ must include at least one set whose intersection with S is less than h. □

We are now ready to prove Proposition E.1.

Proof of Proposition E.1. By Observation E.2 and Lemma E.3, if ALG makes i oracle queries, our adversary will be able to choose a hidden set C^* such that the intersection between the output set S of ALG and C^* is less than

$$\begin{aligned}&\beta k + k\sqrt{(k^{-1}\ln n - \ln\beta - (\beta^{-1}-1)\ln(1-\beta) - \tfrac{1}{k+1}\ln|\mathcal{L}_i|)/2} \\ &\leq \beta k + k \cdot \sqrt{\frac{1}{2} \cdot \left(k^{-1}\ln n - \ln\beta - (\beta^{-1}-1)\ln(1-\beta) + \frac{i \cdot \ln(k+1) - \ln|\mathcal{C}|}{k+1}\right)} \ .\end{aligned}$$

Since the optimal solution for `Set-Identification` is the set C^* itself, whose intersection with itself is k, the above inequality implies that the approximation ratio of ALG is worse than

$$\beta + \sqrt{\frac{1}{2} \cdot \left(k^{-1}\ln n - \ln\beta - (\beta^{-1}-1)\ln(1-\beta) + \frac{i \cdot \ln(k+1) - \ln|\mathcal{C}|}{k+1}\right)} \ .$$

If the approximation ratio of ALG is at least α, then we get the inequality

$$\alpha \leq \beta + \sqrt{\frac{1}{2} \cdot \left(k^{-1}\ln n - \ln\beta - (\beta^{-1}-1)\ln(1-\beta) + \frac{i \cdot \ln(k+1) - \ln|\mathcal{C}|}{k+1}\right)} \ ,$$

which implies that the number i of oracle queries made by ALG obeys

$$i \geq \frac{(k+1) \cdot [2(\alpha-\beta)^2 + \ln\beta + (\beta^{-1}-1)\ln(1-\beta) - k^{-1}\ln n] + \ln|\mathcal{C}|}{\ln(k+1)} \ . \qquad \square$$

Using Proposition E.1, we can now prove the promised version of Theorem 4.2 for deterministic algorithms.

Corollary E.4. *For any rational constant $\beta \in (0,1)$ and (not necessarily constant) $\varepsilon = \omega(\sqrt{\frac{\log n}{n}})$, every deterministic algorithm for* SMC *that guarantees an approximation ratio of $\beta + \varepsilon$ for instances obeying $k = \beta n$ must use $\Omega(\frac{\varepsilon^2 n}{\log n})$ value oracle queries. Moreover, this is true even when the objective function f of* SMC *is guaranteed to be a linear function.*

Proof. Let $\mathcal{C}$ be the set of all subsets of $\mathcal{N}$ of size βn, and let $k = \beta n$. By Stirling's approximation, the size of the set $\mathcal{C}$ is at least

$$\binom{n}{\beta n} \geq \frac{e(n/e)^n}{e\beta n(\beta n/e)^{\beta n} \cdot e(1-\beta)n((1-\beta)n/e)^{(1-\beta)n}} = \frac{1}{en^2\beta^{\beta n+1} \cdot (1-\beta)^{(1-\beta)n+1}} \ .$$

Therefore, the number of oracle queries made by any deterministic $(\beta+\varepsilon)$-approximation algorithm for `Set-Identification` with the above choice of parameters $\mathcal{C}$ and k is at least

$$\begin{aligned}
&\frac{(\beta n+1) \cdot [2\varepsilon^2 + \ln\beta + (\beta^{-1}-1)\ln(1-\beta) - (\beta n)^{-1}\ln n] + \ln|\mathcal{C}|}{\ln(\beta n+1)} \\
\geq\ &\frac{(\beta n+1) \cdot [2\varepsilon^2 + \ln\beta + (\beta^{-1}-1)\ln(1-\beta) - (\beta n)^{-1}\ln n]}{\ln(\beta n+1)} \\
&- \frac{1 + 2\ln n + (\beta n+1)\ln\beta + ((1-\beta)n+1)\ln(1-\beta)}{\ln(\beta n+1)} \\
=\ &\frac{(\beta n+1) \cdot [2\varepsilon^2 - (\beta n)^{-1}\ln n] - 1 - 2\ln n + (\beta^{-1}-2)\ln(1-\beta)}{\ln(\beta n+1)} \\
=\ &\frac{\varepsilon^2 \cdot \Theta(n) - \Theta(\log n)}{\Theta(\log n)} = \Omega\left(\frac{\varepsilon^2 n}{\log n}\right) \ ,
\end{aligned}$$

where the last equality holds since we assume $\varepsilon = \omega(\sqrt{\frac{\log n}{n}})$.

Assume now that we have an algorithm ALG for `SMC` with the guarantee stated in the corollary. We construct an algorithm ALG' for `Set-Identification` that consists of the following three steps.

1. Construct an instance of `SMC` by setting $k = \beta n$ and $f(S) = |S \cap C^*|$.
2. Since f is a linear function, we can use ALG to find a set $S \subseteq \mathcal{N}$ of size at most βn such that $f(S) \geq (\beta+\varepsilon) \cdot f(C^*) = (\beta+\varepsilon)|C^*|$.
3. Return any subset $S' \subseteq \mathcal{N}$ of size βn that includes S. Clearly, $|S' \cap C^*| \geq |S \cap C^*| = f(S) \geq (\beta+\varepsilon)|C^*|$.

The algorithm ALG' obtains $(\beta+\varepsilon)$-approximation for `Set-Identification`, and therefore, by the above discussion, it must use $\Omega(\frac{\varepsilon^2 n}{\log n})$ queries to the oracle of `Set-Identification`. However, since ALG' queries the oracle of `Set-Identification` only once for every time that ALG queries the value oracle of f, we get that ALG must be using $\Omega(\frac{\varepsilon^2 n}{\log n})$ value oracle queries. □

E.2 Inapproximability of `SMC` for Randomized Algorithms and Large k Values

In this section we extend the inapproximability result from Section E.2 to randomized algorithms. To do that, we start by proving the following version of Proposition E.1 for randomized algorithms.

Proposition E.5. *For every $\alpha \in (\beta, 1]$ and set $\mathcal{C}$, any (possibly randomized) α-approximation algorithm for* `Set-Identification` *must make*

$$\frac{(k+1) \cdot [(\alpha-\beta)^2/2 + \ln\beta + (\beta^{-1}-1)\ln(1-\beta) - k^{-1}\ln n] + \ln(|\mathcal{C}|) + \ln(\frac{\alpha-\beta}{2-\alpha-\beta})}{\ln(k+1)}$$

oracle queries in the worst case.[3]

[3] One can also prove in a similar way a version of this proposition bounding the expected number of oracle queries.

Proof. Let ALG be an algorithm of the kind described in the proposition, assume that the set C^* is chosen uniformly at random out of $\mathcal{C}$ and let S_R be the output set of ALG. Using a Markov like argument, we get that since ALG has an approximation guarantee of α and $|S_R \cap C^*|$ is always at most $|C^*| = k$,

$$\Pr\left[|S_R \cap C^*| \geq \frac{(\alpha+\beta)k}{2}\right] \geq \frac{\alpha-\beta}{2-\alpha-\beta} \ ,$$

where the probability is over both the random choice of C^* and the randomness of ALG itself.

Recall now that ALG, as a randomized algorithm, can be viewed as a distribution over deterministic algorithms. This means that there must be at least one (deterministic) algorithm ALG_D in the support of this distribution such that its output set S_D obeys

$$\Pr\left[|S_D \cap C^*| \geq \frac{(\alpha+\beta)k}{2}\right] \geq \Pr\left[|S_R \cap C^*| \geq \frac{(\alpha+\beta)k}{2}\right] \geq \frac{\alpha-\beta}{2-\alpha-\beta} \ .$$

However, since ALG_D is a deterministic algorithm, S_D is a function of the set C^* alone, which implies that there exists a subset $\mathcal{C}' \subseteq \mathcal{C}$ of size at least $\frac{\alpha-\beta}{2-\alpha-\beta} \cdot |\mathcal{C}|$ such that the algorithm ALG_D guarantees $\frac{\alpha+\beta}{2}$-approximation whenever $C^* \in \mathcal{C}'$.

By Proposition E.1, the number of oracle queries used by ALG_D must be at least

$$\frac{(k+1)\cdot[2(\frac{\alpha+\beta}{2}-\beta)^2+\ln\beta+(\beta^{-1}-1)\ln(1-\beta)-k^{-1}\ln n]+\ln|\mathcal{C}'|}{\ln(k+1)}$$
$$\geq \frac{(k+1)\cdot[(\alpha-\beta)^2/2+\ln\beta+(\beta^{-1}-1)\ln(1-\beta)-k^{-1}\ln n]+\ln|\mathcal{C}|+\ln(\frac{\alpha-\beta}{2-\alpha-\beta})}{\ln(k+1)} \ .$$

The proposition now follows since the number of oracle queries made by ALG_D is a lower bound on the number of oracle queries made by ALG in the worst case. □

Corollary E.6. *For any rational constant $\beta \in (0,1)$ and (not necessarily constant) $\varepsilon = \omega(\sqrt{\frac{\log n}{n}})$, every (possibly randomized) algorithm for* `SMC` *that guarantees an approximation ratio of $\beta+\varepsilon$ for instances obeying $k = \beta n$ must use $\Omega(\frac{\varepsilon^2 n}{\log n})$ value oracle queries. Moreover, this is true even when the objective function f of* `SMC` *is guaranteed to be a linear function.*

Proof. Let $\mathcal{C}$ be the set of all subsets of $\mathcal{N}$ of size βn, and let $k = \beta n$. As was proved in the proof of Corollary E.4,

$$\binom{n}{\beta n} \geq \frac{1}{en^2\beta^{\beta n+1}\cdot(1-\beta)^{(1-\beta)n+1}} \ ,$$

and therefore, by Proposition E.5, the number of oracle queries made by any $(\beta+\varepsilon)$-approximation algorithm for `Set-Identification` with the above choice of parameters $\mathcal{C}$ and k is at least

$$\frac{(\beta n+1)\cdot[\varepsilon^2/2+\ln\beta+(\beta^{-1}-1)\ln(1-\beta)-(\beta n)^{-1}\ln n]+\ln|\mathcal{C}|+\ln(\frac{\varepsilon}{2-2\beta-\varepsilon})}{\ln(\beta n+1)}$$
$$\geq \frac{(\beta n+1)\cdot[\varepsilon^2/2+\ln\beta+(\beta^{-1}-1)\ln(1-\beta)-(\beta n)^{-1}\ln n]+\ln(\varepsilon/2)}{\ln(\beta n+1)}$$
$$-\frac{1+2\ln n+(\beta n+1)\ln\beta+((1-\beta)n+1)\ln(1-\beta)}{\ln(\beta n+1)}$$
$$=\frac{(\beta n+1)\cdot[\varepsilon^2/2-(\beta n)^{-1}\ln n]-1-2\ln n+(\beta^{-1}-2)\ln(1-\beta)+\ln(\varepsilon/2)}{\ln(\beta n+1)}$$
$$=\frac{\varepsilon^2\cdot\Theta(n)-\Theta(\log n)-\Theta(\log\varepsilon^{-1})}{\Theta(\log n)}=\Omega\left(\frac{\varepsilon^2 n}{\log n}\right) \ ,$$

where the last equality holds since we assume $\varepsilon = \omega(\sqrt{\frac{\log n}{n}})$.

The rest of the proof is completely identical to the corresponding part in the proof of Corollary E.4 (up to the need to add some expectations signs), and therefore, we omit it. □

Theorem 4.2 is the special case of Corollary E.6 in which ε is a positive constant, and therefore, obeys $\varepsilon = \omega(\sqrt{\frac{\log n}{n}})$.

F Inapproximability for USM

In this section we reuse the machinery developed in Section E.2 to get a query complexity lower bound for USM and prove Theorem 4.3. Before getting to the proof, we recall that Table 2 gives the context of this theorem, and shows that this theorem nearly completes our understanding of the minimum query complexity necessary to obtain various approximation ratios for USM.

We assume throughout the section the following parameters for Set-Identification. The parameter k is set to be $n/2$, and $\mathcal{C}$ is the collection of all subsets of $\mathcal{N}$ of size k. One can verify that the proof of Corollary E.6 also implies the following lemma (as a special case for $\beta = 1/2$).

Lemma F.1. *Given the above parameters, for any $\varepsilon = \omega(\sqrt{\frac{\log n}{n}})$, every (possibly randomized) $(1/2 + \varepsilon)$-approximation algorithm for Set-Identification must use $\Omega(\frac{\varepsilon^2 n}{\log n})$ oracle queries.*

Our next objective is to show a reduction from Set-Identification to USM. Let ALG be a $(1/4 + \varepsilon)$-approximation algorithm for USM. Using ALG, one can design the algorithm for Set-Identification (with the parameters we assume) that appears as Algorithm 5. It is important to observe that the value oracle of the set function f defined on Line 1 of the algorithm can be implemented using one query to the oracle of Set-Identification. Furthermore, f is the cut-function of a directed graph, and therefore, it is non-negative and submodular. This allows Algorithm 5 to use ALG to construct a set T of large expected value. The set T is then replaced by it complement $\mathcal{N} \setminus T$ if this increases the value of $f(T)$. Naturally, this replacement can only increase the expected value of the set T, and also gives it some deterministic properties that we need. Algorithm 5 completes by converting the set T into an output set of size $n/2$ in one of two ways. If T is too small, then a uniformly random subset of $\mathcal{N} \setminus T$ of the right size is added to it. Otherwise, if T is too large, then a uniformly random subset of it of the right size is picked.

Algorithm 5: Reduction from Set-Identification to USM

1 Define a function $f \colon 2^{\mathcal{N}} \to \mathbb{R}_{\geq 0}$ by $f(S) = |S \cap C^*| \cdot (\frac{n}{2} - |S \setminus C^*|)$.
2 Use ALG to find a set T that such that $\mathbb{E}[f(T)] \geq (1/4 + \varepsilon) \cdot \max_{S \subseteq \mathcal{N}} f(S) = (1/4 + \varepsilon) \cdot n^2/4$.
3 **if** $f(T) < f(\mathcal{N} \setminus T)$ **then** Update $T \leftarrow \mathcal{N} \setminus T$.
4 **if** $|T| \leq n/2$ **then**
5 Pick a uniformly at random subset R of $\mathcal{N} \setminus T$ of size $n/2 - |T|$.
6 **return** $T \cup R$.
7 **else**
8 Pick a uniformly at random subset R of T of size $n/2$.
9 **return** R.

Let T and $\tilde{T}$ denote the values of the set T before and after Line 3 of Algorithm 5. The following observation states the properties of the set $\tilde{T}$ that we need below.

Observation F.2. *The set $\tilde{T}$ obeys*

- $\mathbb{E}[f(\tilde{T})] \geq (1/4 + \varepsilon) \cdot n^2/4$.
- $|\tilde{T} \cap C^*| \geq |\tilde{T} \setminus C^*|$.

Proof. Line 3 of Algorithm 5 guarantees that $f(\tilde{T}) \geq f(T)$, which implies

$$\mathbb{E}[f(\tilde{T})] \geq \mathbb{E}[f(T)] \geq (1/4 + \varepsilon) \cdot n^2/4 \ .$$

To see that the second part of the observation holds as well, note that Line 3 also guarantees $f(\tilde{T}) \geq f(\mathcal{N} \setminus \tilde{T})$, which, by the definition of f, implies

$$|\tilde{T} \cap C^*|(\tfrac{n}{2} - |\tilde{T} \setminus C^*|) \geq |\tilde{T} \setminus C^*|(\tfrac{n}{2} - |\tilde{T} \cap C^*|) \ ,$$

Rearranging this inequality now gives

$$\tfrac{n}{2}|\tilde{T} \cap C^*| \geq \tfrac{n}{2}|\tilde{T} \setminus C^*| \ ,$$

and dividing this inequality by $n/2$ yields the required inequality. □

One consequence of the last lemma is given by the next corollary.

Corollary F.3. $\mathbb{E}[|\tilde{T} \cap C^*| - |\tilde{T} \setminus C^*|] \geq n\varepsilon/2$.

Proof. Observe that

$$\begin{aligned}
&\tfrac{n}{2} \cdot \mathbb{E}[|\tilde{T} \cap C^*| - |\tilde{T} \setminus C^*|] \geq \mathbb{E}[(|\tilde{T} \cap C^*| - |\tilde{T} \setminus C^*|)(\tfrac{n}{2} - |\tilde{T} \setminus C^*|)] \\
&\geq \mathbb{E}[(|\tilde{T} \cap C^*| - |\tilde{T} \setminus C^*|)(\tfrac{n}{2} - |\tilde{T} \setminus C^*|)] + \mathbb{E}[|\tilde{T} \setminus C^*|(\tfrac{n}{2} - |\tilde{T} \setminus C^*|)] - \frac{n^2}{16} \\
&= \mathbb{E}[|\tilde{T} \cap C^*|(\tfrac{n}{2} - |\tilde{T} \setminus C^*|)] - \frac{n^2}{16} = \mathbb{E}[f(\tilde{T})] - \frac{n^2}{16} \geq \frac{n^2\varepsilon}{4} \ ,
\end{aligned}$$

where the first inequality holds since Observation F.2 guarantees that $|\tilde{T} \cap C^*| - |\tilde{T} \setminus C^*|$ is always non-negative, and the last inequality follows also from Observation F.2. The corollary now follows by dividing this inequality by $n/2$. □

We are now ready to prove the following lemma, which analyzes the performance guarantee of Algorithm 5.

Lemma F.4. *If T' is the output set of Algorithm 5, then $\mathbb{E}[|T' \cap C^*|] \geq (1/2 + \varepsilon/4) \cdot n/2$, where the expectation is over the randomness of ALG and Algorithm 5.*

Proof. Let us begin this proof by fixing the set $\tilde{T}$. In other words, until we unfix this set, all the expectations we use are assumed to be only over the randomness of Lines 5 and 8 of Algorithm 5. Whenever Line 8 of the algorithm is used, we have

$$\begin{aligned}
\mathbb{E}[|T' \cap C^*|] &= \frac{n/2}{|\tilde{T}|} \cdot |\tilde{T} \cap C^*| = \frac{n}{4} \cdot \left(1 + \frac{2|\tilde{T} \cap C^*| - |\tilde{T}|}{|\tilde{T}|}\right) \\
&= \frac{n}{4} \cdot \left(1 + \frac{|\tilde{T} \cap C^*| - |\tilde{T} \setminus C^*|}{|\tilde{T}|}\right) \geq \frac{n}{4} \cdot \left(1 + \frac{|\tilde{T} \cap C^*| - |\tilde{T} \setminus C^*|}{n}\right) \ ,
\end{aligned}$$

where the inequality holds since Observation F.2 guarantees that $|\tilde{T} \cap C^*| - |\tilde{T} \setminus C^*| \geq 0$. Similarly, whenever Line 5 of Algorithm 5 is used, we get

$$\begin{aligned}
\mathbb{E}[|T' \cap C^*|] &= |\tilde{T} \cap C^*| + \mathbb{E}[|R \cap C^*|] = |\tilde{T} \cap C^*| + \frac{\frac{n}{2} - |\tilde{T}|}{|\mathcal{N} \setminus \tilde{T}|} \cdot |C^* \setminus \tilde{T}| \\
&= |\tilde{T} \cap C^*| + \frac{\frac{n}{2} - |\tilde{T}|}{n - |\tilde{T}|} \cdot (\tfrac{n}{2} - |\tilde{T} \cap C^*|) = \frac{\frac{n}{2}|\tilde{T} \cap C^*| + \frac{n}{2}(\frac{n}{2} - |\tilde{T}|)}{n - |\tilde{T}|} \\
&= \frac{n}{4} \cdot \left(1 + \frac{|\tilde{T} \cap C^*| - |\tilde{T} \setminus C^*|}{n - |\tilde{T}|}\right) \geq \frac{n}{4} \cdot \left(1 + \frac{|\tilde{T} \cap C^*| - |\tilde{T} \setminus C^*|}{n}\right) \ .
\end{aligned}$$

Therefore, even if we unfix $\tilde{T}$, and take expectation also over the randomness of this set, we still get by the law of total expectation that

$$\mathbb{E}[|T' \cap C^*|] \geq \frac{n}{4} \cdot \left(1 + \frac{\mathbb{E}[|\tilde{T} \cap C^*| - |\tilde{T} \setminus C^*|]}{n}\right) \geq \frac{n}{4} \cdot \left(1 + \frac{n\varepsilon/2}{n}\right) = \frac{n}{2} \cdot \left(\frac{1}{2} + \frac{\varepsilon}{4}\right) \ ,$$

where the second inequality follows from Corollary F.3. □

We can now prove Theorem 4.3, which we repeat here for convenience.

Theorem 4.3. *For any constant $\varepsilon > 0$, any algorithm for* `Unconstrained Submodular Maximization` *that guarantees an approximation ratio of $1/4 + \varepsilon$ must use $\Omega(\frac{n}{\log n})$ value oracle queries.*

Proof. By Lemma F.1, Algorithm 5 must use at least $\Omega(\frac{\varepsilon^2 n}{\log n})$ queries to the oracle of `Set-Identification`. However, aside from the queries used by ALG, Algorithm 5 queries this oracle only a constant number of times, which implies that ALG must be using $\Omega(\frac{\varepsilon^2 n}{\log n})$ such queries. Recall now that ALG, as an algorithm for `USM`, queries the oracle of `Set-Identification` only by querying the value oracle of f, and every query to this value oracle results in a single query to the oracle of `Set-Identification`. Therefore, ALG must be using $\Omega(\frac{\varepsilon^2 n}{\log n})$ value oracle queries. This completes the proof of the theorem since ALG was chosen as an arbitrary $(1/4 + \varepsilon)$-approximation algorithm for `USM`. □

G Solving `Set-Identification` using $O(n/\log n)$ Oracle Queries

Recall that the inapproximability results proved in Section 4 (namely, Theorems 4.2 and 4.3) are based on a reduction to a problem named `Set-Identification` (defined in Section E.1). In Section E.2, we prove that any (possibly randomized) algorithm for `Set-Identification` must use $\Omega(n/\log n)$ oracle queries to significantly improve over some "easy" approximation ratio. In this section we show that this result is tight in the sense that $O(n/\log n)$ oracle queries suffice to solve `Set-Identification` exactly. The algorithm we use for this purpose is given as Algorithm 6. This algorithm is not efficient in terms of its time complexity. Nevertheless, it shows that one cannot prove an inapproximability result requiring $\omega(n/\log n)$ oracle queries for `Set-Identification` based on information theoretic arguments only.

For simplicity, we assume in Algorithm 6 that the ground set $\mathcal{N}$ is simply the set $[n]$. Given this assumption, we are able to base the algorithm on the following known lemma.

Lemma G.1 (Due to Lev and Yuster [53])**.** *If $q > (2\log_2 3 + o(1))\frac{n}{\log n}$, then there exists a binary matrix $\mathbf{Q} \in \{0,1\}^{q \times n}$ such that, for every vector $\mathbf{s}$, the equation $\mathbf{Q} \cdot \mathbf{x} = \mathbf{s}$ has at most one binary solution $\mathbf{x} \in \{0,1\}^n$.*

Algorithm 6 begins by finding such a matrix $\mathbf{Q}$ (this can be done by brute-force enumeration since we do not care about the time complexity). Then, the product $\mathbf{Q} \cdot \mathbf{1}_{C^*}$ is calculated using $q = O(n/\log n)$ oracle queries, where $\mathbf{1}_{C^*}$ is the characteristic vector of the set C^* (*i.e.*, a vector that includes 1 in coordinate i if $i \in C^*$, and otherwise, includes 0 in this coordinate). Once Algorithm 6 has the product $\mathbf{Q} \cdot \mathbf{1}_{C^*}$, Lemma G.1 guarantees that it is possible to recover the vector $\mathbf{1}_{C^*}$ itself, which the algorithm can do, again, using brute-force.

Algorithm 6: Algorithm for `Set-Identification`

Let $q = O(n/\log n)$ be a large enough value so that it obeys the requirement of Lemma G.1.
Let $\mathbf{Q} \in \{0,1\}^{q \times n}$ be a binary matrix with the properties stated in Lemma G.1.
for $j = 1$ **to** q **do**
  Let Q_j be a set such that $\mathbf{1}^T_{Q_j}$ is equal to the j-th line of $\mathbf{Q}$.
  Let $s_j = |Q_j \cap C^*|$. `// Can be calculated using a single oracle query.`
Let $\mathbf{s}$ be the vector whose j-th coordinate, for every integer $1 \leq j \leq q$ is s_j. `// Note that` $\mathbf{s} = \mathbf{Q} \cdot \mathbf{1}_{C^*}$.
Let $\mathbf{x} \in \{0,1\}^n$ be a solution for $\mathbf{Q} \cdot \mathbf{x} = \mathbf{s}$. `// By Lemma G.1,` $\mathbf{x} = \mathbf{1}_{C^*}$ `is the sole solution for this equation in` $\{0,1\}^n$.
return *the set whose characteristic vector is* $\mathbf{x}$.

H Proof of Theorem 5.1

In this section we prove Theorem 5.1. The rank of a set system $\mathcal{M} = (\mathcal{N}, \mathcal{I})$ is defined as the maximum size of an independent set in it; and an element $u \in \mathcal{N}$ is a self-loop of this system if $\{u\}$

is not independent, which implies that u does not appear in any independent set of $\mathcal{M}$. Below, we denote by r the rank of the set system $\mathcal{M}$. We also make a few simplifying assumptions.

- We assume that $\mathcal{M}$ does not include any self-loops or elements $u \in \mathcal{N}$ such that $c_i(u) > B_i$ for some integer $1 \leq i \leq d$. Any element violating these assumptions can be simply discarded since it cannot belong to any feasible set.
- We assume that the sum $\sum_{i=1}^{d} c_i(u)$ is strictly positive for every element $u \in \mathcal{N}$. Elements violating this assumption can be added to any solution, and therefore, it suffices to solve the problem without such elements, and then add them to the obtained solution at the every end of the algorithm.
- We assume that $B_i = 1$ for every integer $1 \leq i \leq d$. This can be guaranteed by scaling the cost functions c_i.

In Section H.1 we present and analyze a basic version of the algorithm that we use to prove Theorem 5.1 (in Appendix I.1 we explain how to make this basic algorithm nearly-linear). The basic version of our algorithm assumes access to an estimate ρ of the density of the small elements of an optimal solution for the problem. In Section H.2 we explain how the dependence of the algorithm on ρ can be dropped without increasing the time complexity of the algorithm by too much. Finally, in Section H.3 we show how our algorithm can be used to derive the results stated in Theorem 5.1.

H.1 Basic Algorithm for SMKS

In this section we present and analyze a basic version algorithm of the algorithm that we use to prove Theorem 5.1. This algorithm is given as Algorithm 3, and it gets two parameters. The first of these parameters is an integer $\lambda \geq 1$. Elements that have a value larger than λ^{-1} with respect to at least one function c_i are considered big elements, and are stored in the set B of the algorithm. The other elements of $\mathcal{N}$ are considered small elements. The algorithm never considers any solution that includes both big and small elements. Instead, it creates one candidate solution S_B from the big elements, and one candidate solution from the small elements, and then outputs the better among the two (technically, in some cases the algorithm outputs directly the candidate solution based on the small elements without comparing it to S_B). The candidate solution S_B is constructed using a procedure called `BigElementsAlg` that gets the set B as input and outputs a feasible set whose value is at least $\alpha \cdot f(OPT \cap B)$, where α is a some value in $(0, 1]$ and OPT is an arbitrary optimal solution. At this point we ignore the implementation of `BigElementsAlg`, and leave the value of α unspecified. These gaps are filled in Section H.3.

Most of Algorithm 3 is devoted to constructing the candidate solution out of small elements, which we refer to below as the "small elements solution". In the construction of this solution, Algorithm 3 uses its second parameter, which is a value $\rho \geq 0$ that intuitively should represent the density of the small elements of OPT. The algorithm initializes the small elements solution to be empty, and then iteratively adds to it the element with the largest marginal contribution among the small elements that have two properties: (i) their addition to the solution does not make it dependent in $\mathcal{M}$, and (ii) their density (the ratio between their marginal contribution and cost according to the linear constraints) is at least ρ. This process of growing the small elements solution can end in one of two ways. One option is that the process ends because no additional elements can be added to the solution (in other words, no element has the two properties stated above). In this case the better among S_B and the small elements solution obtained S_k is returned. The other way in which the process of growing the small elements solution can end is when it starts violating at least one linear constraint. When this happens, Algorithm 3 uses a procedure called `SetExtract` to get a subset of the small elements solution that is feasible and also has a good value, and this subset is returned.

Let us now describe the procedure `SetExtract`, which appears as Algorithm 7. As explained above, this procedure gets a set S of small elements that violates at least one of the linear constraints. Its objective is to output a subset T of S that does not violate any linear constraint, but is not very small in terms of the linear constraints. Since the set S passes by Algorithm 3 to `SetExtract` contains only elements of density at least ρ, this implies that the output set T of `SetExtract` has a significant value. Algorithm 7 does its job by constructing $\lambda + 1$ subsets $T_1, T_2, \ldots, T_{\lambda+1}$ of S, and then outputting the subset with the maximum size with respect to the linear constraint. The first subset T_1 is constructed by starting with the empty set, and then simply adding elements of S to T_1 one by one, in an arbitrary order, until some element u_1 cannot be added because adding it will

result in a set that violates some linear constraint. The algorithm then constructs the second subset T_2 in essentially the same way, but makes sure to include u_1 in it by starting with the set $\{u_1\}$ and then adding elements of S to T_2 one by one in an arbitrary order, until some element u_2 cannot be added because adding it will result in a set that violates some linear constraint. The set T_3 is then constructed in the same way starting from the set $\{u_1, u_2\}$, and in general the set T_j is constructed by starting from the set $\{u_1, u_2, \ldots, u_{j-1}\}$ and then adding to it elements of S in an arbitrary order until some element u_j cannot be added because adding it will result in a set that violates some linear constraint. Intuitively, this method of constructing the subsets $T_1, T_2, \ldots, T_{\lambda+1}$ guarantees that every element u_j is rejected at most once from a subset due to the linear constraints.

Algorithm 7: `SetExtract`(λ, S)

1 **for** $j = 1$ **to** $\lambda + 1$ **do**
2 Let $T_j \leftarrow \{u_1, u_2, \ldots, u_{j-1}\}$.
3 **for** *every element* $u \in S$ **do**
4 **if** $\max_{1 \le i \le d} c_i(T_j + u) \le 1$ **then** Add u to T_j.
5 **else** Denote the element u by u_j from this point on, and exit the loop of Line 3.
6 **return** *the set maximizing* $\sum_{i=1}^{d} c_i(T)$ *among all sets* $T \in \{T_1, T_2, \ldots, T_{\lambda+1}\}$.

A formal statement of the guarantee of `SetExtract` is given by the next lemma.

Lemma H.1. *Assuming the input set S of `SetExtract` obeys*

- $c_i(S) > 1$ *for some integer* $1 \le i \le d$ *and*
- $\max_{1 \le i \le d} c_i(u) \le \lambda^{-1}$ *for every element* $u \in S$*,*

then the output set T is a subset of S such that $\max_{1 \le i \le d} c(T) \le 1$*, but* $\sum_{i=1}^{d} c_i(T) \ge \frac{\lambda}{\lambda+1}$.

Proof. For any $1 \le j \le \lambda + 1$, `SetExtract` initializes the set T_j to contain $j - 1 \le \lambda$ elements, which implies that immediately after this initialization the set T_j obeyed $\max_{1 \le i \le d} c_i(T_j) \le 1$ because of the second condition of the lemma. After the initialization of T_j, Algorithm 7 grows it by adding to it only elements whose addition does not make $\max_{1 \le i \le d} c_i(T_j)$ exceed 1. This method of growth guarantees that the set T_j keeps obeying $\max_{1 \le i \le d} c_i(T_j) \le 1$ throughout the execution of the algorithm. Therefore, since T is chosen as the set T_j for some integer j, it obeys $\max_{1 \le i \le d} c_i(T) \le 1$.

Consider any iteration of the loop on Line 1 of `SetExtract`. If this loop never reaches Line 5, then we are guaranteed that all the elements of S are added to T_j, which contradicts the inequality $\max_{1 \le i \le d} c_i(T) \le 1$ that we have proved above because we are guaranteed that $c_i(S) > 1$ for some integer $1 \le i \le d$. Therefore, `SetExtract` reaches Line 5 in every iteration of the outter loop. Specifically, for any integer $1 \le j \le \lambda$, since the algorithm reached Line 5 in iteration number j of this loop, we must have $\max_{1 \le i \le d} c_i(T_j + u_j) > 1$. Hence,

$$\sum_{i=1}^{d} c_i(T_j) = \sum_{i=1}^{d} [c_i(T_j + u_j) - c_i(u_j)] \ge \max_{1 \le i \le d} c_i(T_j + u_j) - \sum_{i=1}^{d} c_i(u_j) > 1 - \sum_{i=1}^{d} c_i(u_j) \ .$$

Using this inequity and the observation that $T_{\lambda+1}$ includes all the elements $u_1, u_2, \ldots, u_\lambda$, we get

$$\begin{aligned} \sum_{i=1}^{d} c_i(T) &= \max_{1 \le j \le \lambda+1} \sum_{i=1}^{d} c_i(T_j) \ge \frac{\sum_{j=1}^{\lambda+1} \sum_{i=1}^{d} c_i(T_j)}{\lambda + 1} \\ &\ge \frac{\sum_{j=1}^{\lambda} [1 - \sum_{i=1}^{d} c_i(u_j)] + \sum_{j=1}^{\lambda} \sum_{i=1}^{d} c_i(u_j)}{\lambda + 1} = \frac{\lambda}{\lambda + 1} \ . \end{aligned}$$

□

We now get to the analysis of the full Algorithm 3. Let ℓ be the final value of the variable k of Algorithm 3.

Observation H.2. *Algorithm 3 outputs a feasible set.*

Proof. One can observe that the set S_ℓ is feasible because the conditions of Lines 4 and 7 of Algorithm 3 guarantee that it assigns to S_k only feasible sets for any $0 \leq k \leq \ell$. Furthermore, S_B is feasible by the properties we assume for the procedure `BigElementsAlg`. Therefore, to prove the observation it only remains to show that the output set T of $\texttt{SetExtract}(\lambda, S_{\ell+1})$ is feasible when Line 8 is executed.

To see that this is indeed the case, we note that T is a subset of $S_{\ell+1}$, which is an independent set of $\mathcal{M}$, and therefore, T is also independent in $\mathcal{M}$. Additionally, Lemma H.1 implies that $\max_{1 \leq i \leq d} c_i(T) \leq 1$ because (i) we are guaranteed by the condition of Line 7 that there exists an integer $1 \leq d \leq k$ such that $c_i(S_{\ell+1}) = c_i(S_\ell + v_{\ell+1}) > 1$ when Line 8 is executed, and (ii) we are guaranteed that $\max_{u \in S_{\ell+1}} \max_{1 \leq i \leq d} c_i(u) \leq \lambda^{-1}$ since $S_{\ell+1}$ contains only small elements. □

Next, we analyze the time complexity of Algorithm 3.

Lemma H.3. *Algorithm 3 has a time complexity of $O(\lambda nd + nr + T_B)$, where T_B is the time complexity of `BigElementsAlg`.*

Proof. The construction of the set B requires $O(dn)$ time, and therefore, the time complexity required for the entire Algorithm 3 except for the loop starting on Line 4 is $O(dn + T_B)$. In the rest of this proof we show that this loop requires $O(\lambda nd + nr)$ time, which implies the lemma.

The loop starting on Line 4 of Algorithm 3 runs at most r times because the size of the set S_k grows by at least one after every such iteration (and this set always remains feasible in $\mathcal{M}$). To understand the time complexity of the iterations of the last loop, we can observe that, assuming we maintain the values $c_i(S_k)$, each such iteration takes $O(n + d)$ time, with the exception of the following operations.

- In Line 4 we need to calculate the sum $\sum_{i=1}^{d} c_i(u)$ for multiple elements u, which takes $O(d)$ time per element. However, we can pre-calculate this sum for all the elements of $\mathcal{N}$ in $O(nd)$ time.
- Executing $\texttt{SetExtract}(\lambda, S_{k+1})$ requires $O(\lambda |S_{k+1}| d) = O(\lambda rd)$ time. However, this procedure is executed at most once by Algorithm 3.

Combining all the above, we get that the loop starting on Line 4 of Algorithm 3 requires only $O(nd + \lambda rd + r(n + d)) = O(\lambda nd + nr)$ time. □

Our next objective is to analyze the approximation ratio of Algorithm 3. Let E be the event that Algorithm 3 returns through Line 8. We begin the analysis of Algorithm 3 by looking separately at the case in which the event E happens and at the case in which it does not happen. When the event E happens, the output set of Algorithm 3 is the output set of `SetExtract`. This set contains only high density elements and is large in terms of the linear constraints (by Lemma H.1), which provides a lower bound on its value. The following lemma formalizes this argument.

Lemma H.4. *If the event E happens, then Algorithm 3 returns a solution of value at least $\frac{\lambda\rho}{\lambda+1}$.*

Proof. Let T be the output set of $\texttt{SetExtract}(\lambda, S_{\ell+1})$. We need to show that $f(T) \geq \frac{\lambda\rho}{\lambda+1}$. Since T is a subset of $S_{\ell+1} = \{v_1, v_2, \ldots, v_{\ell+1}\}$,

$$\begin{aligned} f(T) &= \sum_{v_k \in T} f(v_k \mid T \cap S_{k-1}) \geq \sum_{v_k \in T} f(v_k \mid S_{k-1}) \\ &\geq \sum_{v_k \in T} \left(\rho \cdot \sum_{i=1}^{d} c_i(v_k) \right) = \rho \cdot \sum_{i=1}^{d} c_i(T) \geq \frac{\lambda\rho}{\lambda+1} \ , \end{aligned}$$

where the first inequality follows from the submodularity of f, the second inequality follows from the definition of v_k, and the last inequality follows from the guarantee of Lemma H.1. □

Handling the case in which the event E does not happen is somewhat more involved. Towards this goal, let us recursively define a set O_k for every $0 \leq k \leq \ell$. The base of the recursion is that for $k = \ell$

we define $O_\ell = OPT \setminus (B \cup \{u \in OPT \mid f(u \mid S_\ell) < \rho \cdot \sum_{i=1}^{d} c_i(u)\})$. Assuming O_{k+1} is already defined for some $0 \leq k < \ell$, we define O_k as follows. Let $D_k = \{u \in O_{k+1} \setminus S_k \mid S_k + u \in \mathcal{I}\}$. If $|D_k| \leq p$, we define $O_k = O_{k+1} \setminus D_k$. Otherwise, we let D'_k be an arbitrary subset of D_k of size p, and we define $O_k = O_{k+1} \setminus D'_k$.

Lemma H.5. *Assuming E does not happen, $O_0 = \varnothing$.*

Proof. We prove by a downward induction the stronger claim that $|O_k| \leq pk$ for every $0 \leq k \leq \ell$. To prove this inequality for $k = \ell$, we note that the fact that E did not happen and still Algorithm 3 terminated after ℓ iterations implies that no element of $O_\ell \setminus S_\ell$ can be added to S_ℓ without violating independence in $\mathcal{M}$. Therefore, S_ℓ is a base of $S_\ell \cup O_\ell$ in this p-system, which implies $|O_\ell| \leq p|S_\ell| = p\ell$ by the definition of a p-system since $O_\ell \subseteq OPT$ is independent in $\mathcal{M}$.

Assume now that the inequality $|O_{k+1}| \leq p(k+1)$ holds for some $0 \leq k < \ell$, and let us prove $|O_k| \leq pk$. There are two cases to consider. If $O_k = O_{k+1} \setminus D'_k$, then $|O_k| = |O_{k+1}| - |D'_k| \leq p(k+1) - p = pk$. Otherwise, if $O_k = O_{k+1} \setminus D_k$, then, by the definition of D_k, there are no elements of $O_k \setminus S_k$ that can be added to S_k without violating independence in $\mathcal{M}$. Therefore, S_k is a base of $S_k \cup O_k$ in this p-system, which implies $|O_k| \leq p|S_k| = pk$ by the definition of a p-system since $O_k \subseteq O_\ell \subseteq OPT$ is independent in $\mathcal{M}$. □

Corollary H.6. *Assuming E does not happen, $f(S_\ell) \geq \frac{f(O_\ell \cup S_\ell)}{p+1}$.*

Proof. We prove by induction the stronger claim that, for every integer $0 \leq k \leq \ell$,

$$f(S_k) \geq \frac{f(O_k \cup S_k)}{p+1} \quad . \tag{4}$$

For $k = 0$ this inequality follows from the non-negativity of f since $S_0 = O_0 = \varnothing$ by Lemma H.5. Assume now that Inequality (4) holds for some value $k-1$ obeying $0 \leq k-1 < \ell$, and let us prove it for k.

Consider the set $\Delta_k = O_k \setminus O_{k-1}$. The construction of O_{k-1} guarantees that every element of Δ_k can be added to S_{k-1} without violating independence in $\mathcal{M}$. Furthermore, since $\Delta_k \subseteq O_\ell$, every element $u \in \Delta_k$ also obeys $f(u \mid S_k) \geq f(u \mid S_\ell) \geq \rho \cdot \sum_{i=1}^{d} c_i(u)$. Therefore, every element of Δ_k obeys the condition of the loop on Line 4 of Algorithm 3 in the k-th iteration of the loop, and by the definition of v_k, this implies

$$\begin{aligned}
f(S_k) = f(S_{k-1}) + f(v_k \mid S_{k-1}) &\geq f(S_{k-1}) + \frac{f(v_k \mid S_{k-1}) + \sum_{u \in \Delta_k} f(u \mid S_{k-1})}{|\Delta_k| + 1} \\
&\geq \frac{f(O_{k-1} \cup S_{k-1})}{p+1} + \frac{f(\Delta_k + v_k \mid S_{k-1})}{|\Delta_k| + 1} \\
&\geq \frac{f(O_{k-1} \cup S_{k-1})}{p+1} + \frac{f(\Delta_k + v_k \mid S_{k-1})}{p+1} \geq \frac{f(O_k \cup S_k)}{p+1} \quad ,
\end{aligned}$$

where the second inequality follows from the induction hypothesis and the submodularity of f, the penultimate inequality follows from the monotonicity of f and the observation that the construction of O_{k-1} guarantees $|\Delta_k| \leq p$, and the last inequality follows again from the submodularity of f. □

To use the last corollary, we need a lower bound on $O_\ell \cup S_\ell$, which is given by the next lemma.

Lemma H.7. $f(O_\ell \cup S_\ell) \geq f(OPT) - f(S_B)/\alpha - \rho \cdot \left[d - \frac{|OPT \cap B|}{\lambda}\right]$.

Proof. Observe that

$$\begin{aligned}
f(O_\ell \cup S_\ell) &= f(OPT \setminus (B \cup \{u \in OPT \mid f(u \mid S_\ell) < \rho \cdot \sum\nolimits_{i=1}^{d} c_i(u)\}) \cup S_\ell) \qquad (5) \\
&\geq f(OPT) - f(OPT \cap B) - f(\{u \in OPT \setminus B \mid f(u \mid S_\ell) < \rho \cdot \sum\nolimits_{i=1}^{d} c_i(u)\} \mid S_\ell) \\
&\geq f(OPT) - f(S_B)/\alpha - f(\{u \in OPT \setminus B \mid f(u \mid S_\ell) < \rho \cdot \sum\nolimits_{i=1}^{d} c_i(u)\} \mid S_\ell) \quad ,
\end{aligned}$$

where the first inequality follows from the submodularity and monotonicity of f, and the second inequality follows from the definition of S_B. To lower bound the rightmost side of the last inequality, we need to upper bound the last term in it.

$$\begin{aligned} f(\{u \in OPT \setminus B \mid f(u \mid S_\ell) < \rho \cdot \textstyle\sum_{i=1}^{d} c_i(u)\} \mid S_\ell) &\leq \sum_{\substack{u \in OPT \setminus B \\ f(u \mid S_\ell) < \rho \cdot \sum_{i=1}^{d} c_i(u)}} f(u \mid S_\ell) \\ &\leq \rho \cdot \sum_{u \in OPT \setminus B} \sum_{i=1}^{d} c_i(u) = \rho \cdot \left[\sum_{u \in OPT} \sum_{i=1}^{d} c_i(u) - \sum_{u \in OPT \cap B} \sum_{i=1}^{d} c_i(u) \right] \\ &\leq \rho \cdot \left[d - \frac{|OPT \cap B|}{\lambda} \right] \ , \end{aligned}$$

where the last inequality holds since OPT is a feasible set and every element of B is big. Plugging the last inequality into Inequality (5) completes the proof of the lemma. □

We are now ready to lower bound the value of the solution produced by Algorithm 3. While reading the following proposition, it useful to keep in mind that $|OPT \cap B| \leq d(\lambda - 1)$ because (i) the feasibility of OPT implies $c_i(OPT \cap B) \leq 1$ for every integer $1 \leq i \leq d$ and (ii) every element $u \in OPT \cap B$ obeys $c_i(u) > \lambda^{-1}$ for at least one such i.

Proposition H.8. *Let*

$$\rho^* = \frac{f(OPT)}{(p + 1 + \alpha^{-1})/(1 + \lambda^{-1}) + d - |OPT \cap B|/\lambda} \ .$$

If $\rho \geq \rho^*$ and the event E happened, or $\rho \leq \rho^*$ and the event E did not happen, then Algorithm 3 outputs a solution of value at least $\lambda\rho^*/(\lambda + 1)$. Furthermore, if $\rho \in [(1 - \delta)\rho^*, \rho^*]$ for some value $\delta \in (0, 1]$, then, regardless of the realization of the event E, Algorithm 3 outputs a solution of value at least $(1 - \delta)\lambda\rho^*/(\lambda + 1)$. Finally, Algorithm 3 runs in $O(\lambda nd + nr + T_B)$ time, where T_B is the time complexity of `BigElementsAlg`*.*

Proof. If the event E happened, then Lemma H.4 guarantees that the output of Algorithm 3 is of value at least $\lambda\rho/(\lambda + 1)$. Therefore, to complete the proof of the proposition, it suffices to show that when the event E does not happen and $\rho^* \geq \rho$, the value of the output of the Algorithm 3 is at least $\lambda\rho^*/(\lambda + 1)$; and the rest of this proof is devoted to showing that this is indeed the case.

In the last case, Corollary H.6 and Lemma H.7 imply together

$$f(S_\ell) \geq \frac{f(OPT) - f(S_B)/\alpha - \rho \cdot \left[d - \frac{|OPT \cap B|}{\lambda} \right]}{p + 1} \ ,$$

and therefore, the output set of Algorithm 3 is of value at least

$$\begin{aligned} \max\{f(S_\ell), f(S_B)\} &\geq \frac{(p + 1) \cdot f(S_\ell) + \alpha^{-1} \cdot f(S_B)}{p + 1 + \alpha^{-1}} \geq \frac{f(OPT) - \rho(d - |OPT \cap B|/\lambda)}{p + 1 + \alpha^{-1}} \\ &\geq \frac{f(OPT) - \rho^*(d - |OPT \cap B|/\lambda)}{p + 1 + \alpha^{-1}} = \frac{\lambda\rho^*}{\lambda + 1} \ . \end{aligned}$$

□

In Appendix I.1 we show a modified version of Algorithm 3 that appears as Algorithm 9, accepts a quality control parameter $\varepsilon \in (0, 1/4)$ and employs the thresholding speedup technique due to Badanidiyuru and Vondrák [4]. Formally, the properties of Algorithm 9 are given by the following variant of Proposition H.8.

Proposition H.9. *Let*

$$\rho^* = \frac{(1 - \varepsilon)f(OPT)}{((1 + \varepsilon)p + 1 + \alpha^{-1})/(1 + \lambda^{-1}) + d - |OPT \cap B|/\lambda} \ .$$

There exists an event $\tilde{E}$ such that, if $\rho \geq \rho^*$ and the event $\tilde{E}$ happened, or $\rho \leq \rho^*$ and the event $\tilde{E}$ did not happen, then Algorithm 9 outputs a solution of value at least $\lambda\rho^*/(\lambda + 1)$. Furthermore, if $\rho \in [(1 - \delta)\rho^*, \rho^*]$ for some value $\delta \in (0, 1]$, then, regardless of the realization of the event $\tilde{E}$, Algorithm 9 outputs a solution of value at least $(1 - \delta)\lambda\rho^*/(\lambda + 1)$. Finally, Algorithm 9 runs in $O(\lambda nd + n\varepsilon^{-1}(\log n + \log \varepsilon^{-1}) + T_B)$ time, where T_B is the time complexity of `BigElementsAlg`*.*

H.2 Guessing ρ

To use Algorithm 3 (or the faster Algorithm 9), one must supply a value for ρ. Furthermore, according to Proposition H.8, it is best if the supplied value of ρ is close to ρ^* (throughout this section, unless stated explicitly otherwise, ρ^* refers to its value as defined in Proposition H.8). In this section we present an algorithm that manages to supply such a value for ρ using binary search. However, before presenting this algorithm, we first need to find a relatively small range which is guaranteed to include ρ^*. Let $\underline{\alpha}$ be a known lower bound on the value of α.

Observation H.10. *It always holds that*

$$\frac{1}{p+1+\underline{\alpha}^{-1}+d} \leq \frac{\rho*}{\max_{u\in\mathcal{N}} f(u)} \leq \frac{2n}{p} \quad .$$

Proof. According to the definition of ρ^*,

$$\begin{aligned}\rho^* &= \frac{f(OPT)}{(p+1+\alpha^{-1})/(1+\lambda^{-1})+d-|OPT\cap B|/\lambda} \\ &\leq \frac{n\cdot\max_{u\in\mathcal{N}} f(\{u\})}{p/(1+\lambda^{-1})+d-|OPT\cap B|/\lambda} \leq \frac{2n\cdot\max_{u\in\mathcal{N}} f(\{u\})}{p} \quad ,\end{aligned}$$

where the first inequity follows from the submodularity and non-negativity of f, and the second inequality follows from the upper bound on $|OPT\cap B|$ given in the discussion before Proposition H.8 and the inequality $\lambda \geq 1$.

Similarly,

$$\begin{aligned}\rho^* &= \frac{f(OPT)}{(p+1+\alpha^{-1})/(1+\lambda^{-1})+d-|OPT\cap B|/\lambda} \\ &\geq \frac{\max_{u\in\mathcal{N}} f(\{u\})}{(p+1+\alpha^{-1})/(1+\lambda^{-1})+d-|OPT\cap B|/\lambda} \geq \frac{\max_{u\in\mathcal{N}} f(\{u\})}{p+1+\underline{\alpha}^{-1}+d} \quad ,\end{aligned}$$

where the first inequality holds since every singleton is a feasible set by our assumptions, and the second inequality holds since $1+\lambda^{-1} \geq 1$ and $|OPT\cap B|/\lambda \geq 0$. □

We are now ready to present, as Algorithm 8, the algorithm that avoids the need to guess ρ. This algorithm gets a quality control parameter $\delta \in (0,1)$ in addition to the parameter λ of Algorithm 3. In Algorithm 8 we use the shorthand $\rho(i) \triangleq (1+\delta)^i \cdot \max_{u\in\mathcal{N}} f(\{u\})/(p+1+\underline{\alpha}^{-1}+d)$.

Algorithm 8: ρ `Guessing Algorithm`(λ,δ)

Let $\underline{i} \leftarrow 0$, $\bar{i} \leftarrow \left\lceil \log_{1+\delta}\frac{2n}{p} - \log_{1+\delta}\frac{1}{p+1+\underline{\alpha}^{-1}+d}\right\rceil$ and $k \leftarrow 0$.
while $\bar{i}-\underline{i} > 1$ **do**
  Update $k \leftarrow k+1$.
  Let $i_k \leftarrow \lceil(\underline{i}+\bar{i})/2\rceil$.
  Execute Algorithm 3 with $\rho = \rho(i_k)$. Let A_k denote the output set of this execution of Algorithm 3, and let E_k denote the event E for the execution.
  if *the event* E_k *happened* **then** Update $\underline{i} \leftarrow i_k$.
  else Update $\bar{i} \leftarrow i_k$.
Execute Algorithm 3 with $\rho = \rho(\underline{i})$. Let A' denote the output set of this execution of Algorithm 9.
return *the set maximizing* f *in* $\{A'\} \cup \{A_{k'} \mid 1 \leq k' \leq k\}$.

Proposition H.11. *Algorithm 8 has a time complexity of* $O((\lambda nd+nr+T_B)\cdot(\log\delta^{-1}+\log(\log n+\log(\underline{\alpha}^{-1}+d)))$ *and outputs a set of value at least*

$$\frac{(1-\delta)\lambda\rho^*}{\lambda+1} \geq \frac{(1-\delta)\lambda\cdot f(OPT)}{\lambda(p+1+\alpha^{-1})+(\lambda+1)(d-|OPT\cap B|/\lambda)} \quad .$$

Proof. The number of iterations done by Algorithm 8 is at most

$$\begin{aligned}
&2 + \log_2 \left[\log_{1+\delta} \frac{2n}{p} - \log_{1+\delta} \frac{1}{p + 1 + \underline{\alpha}^{-1} + d} \right] \\
&= 2 + \log_2[\ln(2n) - \ln(p) + \ln(p + 1 + \underline{\alpha}^{-1} + d)] - \log_2 \ln(1+\delta) \\
&\leq 3 + \log_2[\ln n + \ln 2 + \ln(2 + \underline{\alpha}^{-1} + d)] + \log_2 \delta^{-1} \\
&= O(\log \delta^{-1} + \log(\log n + \log(\underline{\alpha}^{-1} + d)))\ .
\end{aligned}$$

The time complexity guaranteed by the proposition now follows by multiplying the last expression with the time complexity of Algorithm 3 given by Proposition H.8.

It remains to prove the approximation guarantee stated in the proposition. Assume towards a contradiction that this approximation guarantee does not hold, and let us show that this results in a contradiction. Our first objective is to show that this assumption implies that the inequality

$$\rho(\underline{i}) \leq \rho^* \leq \rho(\bar{i}) \tag{6}$$

holds throughout the execution of Algorithm 8. Immediately after the initialization of $\underline{i}$ and $\bar{i}$, this inequality holds by Observation H.10. Assume now that Inequality (6) held before some iteration of the loop of Algorithm 8, and let us explain why it holds also after the iteration. By Proposition H.8, our assumption (that the approximation guarantee does not hold) implies that, in every iteration of the loop of Algorithm 8, the event E_k happened if and only if $\rho(i_k) < \rho^*$, and therefore, Algorithm 8 updates $\underline{i}$ in the iteration when $\rho(i_k) < \rho^*$ and updates $\bar{i}$ in the iteration when $\rho(i_k) \geq \rho^*$.

By Inequality (6) and the observation that $\bar{i} - \underline{i} \leq 1$ when Algorithm 8 terminates, we get

$$(1-\delta)\rho^* \leq (1-\delta)\rho(\bar{i}) \leq \rho(\underline{i}) \leq \rho^*\ .$$

However, the last inequality implies $f(A') \geq (1-\delta)\lambda\rho^*/(\lambda+1)$ by Proposition H.8, and hence, completes the proof. □

In Appendix I.2 we give, as Algorithm 10, a modified version of Algorithm 8 that is based on Algorithm 9 instead of Algorithm 3. We prove for this algorithm the following proposition.

Proposition H.12. *Algorithm 10 has a time complexity of* $O((\lambda nd + n\varepsilon^{-1}(\log n + \log \varepsilon^{-1}) + T_B) \cdot (\log \delta^{-1} + \log(\log n + \log(\underline{\alpha}^{-1} + d)))$ *and outputs a set of value at least*

$$\frac{(1-\delta)\lambda\rho^*}{\lambda+1} \geq \frac{(1-\delta-\varepsilon)\lambda \cdot f(OPT)}{\lambda((1+\varepsilon)p + 1 + \alpha^{-1}) + (\lambda+1)(d - |OPT \cap B|/\lambda)}\ ,$$

where ρ^* *represents here the value stated in Proposition H.9.*

H.3 Proof of Theorem 5.1

In this section we use the machinery developed so far to prove Theorem 5.1. To use Algorithms 8 or 10 one must choose the algorithm `BigElementsAlg`. A very simple choice is to use an algorithm that just returns the best singleton subset of B. This leads to the following theorem.

Theorem H.13. *For every value* $\varepsilon \in (0, 1/4]$*, there is a polynomial time* $[(1+O(\varepsilon))(p+1+\frac{7}{4}d)]^{-1}$*-approximation algorithm for* `SMKS` *that runs in* $\tilde{O}(nd + n/\varepsilon)$ *time.*

Proof. Given the above choice for `BigElementsAlg`, we can set $\alpha = |OPT \cap B|^{-1}$ because the submodularity of f implies

$$\max_{u \in B} f(\{u\}) \geq \frac{\sum_{u \in OPT \cap B} f(\{u\})}{|OPT \cap B|} \geq \frac{f(OPT \cap B)}{|OPT \cap B|}\ .$$

Thus, for $\lambda \geq 2$, it is valid to choose $\underline{\alpha}^{-1} = d(\lambda - 1)$ because of the upper bound on $|OPT \cap B|$ explained in the discussion before Proposition H.8.

If we now choose to use Algorithm 10 and set $\lambda = 2$ and $\delta = \varepsilon$, we get by Proposition H.12 that the inverse of the approximation ratio of the algorithm we consider is at most

$$\begin{aligned}
&\frac{\lambda((1+\varepsilon)p + 1 + \alpha^{-1}) + (\lambda+1)(d - |OPT \cap B|/\lambda)}{(1-\delta-\varepsilon)\lambda} \\
&\qquad = \frac{(1+\varepsilon)p + 1 + \alpha^{-1}(1-\lambda^{-1}-\lambda^{-2}) + d(1+\lambda^{-1})}{1-\delta-\varepsilon} \\
&\qquad \leq \frac{(1+\varepsilon)p + 1 + d(\lambda-1)(1-\lambda^{-1}-\lambda^{-2}) + d(1+\lambda^{-1})}{1-\delta-\varepsilon} \\
&\qquad = \frac{(1+\varepsilon)p + 1 + d(\lambda - 1 + \lambda^{-1} + \lambda^{-2})}{1-\delta-\varepsilon} = \frac{(1+\varepsilon)p + 1 + \frac{7}{4}d}{1-\delta-\varepsilon} \\
&\qquad \leq (1+3\delta+3\varepsilon)(p+1+\tfrac{7}{4}d) = (1+6\varepsilon)(p+1+\tfrac{7}{4}d) \ ,
\end{aligned}$$

and its time complexity is

$$\begin{aligned}
&O((\lambda nd + n\varepsilon^{-1}(\log n + \log \varepsilon^{-1}) + T_B) \cdot (\log \delta^{-1} + \log(\log n + \log(\underline{\alpha}^{-1} + d))) \\
&= O((nd + n\varepsilon^{-1}(\log n + \log \varepsilon^{-1}) + T_B) \cdot (\log \varepsilon^{-1} + \log(\log n + \log(2d))) \\
&= \tilde{O}(nd + n/\varepsilon + T_B) = \tilde{O}(nd + n/\varepsilon) \ ,
\end{aligned}$$

where the last equality holds because `BigElementsAlg` can be implemented by simply scanning all the n possible singleton sets, which requires a time complexity of $T_B = O(n)$. □

A slightly more involved option for `BigElementsAlg` is an algorithm that enumerates over all sets of up to two big elements, and outputs the best such set that is feasible. This leads to the following theorem.

Theorem H.14. *For every value $\varepsilon \in (0, 1/4]$, there is a polynomial time $(p + 1.5556 + \frac{13}{9}d + \varepsilon)$-approximation algorithm for `SMKS` that runs in $\tilde{O}(n^2 + nd)$ time.*

Proof. In this proof we need the following known lemma.

Lemma H.15 (Lemma 2.2 of Feige et al. [20])**.** *Let $f : 2^{\mathcal{N}} \to \mathbb{R}$ be a submodular function, let A be an arbitrary subset of $\mathcal{N}$, and let $A(p)$ be a random subset of A containing every element of A with probability p (not necessarily independently). Then,*

$$\mathbb{E}[f(A(p))] \geq (1-p) \cdot f(\varnothing) + p \cdot f(A) \ .$$

Next, we show that one is allowed to choose $\alpha = \min\{2/|OPT \cap B|, 1\}$ for the above described `BigElementsAlg`. If $|OPT \cap B| \leq 2$ then this is trivial. Otherwise, by choosing R to be a uniformly random subset of $OPT \cap B$ of size 2, we get that the value of the output of `BigElementsAlg` is at least

$$\mathbb{E}[f(R)] \geq \left(1 - \frac{2}{|OPT \cap B|}\right) \cdot f(\varnothing) + \frac{2}{|OPT \cap B|} \cdot f(OPT \cap B) \geq \frac{2}{|OPT \cap B|} \cdot f(OPT \cap B) \ ,$$

where the first inequality follows from Lemma H.15 and the observation that R includes every element of $OPT \cap B$ with probability exactly $2/|OPT \cap B|$. Hence, it is valid to set $\underline{\alpha}^{-1} = \max\{d(\lambda-1)/2, 1\}$ because of the upper bound on $|OPT \cap B|$ explained in the discussion before Proposition H.8.

Consider now the algorithm obtained by plugging $\delta = \frac{\varepsilon}{2p+4d+4}$, $\lambda = 3$ and the above choice for `BigElementsAlg` into Algorithm 8. We would like to show that this algorithm has all the properties guaranteed by the theorem. By Proposition H.11, the time complexity of this algorithm is

$$\begin{aligned}
&O((\lambda nd + nr + T_B) \cdot (\log \delta^{-1} + \log(\log n + \log(\underline{\alpha}^{-1} + d))) \\
&= O((nd + nr + T_B) \cdot (\log(p+d) + \log \varepsilon^{-1} + \log(\log n + \log d)) \\
&= \tilde{O}(nr + nd + T_B) = \tilde{O}(n^2 + nd) \ ,
\end{aligned}$$

where the last equality holds since a brute force implementation of `BigElementsAlg` runs in $O(n^2)$ time.

It remains to analyze the approximation ratio of our suggested algorithm. Towards this goal, we need to consider a few cases. If $|OPT \cap B| \geq 2$, then, by Proposition H.11, the inverse of the approximation ratio of the above algorithm is no worse than

$$\begin{aligned}
&\frac{\lambda(p+1+\alpha^{-1}) + (\lambda+1)(d - |OPT \cap B|/\lambda)}{(1-\delta)\lambda} \\
&= \frac{p+1+|OPT \cap B| \cdot (1/2 - \lambda^{-1} - \lambda^{-2}) + d(1+\lambda^{-1})}{1-\delta} \\
&\leq \frac{p+1+d(\lambda-1) \cdot (1/2 - \lambda^{-1} - \lambda^{-2}) + d(1+\lambda^{-1})}{1-\delta} = \frac{p+1+d(\lambda/2 - 1/2 + \lambda^{-1} + \lambda^{-2})}{1-\delta} \\
&= \frac{p+1+\frac{13}{9}d}{1-\delta} \leq (1+2\delta)(p+1+\tfrac{13}{9}d) \leq p+1+\tfrac{13}{9}d+\varepsilon \ .
\end{aligned}$$

Otherwise, if $|OPT \cap B| \leq 1$, then, by the same proposition, the inverse of the approximation ratio of the above algorithm is at most

$$\begin{aligned}
&\frac{\lambda(p+1+\alpha^{-1}) + (\lambda+1)(d - |OPT \cap B|/\lambda)}{(1-\delta)\lambda} \leq \frac{p+2+d(1+\lambda^{-1})}{1-\delta} \\
&= \frac{p+2+\frac{4}{3}d}{1-\delta} \leq (1+2\delta)(p+2+\tfrac{4}{3}d) \leq p+2+\tfrac{4}{3}d+\varepsilon \ .
\end{aligned}$$

The above inequalities prove an approximation ratio which is a bit weaker than the one guaranteed by the theorem. To get exactly the approximation ratio guaranteed by the theorem, we need to be a bit more careful with the last case. First, for $|OPT \cap B| = 1$,

$$\begin{aligned}
&\frac{\lambda(p+1+\alpha^{-1}) + (\lambda+1)(d - |OPT \cap B|/\lambda)}{(1-\delta)\lambda} = \frac{p+2-1/\lambda-1/\lambda^2+d(1+\lambda^{-1})}{1-\delta} \\
&= \frac{p+\frac{14}{9}+\frac{4}{3}d}{1-\delta} \leq (1+2\delta)(p+1.5556+\tfrac{4}{3}d) \leq p+1.5556+\tfrac{4}{3}d+\varepsilon \ .
\end{aligned}$$

Handling the case $OPT \cap B = \varnothing$ is a bit more involved. Since there are no big elements in this case in OPT, there is no need to take S_B into account in the analysis of Algorithm 8. It can be observed that by repeating the analysis, but ignoring this set, we can get that the value of the output set of this algorithm is also lower bounded by

$$\frac{(1-\delta)\lambda \cdot f(OPT \setminus B)}{\lambda(p+1) + (\lambda+1)(d - |OPT \cap B|/\lambda)} \ ,$$

which in our case implies that the inverse of the approximation ratio of our algorithm is at most

$$\frac{\lambda(p+1)+d(\lambda+1)}{(1-\delta)\lambda} = \frac{p+1+d(1+\lambda^{-1})}{1-\delta} = \frac{p+1+\frac{4}{3}d}{1-\delta} \leq p+1+\tfrac{4}{3}d+\varepsilon \ . \qquad \square$$

Theorem 5.1 now follows immediately from Theorems H.13 and H.14.

Remark: It is natural to consider also candidates for `BigElementsAlg` that enumerate over larger subsets of B. However, this will require $\Omega(n^3)$ time, and is, therefore, of little interest as one can obtain a clean $(p+d+1)$-approximation for `SMKS` in $\tilde{O}(n^3)$ time (see Section 1.1).

I Fast Versions of Algorithms from Appendix H

I.1 Nearly Linear Time Version of Algorithm 3

In this section we present a version of Algorithm 3 that runs in nearly linear time. This version appears as Algorithm 9, and it gets a quality control parameter $\varepsilon \in (0, 1/4)$ (in addition to the parameters λ and ρ of Algorithm 3). The speedup in this version of the algorithm is obtained using a technique due to Badanidiyuru and Vondrák [4] which employs a decreasing threshold τ. In every iteration of the loop starting on Line 5 of Algorithm 9, the algorithm looks for elements whose marginal value exceeds this threshold. This guarantees that the element selected by the algorithm

Algorithm 9: `Nearly Linear Time General Algorithm`$(\lambda, \rho, \varepsilon)$

```
// Build the set of big elements, and find a candidate solution based on
   them.
```

1 Let $B \leftarrow \{u \in \mathcal{N} \mid \exists_{1 \le i \le d}\, c_i(u) > \lambda^{-1}\}$.

2 Let S_B be the output set of `BigElementsAlg`(B).

```
// Construct a solution from the small elements.
```

3 Let $S_0 \leftarrow \varnothing$, $k \leftarrow 0$, $M \leftarrow \max_{u \in \mathcal{N}} f(\{u\})$ and $\tau \leftarrow M$.

4 **while** $\tau \ge \varepsilon M/[(1+\varepsilon)n]$ **do**

5 **for** *every element* $u \in \mathcal{N} \setminus (S_k \cup B)$ **do**

6   **if** $S_k + u \in \mathcal{I}$ *and* $f(u \mid S_k) \ge \max\{\tau, \rho \cdot \sum_{i=1}^{d} c_i(u)\}$ **then**

7   Let $v_{k+1} \leftarrow u$, and let $S_{k+1} \leftarrow S_k + v_{k+1}$.

8   **if** $\max_{1 \le i \le d} c_i(S_{k+1}) \le 1$ **then** Increase k by 1.

9   **else return** *the output set of* `SetExtract`(λ, S_{k+1}).

10 Update $\tau \leftarrow \tau/(1+\varepsilon)$.

11 **return** *the better set among* S_B *and* S_k.

has an almost maximal marginal among the elements that can be selected because the previous iteration of the same loop already selected every element that could be selected when τ was larger.

As explained in Section H.1, the properties we would like to prove for Algorithm 9 are summarized by Proposition H.9, which we repeat here for convenience.

Proposition H.9. *Let*

$$\rho^* = \frac{(1-\varepsilon) f(OPT)}{((1+\varepsilon)p + 1 + \alpha^{-1})/(1+\lambda^{-1}) + d - |OPT \cap B|/\lambda} \ .$$

There exists an event $\tilde{E}$ such that, if $\rho \ge \rho^*$ and the event $\tilde{E}$ happened, or $\rho \le \rho^*$ and the event $\tilde{E}$ did not happen, then Algorithm 9 outputs a solution of value at least $\lambda\rho^*/(\lambda+1)$. Furthermore, if $\rho \in [(1-\delta)\rho^*, \rho^*]$ for some value $\delta \in (0,1]$, then, regardless of the realization of the event $\tilde{E}$, Algorithm 9 outputs a solution of value at least $(1-\delta)\lambda\rho^*/(\lambda+1)$. Finally, Algorithm 9 runs in $O(\lambda n d + n\varepsilon^{-1}(\log n + \log \varepsilon^{-1}) + T_B)$ time, where T_B is the time complexity of `BigElementsAlg`.

We begin the proof of Proposition H.9 with the following lemma, which proves that Algorithm 9 has the time complexity stated in the proposition.

Lemma I.1. *Algorithm 9 has a time complexity of $O(\lambda n d + n\varepsilon^{-1}(\log n + \log \varepsilon^{-1}) + T_B)$, where T_B is the time complexity of `BigElementsAlg`.*

Proof. The construction of the set B requires $O(dn)$ time, and therefore, the time complexity required for the entire Algorithm 9 except for the loop starting on Line 4 is $O(dn + T_B)$. In the rest of this proof we show that this loop requires $O(\lambda n d + n\varepsilon^{-1}(\log n + \log \varepsilon^{-1}))$ time, which implies the lemma.

The loop starting on Line 4 of Algorithm 9 runs at most the number of times that is required to decrease τ from M to $\varepsilon M/[(1+\varepsilon)n]$, which is

$$\lceil \log_{1+\varepsilon}(n/\varepsilon) \rceil + 1 \le \frac{\ln n - \ln \varepsilon}{\varepsilon/2} + 2 = O(\varepsilon^{-1}(\log n + \log \varepsilon^{-1})) \ .$$

Within each iteration of the loop on Line 4, the loop starting on Line 5 executes at most n times. To understand the time complexity of the iterations of the last loop, we can observe that each such iteration takes a constant time with the exception of the following operations.

- In Line 6 we need to calculate the sum $\sum_{i=1}^{d} c_i(u)$, which takes $O(d)$ time. However, we can pre-calculate this sum for all the elements of $\mathcal{N}$ in $O(nd)$ time.
- Checking the condition on Line 8 requires $O(d)$ time (assuming we maintain the values $c_i(S_k)$). However, this line is executed only when an element is added to the solution of the algorithm, which happens at most $O(r) = O(n)$ times.

- Executing $\texttt{SetExtract}(\lambda, S_{k+1})$ requires $O(\lambda|S_{k+1}|d) = O(\lambda rd) = O(\lambda nd)$ time. However, this procedure is executed at most once by Algorithm 9.

Combining all the above, we get that the loop starting on Line 4 of Algorithm 9 requires only $O(\lambda nd + n\varepsilon^{-1}(\log n + \log \varepsilon^{-1}))$ time. □

Like in Section H.1, we use ℓ below to denote the final value of the variable k. Furthermore, one can observe that the proof of Observation H.2 applies to Algorithm 9 up to some natural modifications, and therefore, we are guaranteed that the output set of Algorithm 9 is feasible. The rest of this section is devoted to bounding the approximation guarantee of this output set.

Let $\tilde{E}$ be the event that Algorithm 9 returns through Line 9. We consider separately the case in which the event $\tilde{E}$ happens and at the case in which it does not happen. When the event $\tilde{E}$ happens, the output set of Algorithm 3 is the output set of $\texttt{SetExtract}$. This set contains only high density elements and is large in terms of the linear constraints (by Lemma H.1), which provides a lower bound on its value. This argument can be formalized, leading to the following lemma, whose proof is omitted since it is analogous to the proof of Lemma H.4.

Lemma I.2. *If the event $\tilde{E}$ happens, then Algorithm 3 returns a solution of value at least $\frac{\lambda\rho}{\lambda+1}$.*

Handling the case in which the event $\tilde{E}$ does not happen is somewhat more involved. Towards this goal, we recursively define a set O_k for every $0 \leq k \leq \ell$ (note that the definition we give here is slightly different compared to the one given in Section H.1). The base of the recursion is that for $k = \ell$ we define $O_\ell = OPT \setminus (B \cup \{u \in OPT \mid f(u \mid S_\ell) < \max\{\rho \cdot \sum_{i=1}^{d} c_i(u)), \varepsilon M/n\}\}$. Assuming O_{k+1} is already defined for some $0 \leq k < \ell$, we define O_k as follows. Let $D_k = \{u \in O_{k+1} \setminus S_k \mid S_k + u \in \mathcal{I}\}$. If $|D_k| \leq p$, we define $O_k = O_{k+1} \setminus D_k$. Otherwise, we let D'_k be an arbitrary subset of D_k of size p, and we define $O_k = O_{k+1} \setminus D'_k$.

Lemma I.3. *Assuming $\tilde{E}$ does not happen, $O_0 = \varnothing$.*

Proof Sketch. The proof of this lemma is very similar to the proof of Lemma H.5. The only difference is that arguing why O_ℓ is a base of $S_\ell \cup O_\ell$ is a bit more involved now. Specifically, consider the last iteration of the loop starting on Line 4 of Algorithm 9. In this iteration the value of τ was at most $\varepsilon M/n$, and therefore, every element of O_ℓ would have been added to S_ℓ during this iteration unless this addition violates independence in $\mathcal{M}$. □

Using the last lemma, we can now prove the following corollary, which corresponds to Corollary H.6 from Section H.1.

Corollary I.4. *Assuming $\tilde{E}$ does not happen, $f(S_\ell) \geq \frac{f(O_\ell \cup S_\ell)}{(1+\varepsilon)p+1}$.*

Proof. We prove by induction the stronger claim that, for every integer $0 \leq k \leq \ell$,

$$f(S_k) \geq \frac{f(O_k \cup S_k)}{(1+\varepsilon)p+1} \quad . \tag{7}$$

For $k = 0$ this inequality follows from the non-negativity of f since $S_0 = O_0 = \varnothing$ by Lemma I.3. Assume now that Inequality (7) holds for some value $k-1$ obeying $0 \leq k-1 < \ell$, and let us prove it for k.

Consider the set $\Delta_k = O_k \setminus O_{k-1}$. Let u' be an element of Δ_k maximizing $f(u' \mid S_{k-1})$, and let τ' be the maximum value that τ takes in any iteration of the loop starting on Line 4 of Algorithm 9 that is not larger than $f(u' \mid S_k)$. Such a value exists because the inclusion $\Delta_k \subseteq O_\ell$ implies that every element $u \in \Delta_k$ obeys $f(u \mid S_{k-1}) \geq f(u \mid S_\ell) \geq \max\{\rho \cdot \sum_{i=1}^{d} c_i(u), \varepsilon M/n\}$. Observe now that u' cannot belong to S_ℓ because $u' \in \Delta_k \subseteq O_\ell$ guarantees $f(u' \mid S_\ell) > 0$, which implies that, during the iteration of of the loop starting on Line 4 that corresponds to τ', the element u' was not added to the solution of Algorithm 9. Let us study the reason that u' was not added. By the definition of τ', $f(u' \mid S_{k-1}) \geq \tau'$. Additionally, by the construction of O_{k-1}, every element of Δ_k can be added to S_{k-1} without violating independence in $\mathcal{M}$. These two facts imply that the reason that u' was not added must have been that, by the time Algorithm 9 considers the element u' in the iteration correspond to τ', the solution of Algorithm 9 already contained at least k elements. Since

up to this time the algorithms adds to its solution only elements whose marginal contribution is at least τ', this implies

$$f(S_k) - f(S_{k-1}) \geq \tau' \geq \frac{f(u' \mid S_{k-1})}{1+\varepsilon} \geq \frac{\sum_{u \in \Delta_k} f(u \mid S_{k-1})}{(1+\varepsilon)|\Delta_k|} \ .$$

Recall now that $f(S_k) - f(S_{k-1}) = f(v_k \mid S_{k-1})$. Combining this equality with the previous one, we get

$$\begin{aligned} f(S_k) &\geq f(S_{k-1}) + \frac{f(v_k \mid S_{k-1}) + \sum_{u \in \Delta_k} f(u \mid S_{k-1})}{(1+\varepsilon)|\Delta_k| + 1} \\ &\geq \frac{f(O_{k-1} \cup S_{k-1})}{(1+\varepsilon)p+1} + \frac{f(\Delta_k + v_k \mid S_{k-1})}{(1+\varepsilon)|\Delta_k|+1} \\ &\geq \frac{f(O_{k-1} \cup S_{k-1})}{(1+\varepsilon)p+1} + \frac{f(\Delta_k + v_k \mid S_{k-1})}{(1+\varepsilon)p+1} \geq \frac{f(O_k \cup S_k)}{(1+\varepsilon)p+1} \ , \end{aligned}$$

where the second inequality follows from the induction hypothesis and the submodularity of f, the penultimate inequality follows from the monotonicity of f and the observation that the construction of O_{k-1} guarantees $|\Delta_k| \leq p$, and the last inequality follows again from the submodularity of f. □

To use the last corollary, we need a lower bound on $O_\ell \cup S_\ell$, which is given by the next lemma (and corresponds to Lemma H.7).

Lemma I.5. $f(O_\ell \cup S_\ell) \geq (1-\varepsilon) f(OPT) - f(S_B)/\alpha - \rho \cdot \left[d - \frac{|OPT \cap B|}{\lambda}\right]$.

Proof. Observe that

$$\begin{aligned} &f(O_\ell \cup S_\ell) \qquad (8) \\ &\quad = f(OPT \setminus (B \cup \{u \in OPT \mid f(u \mid S_\ell) < \max\{\rho \cdot \textstyle\sum_{i=1}^d c_i(u), \varepsilon M/n\}\}) \cup S_\ell) \\ &\quad \geq f(OPT) - f(OPT \cap B) - f(\{u \in OPT \setminus B \mid f(u \mid S_\ell) < \rho \cdot \textstyle\sum_{i=1}^d c_i(u)\} \mid S_\ell) \\ &\qquad - f(\{u \in OPT \setminus B \mid f(u \mid S_\ell) < \varepsilon M/n\} \mid S_\ell) \\ &\quad \geq f(OPT) - f(S_B)/\alpha - f(\{u \in OPT \setminus B \mid f(u \mid S_\ell) < \rho \cdot \textstyle\sum_{i=1}^d c_i(u)\} \mid S_\ell) \\ &\qquad - f(\{u \in OPT \setminus B \mid f(u \mid S_\ell) < \varepsilon M/n\} \mid S_\ell) \ , \end{aligned}$$

where the first inequality follows from the submodularity and monotonicity of f, and the second inequality follows from the definition of S_B. To lower bound the rightmost side of the last inequality, we need to upper bound the two last terms in it.

$$\begin{aligned} f(\{u \in OPT \setminus B \mid f(u \mid S_\ell) < \rho \cdot \textstyle\sum_{i=1}^d c_i(u)\} \mid S_\ell) &\leq \sum_{\substack{u \in OPT \setminus B \\ f(u|S_\ell) < \rho \cdot \sum_{i=1}^d c_i(u)}} f(u \mid S_\ell) \\ &\leq \rho \cdot \sum_{u \in OPT \setminus B} \sum_{i=1}^d c_i(u) = \rho \cdot \left[\sum_{u \in OPT} \sum_{i=1}^d c_i(u) - \sum_{u \in OPT \cap B} \sum_{i=1}^d c_i(u)\right] \\ &\leq \rho \cdot \left[d - \frac{|OPT \cap B|}{\lambda}\right] \ , \end{aligned}$$

where the last inequality holds since OPT is a feasible set and every element of B is big. Additionally,

$$f(\{u \in OPT \setminus B \mid f(u \mid S_\ell) < \varepsilon M/n\} \mid S_\ell) \leq \sum_{\substack{u \in OPT \setminus B \\ f(u|S_\ell) < \varepsilon M/n}} f(u \mid S_\ell) \leq \varepsilon M \leq \varepsilon \cdot f(OPT) \ ,$$

where the second inequality holds since $|OPT \cap B|$ is a subset of $\mathcal{N}$, and therefore, is of size at most n; and the last inequality holds since every element of $\mathcal{N}$ is a feasible solution by our assumption. Plugging the last inequalities into Inequality (8) completes the proof of the lemma. □

We are now ready to prove Proposition H.9.

Proof of Proposition H.9. If the event $\tilde{E}$ happened, then Lemma I.2 guarantees that the output of Algorithm 9 is of value at least $\lambda\rho/(\lambda+1)$. Therefore, to complete the proof of the lemma, it suffices to show that when the event E does not happen and $\rho^* \geq \rho$, the value of the output of the Algorithm 9 is at least $\lambda\rho^*/(\lambda+1)$; and the rest of this proof is devoted to showing that this is indeed the case.

In the last case, Corollary I.4 and Lemma I.5 imply together

$$f(S_\ell) \geq \frac{(1-\varepsilon)f(OPT) - f(S_B)/\alpha - \rho \cdot \left[d - \frac{|OPT \cap B|}{\lambda}\right]}{(1+\varepsilon)p+1} \ ,$$

and therefore, the output set of Algorithm 9 is of value at least

$$\begin{aligned}
\max\{f(S_\ell), f(S_B)\} &\geq \frac{((1+\varepsilon)p+1) \cdot f(S_\ell) + \alpha^{-1} \cdot f(S_B)}{(1+\varepsilon)p+1+\alpha^{-1}} \\
&\geq \frac{(1-\varepsilon)f(OPT) - \rho(d - |OPT \cap B|/\lambda)}{(1+\varepsilon)p+1+\alpha^{-1}} \\
&\geq \frac{(1-\varepsilon)f(OPT) - \rho^*(d - |OPT \cap B|/\lambda)}{(1+\varepsilon)p+1+\alpha^{-1}} = \frac{\lambda\rho^*}{\lambda+1} \ . \qquad \square
\end{aligned}$$

I.2 Nearly-Linear Time Version of Algorithm 8

In this section we describe a fast version of Algorithm 8. This version appears as Algorithm 10. There are only three differences between the two algorithms: (1) Algorithm 10 uses Algorithm 9 instead of Algorithm 3, (2) the value of $\bar{\tau}$ is slightly higher in Algorithm 9, and (3) the definition $\rho(i)$ is slightly modified in Algorithm 9 to be $\rho(i) \triangleq (1-2\varepsilon)(1+\delta)^i \cdot \max_{u \in \mathcal{N}} f(\{u\})/(p+1+\underline{\alpha}^{-1}+d)$.

Algorithm 10: ρ `Guessing Algorithm`$(\lambda, \varepsilon, \delta)$

Let $\underline{i} \leftarrow 0$, $\bar{i} \leftarrow \left\lceil \log_{1+\delta} \frac{2n}{p} - \log_{1+\delta} \frac{1-2\varepsilon}{p+1+\underline{\alpha}^{-1}+d} \right\rceil$ and $k \leftarrow 0$.
while $\bar{i} - \underline{i} > 1$ **do**
  Update $k \leftarrow k+1$.
  Let $i_k \leftarrow \lceil (\underline{i} + \bar{i})/2 \rceil$.
  Execute Algorithm 9 with $\rho = \rho(i_k)$. Let A_k denote the output set of this execution of Algorithm 9, and let E_k denote the event $\tilde{E}$ for the execution.
  if *the event* E_k *happened* **then** Update $\underline{i} \leftarrow i_k$.
  else Update $\bar{i} \leftarrow i_k$.
Execute Algorithm 9 with $\rho = \rho(\underline{i})$. Let A' denote the output set of this execution of Algorithm 9.
return *the set maximizing* f *in* $\{A'\} \cup \{A_{k'} \mid 1 \leq k' \leq k\}$.

The following observation corresponds to Observation H.10. The proofs of the two observations are very similar.

Observation I.6. *For the value* ρ^* *stated in Proposition H.9,*

$$\frac{1-2\varepsilon}{p+1+\underline{\alpha}^{-1}+d} \leq \frac{\rho*}{\max_{u \in \mathcal{N}} f(u)} \leq \frac{2n}{p} \ .$$

Proof. According to the definition of ρ^*,

$$\rho^* = \frac{(1-\varepsilon)f(OPT)}{((1+\varepsilon)p+1+\alpha^{-1})/(1+\lambda^{-1}) + d - |OPT \cap B|/\lambda}$$

$$\leq \frac{(1-\varepsilon)n \cdot \max_{u \in \mathcal{N}} f(\{u\})}{(1+\varepsilon)p/(1+\lambda^{-1}) + d - |OPT \cap B|/\lambda}$$
$$\leq \frac{(1-\varepsilon)2n \cdot \max_{u \in \mathcal{N}} f(\{u\})}{(1+\varepsilon)p} \leq \frac{2n \cdot \max_{u \in \mathcal{N}} f(\{u\})}{p} \ ,$$

where the first inequity follows from the submodularity and non-negativity of f, and the second inequality follows from the upper bound on $|OPT \cap B|$ given in the discussion before Proposition H.8 and the inequality $\lambda \geq 1$.

Similarly,

$$\rho^* = \frac{(1-\varepsilon)f(OPT)}{((1+\varepsilon)p + 1 + \alpha^{-1})/(1+\lambda^{-1}) + d - |OPT \cap B|/\lambda}$$
$$\geq \frac{(1-\varepsilon)\max_{u \in \mathcal{N}} f(\{u\})}{((1+\varepsilon)p + 1 + \alpha^{-1})/(1+\lambda^{-1}) + d - |OPT \cap B|/\lambda}$$
$$\geq \frac{(1-\varepsilon)\max_{u \in \mathcal{N}} f(\{u\})}{(1+\varepsilon)p + 1 + \underline{\alpha}^{-1} + d} \geq \frac{(1-2\varepsilon)\max_{u \in \mathcal{N}} f(\{u\})}{p + 1 + \underline{\alpha}^{-1} + d} \ ,$$

where the first inequality holds since every singleton is a feasible set by our assumption, and the second inequality holds since $1 + \lambda^{-1} \geq 1$ and $|OPT \cap B|/\lambda \geq 0$. □

Given the last observation, the proof of Proposition H.12 is identical to the proof of Proposition H.11 up to the following changes.

- One has to use Proposition H.9 and Observation I.6 instead of Proposition H.8 and Observation H.10.
- The calculation needed to bound the number of iterations performed by the algorithm becomes a bit more involved. Specifically, we can upper bound it by

$$2 + \log_2 \left[\log_{1+\delta} \frac{2n}{p} - \log_{1+\delta} \frac{1-2\varepsilon}{p + 1 + \underline{\alpha}^{-1} + d} \right]$$
$$= 2 + \log_2[\ln(2n) - \ln(p) - \ln(1-2\varepsilon) + \ln(p + 1 + \underline{\alpha}^{-1} + d)] - \log_2 \ln(1+\delta)$$
$$\leq 4 + \log_2[\ln n + \ln 2 + \ln(2 + \underline{\alpha}^{-1} + d)] + \log_2 \delta^{-1}$$
$$= O(\log \delta^{-1} + \log(\log n + \log(\underline{\alpha}^{-1} + d))) \ .$$

J Additional Experiments

J.1 Location Summarization Subject to a Cardinality Constraint

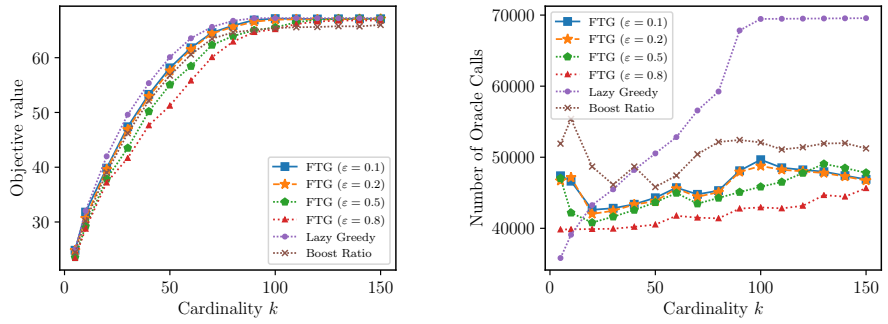
In this section we consider a location summarization application. The goal is to find a representative summary of tens of thousands of Yelp business locations (from Charlotte, Edinburgh, Las Vegas, Madison, Phoenix and Pittsburgh) by using their related attributes [6]. The representativeness of a business at location i for another business at a location j is quantified by a similarity measure $M_{i,j} = e^{-\lambda \cdot \text{dist}(v_i, v_j)}$, where v_i and v_j are vectors of attributes of the facilities at locations i and j. To select a good representation of all the locations $\mathcal{N} = \{1, \ldots, n\}$, we use the monotone and submodular facility location function [44] given by

$$f(S) = \frac{1}{n} \sum_{i=1}^{n} \max_{j \in S} M_{i,j} \ . \tag{9}$$

In Figure 2a, we observe that LAZYGREEDY performs slightly better than FASTTHRESHOLDGREEDY. On the other hand, Figure 2b shows that the query complexity of FASTTHRESHOLDGREEDY is much lower.

J.2 Comparing FASTTHRESHOLDGREEDY with STOCHASTICGREEDY

Recall that STOCHASTICGREEDY is a fast but randomized approach for maximizing submodular functions subject to a cardinality constraint [55]. In the next experiment, we compare FASTTHRESHOLDGREEDY with STOCHASTICGREEDY under the cardinality constraint. We consider



(a) (b)

Figure 2: Comparing FastThresholdGreedy with LazyGreedy and BoostRatio [46] under a cardinality constraint in Location Summarization.

the monotone and submodular vertex cover function. This function is defined over vertices of a (directed) graph $G = (V, E)$. For a given vertex set $S \subseteq V$, let $N(S)$ be the set of vertices which are pointed to by S, i.e., $N(S) \triangleq \{v \in V \mid \exists\, u \in S \text{ s.t. } (u, v) \in E\}$. The vertex cover function $f\colon 2^V \to \mathbb{R}_{\geq 0}$ is then defined as $f(S) = |N(S) \cup S|$. We use two different graphs: i) a random graph of $n = 10^6$ nodes with an average degree of 2, where 20 additional nodes with degrees 50 are added (the neighbors of these high degree nodes are chosen randomly); ii) Slashdot social network [51]. In Appendix J.2, we observe that the utility of FastThresholdGreedy is significantly better than that of StochasticGreedy. We also observe a high variability in the utility of solutions returned by StochasticGreedy. Furthermore, FastThresholdGreedy ($\varepsilon = 0.8$) outperforms StochasticGreedy ($\varepsilon \in \{0.1, 0.2\}$) in terms of both utility and query complexity. Note that, while StochasticGreedy is performing quite well in many practical scenarios, the theoretical guarantee of this algorithm holds only in expectation, and there are cases resulting in high variance. In these high variance cases, one has to run StochasticGreedy multiple times, which diminishes the benefit from the algorithm.

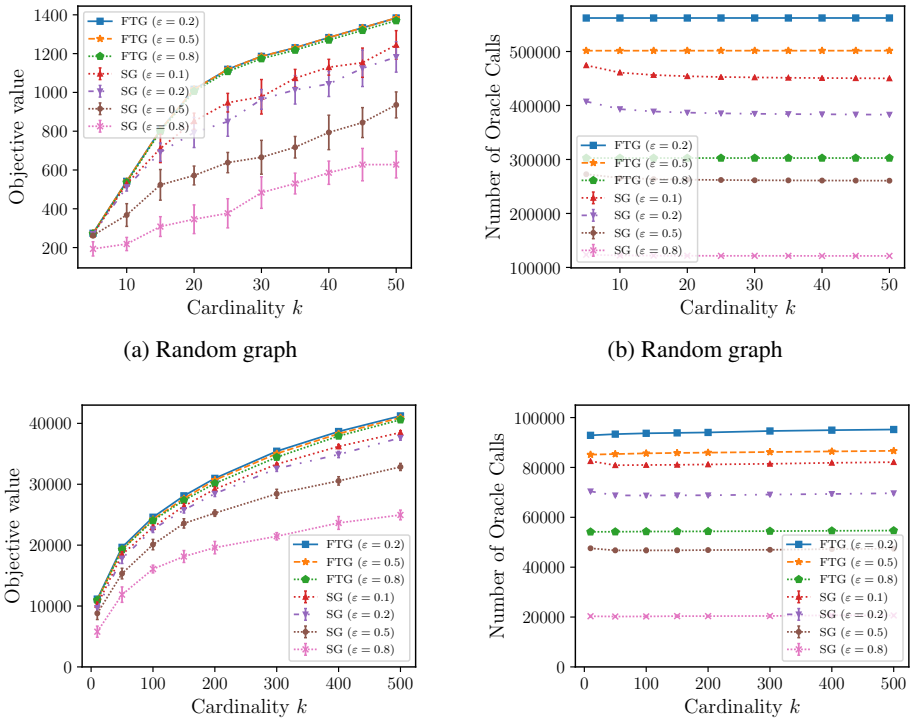
J.3 Twitter Text Summarization

In this section, we consider Twitter text summarization with the goal of producing a representative summary of Tweets around the first day of January 2019 [29]. In this application, the objective is to find a diverse summary from the Twitter feeds of news agencies. The monotone and submodular function f used is defined over a ground set $\mathcal{N}$ of tweets as follows: $f(S) = \sum_{w \in \mathcal{W}} \sqrt{\sum_{e \in S} \text{score}(w, e)}$, where $\mathcal{W}$ is the set of all the English words [42]. If word w is in Tweet e, then we have $\text{score}(w, e) = \{\text{number of retweets } e \text{ has received}\}$. Otherwise, we define $\text{score}(w, e) = 0$. The cost of each tweet is proportional to the time difference (in months) between the date of that Tweet and the first day of January 2019 [29]. Like in Section 6.2, it is evident from Figures 4a and 4b that Algorithm 2 surpasses the baseline algorithms.

J.4 p-Set System and d Knapsack Constraints

In this section we describe a set of experiments designed to compare the performance of Algorithm 10 with several other algorithms under the combination of a p-system and d knapsack constraints. We consider BarrierGreedy [6], Fast [4], Greedy and DensityGreedy as our baselines. Greedy keeps adding elements one by one according to their marginal gain as long as the p-system and knapsack constraints allow it. DensityGreedy is very similar to Greedy with the only difference being that it picks elements according to the ratio between their marginal gain and their total knapsack cost. Note that Greedy and DensityGreedy are heuristic algorithms without any theoretical guarantees for the setup of this experiment.

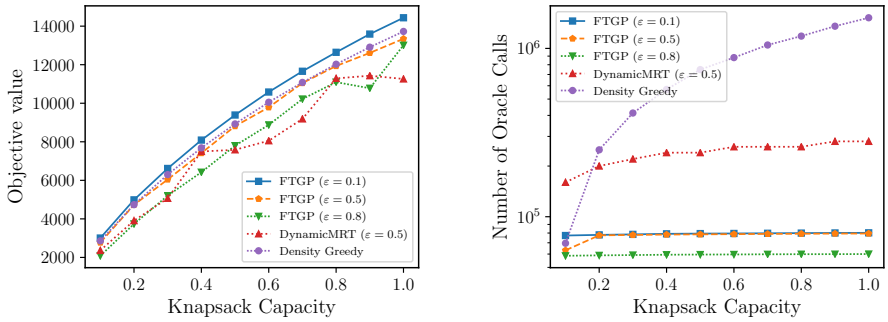
In the first experiment of this section, we again consider the location summarization application from Section J.1. The goal is to maximize (9) subject to the following constraints: (i) the maximum number of locations from each city is 5, (ii) the total size of the summary is at most 20, and (iii)



(a) Random graph (b) Random graph

(c) Slashdot network (d) Slashdot network

Figure 3: Comparing FASTTHRESHOLDGREEDY with STOCHASTICGREEDY on vertex cover problems under a cardinality constraint.

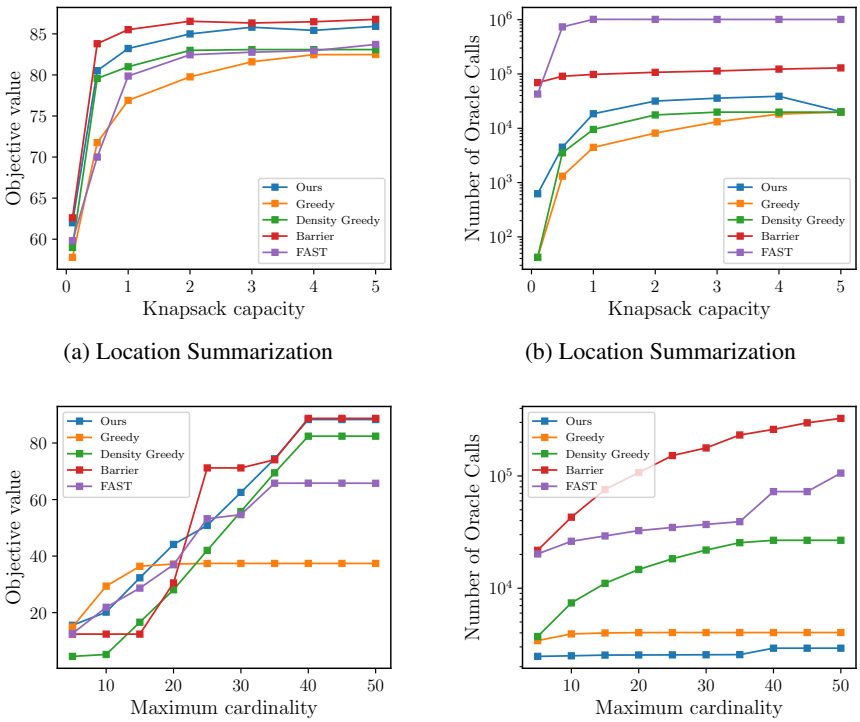


(a) (b)

Figure 4: Comparing Algorithm 2 (referred to as FTGP) with DYNAMICMRT [37] and DENSITY-GREEDY under a single knapsack constraint in Text Summarization.

two knapsacks constraints c_1 and c_2, where $c_i(j) = \text{distance}(j, \text{POI}_i)$ is the normalized distance of location j from a given location in its city (for c_1, POI_1 is the down-town; and for c_2, POI_2 is the international airport in that city). For ease of understanding the results of the experiments, the distances are normalized to make one unit of knapsack budget equivalent to 100km. Figures 5a and 5b compare algorithms for varying values of knapsack budget. In terms of utility, we observe that BARRIERGREEDY is the best performing algorithm followed by our algorithm (Algorithm 10). Despite the very competitive performance of Algorithm 10, its query complexity is almost an order of magnitude faster than that of BARRIERGREEDY.

The second application is a video summarization task over a collation of videos from the VSUMM dataset [15].[4] The features for each frame of a video are generated by a pre-trained ResNet-18 model [34, 43]. The similarity between two frames i and j is defined by $e^{-\lambda \cdot \mathrm{dist}(x_i, x_j)}$, where $\mathrm{dist}(x_i, x_j)$ is the Euclidean distance of the corresponding features of the frames. Similarly to the movie recommendation applications of Sections 6.1 and 6.2, the goal of the summarization task is to maximize the monotone and submodular function $f(S) = \log\det(\mathbf{I} + \alpha M_S)$ subject to the combination of the following constraints: (i) the maximum allowed cardinality of the final summary is k (in Figures 5c and 5d, we compare algorithms by varying this value), (ii) the maximum number of allowed frames from each video is 5, and (iii) a single knapsack constraint. The knapsack cost for each frame u is defined as $\mathrm{H}(u)/20$, where $\mathrm{H}(u)$ is the entropy of u. This extra knapsack constraint allows us to bound the required bits to encode the produced summary by using the entropy of each frame as a proxy for its encoding size. We again observe that BARRIERGREEDY and our algorithm (Algorithm 10) produce the summaries with the highest utilities (see Figure 5c). Furthermore, in Figure 5d, we observe that Algorithm 10 is the fastest algorithm. Particularly, Algorithm 10 is several orders of magnitudes faster than BARRIERGREEDY and FAST.



(a) Location Summarization (b) Location Summarization

(c) Video Summarization (d) Video Summarization

Figure 5: Comparing Algorithm 10 with the state-of-the-art algorithms subject to a combination of a p-system and d knapsack constraints.

[4]The dataset is available for download from `https://sites.google.com/site/vsummsite/`.