Fast Adaptive Non-Monotone Submodular Maximization Subject to a Knapsack Constraint*

Georgios Amanatidis† Federico Fusco‡ Philip Lazos‡ Stefano Leonardi‡

Rebecca Reiffenhäuser‡

July 25, 2020

Abstract

Constrained submodular maximization problems encompass a wide variety of applications, including personalized recommendation, team formation, and revenue maximization via viral marketing. The massive instances occurring in modern day applications can render existing algorithms prohibitively slow, while frequently, those instances are also inherently stochastic. Focusing on these challenges, we revisit the classic problem of maximizing a (possibly non-monotone) submodular function subject to a knapsack constraint. We present a simple randomized greedy algorithm that achieves a 5.83 approximation and runs in $O(n \log n)$ time, i.e., at least a factor n faster than other state-of-the-art algorithms. The robustness of our approach allows us to further transfer it to a stochastic version of the problem. There, we obtain a 9-approximation to the best adaptive policy, which is the first constant approximation for non-monotone objectives. Experimental evaluation of our algorithms showcases their improved performance on real and synthetic data.

1 Introduction

Constrained submodular maximization is a fundamental problem at the heart of discrete optimization. The reason for this is as simple as it is clear: submodular functions capture the notion of *diminishing returns* present in a wide variety of real-world settings.

Consequently to its striking importance and coinciding NP-hardness [20], extensive research has been conducted on submodular maximization since the seventies (e.g., [15, 42]), with focus lately shifting towards handling the massive datasets emerging in modern applications. With a wide variety of possible constraints, often regarding cardinality, independence in a matroid, or knapsack-type restrictions, the number of applications is vast. To name just a few, there are recent works on feature selection in machine learning [13, 14, 32], influence maximization in viral marketing [2, 31], and data summarization [43, 38, 45]. Many of these applications have *non-monotone* submodular objectives, meaning that adding an element to an existing set might actually decrease its value. Two such examples are discussed in detail in Section 5.

*This work was supported by the ERC Advanced Grant 788893 AMDROMA "Algorithmic and Mechanism Design Research in Online Markets" and the MIUR PRIN project ALGADIMAR "Algorithms, Games, and Digital Markets."

†Department of Mathematical Sciences, University of Essex, UK, and Institute for Logic, Language and Computation, University of Amsterdam, The Netherlands. Email: georgios.amanatidis@essex.ac.uk

‡Department of Computer, Control, and Management Engineering "Antonio Ruberti", Sapienza University of Rome, Italy. Email: {fuscof, lazos, leonardi, rebeccar}@diag.uniroma1.it

Modern-day applications increasingly force us to face two distinct, but often entangled challenges. First, the massive size of occurring instances fuels a need for very fast algorithms. As the running time is dominated by the *objective function evaluations* (also known as *value oracle calls*), it is typically measured (as in this work) by their number. So, here the goal is to design algorithms requiring an almost linear number of such evaluations. There is extensive research focusing on this issue, be it in the standard algorithmic setting [39], or in streaming [9, 3] and distributed submodular maximization [38, 12]. The second challenge is the inherent uncertainty in problems like sensor placement or revenue maximization, where one does not learn the exact marginal value of an element until it is added to the solution (and thus "paid for"). This, too, has motivated several works on adaptive submodular maximization [25, 26, 28, 41]. Note that even estimating the expected value to a partially unknown objective function can be very costly and this makes the reduction of the number of such calls all the more important.

Knapsack constraints are one of the most natural types of restriction that occurs in real-world problems and are often *hard* budget, time, or size constraints. Other combinatorial constraints like partition matroid constraints, on the other hand, model less stringent requirements, e.g., avoiding too many similar items in the solution. As the soft versions of such constraints can be often hardwired in the objective itself (see the *Video Recommendation* application in Section 5), we do not deal with them directly here.

The nearly-linear time requirement, without large constants involved, leaves little room for using sophisticated approaches like continuous greedy methods [23] or enumeration of initial solutions [44]. To further highlight the delicate balance between function evaluations and approximation, it is worth mentioning that, even for the monotone case, the first result combining $O(n \log n)$ oracle calls with an approximation better than 2 is the very recent $\frac{e}{e-1}$-approximation algorithm of Ene and Nguyen [16]. While this is a very elegant theoretical result, the huge constants involved render it unusable in practice.

At the same time, the strikingly simple, 2-approximation *modified densitiy greedy* algorithm of Wolsey [46] deals well with both issues in the *monotone* case: *Sort the items in decreasing order according to their marginal value over cost ratio and pick as many items as possible in that order without violating the constraint. Finally, return the best among this solution and the best single item.* When combined with lazy evaluations [37], this algorithm requires only $O(n \log n)$ value oracle calls and can be adjusted to work equally well for adaptive submodular maximization [25]. For *non-monotone* objectives, however, the only practical algorithm is the $(10 + \varepsilon)$-approximation FANTOM algorithm of Mirzasoleiman et al. [39] requiring $O(n^2 \log n)$ value oracle calls. Moreover, there is no known algorithm for the adaptive setting that can handle anything beyond a cardinality constraint [26].

We aim to tackle both aforementioned challenges for non-monotone submodular maximization under a knapsack constraint, by revisiting the simple algorithmic principle of Wolsey's density greedy algorithm. Our approach is along the lines of recent results on *random greedy* combinatorial algorithms [6, 24], which show that introducing randomness into greedy algorithms can extend their guarantees to the non-monotone case. Here, we give the first such algorithm for a knapsack constraint.

1.1 Contribution and Outline

The density greedy algorithm mentioned above may produce arbitrarily poor solutions when the objective is non-monotone. In this work we show that introducing some randomization leads to a simple algorithm, SampleGreedy, that outperforms existing algorithms both in theory and in practice. SampleGreedy flips a coin before greedily choosing any item in order to decide whether to include it to the solution or ignore it. The algorithmic simplicity of such an approach keeps SampleGreedy fast, easy to implement, and flexible enough to adjust to other related settings. At the same time the added randomness prevents it from getting trapped in solutions of poor quality.

In particular, in Section 3 we show that SampleGreedy is a 5.83-approximation algorithm using only $O(n \log n)$ value oracle calls. When all singletons have small value compared to an optimal solution, the approximation factor improves to almost 4. This is the first constant-factor approximation algorithm for the non-monotone case using this few queries. The only other algorithm fast enough to be suitable for large instances is the aforementioned FANTOM [39] which, for a knapsack constraint,[1] achieves an approximation factor of $(10 + \varepsilon)$ with $O(nr\varepsilon^{-1} \log n)$ queries, where r is the size of the largest feasible set and can be as large as $\Theta(n)$. Even if we modify FANTOM to use lazy evaluations, we still improve the query complexity by a logarithmic factor (see also Remark 1).

For the adaptive setting, where the stochastic submodular objective is learned as we build the solution, we show in Section 4 that a variant of our algorithm, AdaptiveGreedy, still guarantees a 9-approximation to the *best adaptive policy*. This is not only a relatively small loss given the considerably stronger benchmark, but is in fact the first constant approximation known for the problem in the adaptive submodular maximization framework of Golovin and Krause [25] and Gotovos et al. [26]. Hence we fill a notable theoretical gap, given that models with incomplete prior information or those capturing evolving settings are becoming increasingly important in practice.

From a technical point of view, our algorithm combines the simple principle of always choosing a high-density item with maintaining a careful exploration-exploitation balance, as is the case in many stochastic learning problems. It is therefore directly related to the recent simple randomized greedy approaches for maximizing non-monotone submodular objectives subject to other (i.e., non-knapsack) constraints [6, 9, 24]. However, there are underlying technical difficulties that make the analysis for knapsack constraints significantly more challenging. Every single result in this line of work critically depends on making a random choice in each step, in a way so that "good progress" is consistently made. This is not possible under a knapsack constraint. Instead, we argue globally about the value of the SampleGreedy output via a comparison with a carefully maintained *almost integral* solution. When it comes to extending this approach to the adaptive non-monotone submodular maximization framework, we crucially use the fact that the algorithm builds the solution iteratively, committing in every step to all the past choices. This is the main technical reason why it is not possible to adjust algorithms with multiple "parallel" runs, like FANTOM, to the adaptive setting.

Our algorithms provably handle well the aforementioned emerging, modern-day challenges, i.e., stochastically evolving objectives and rapidly growing real-world instances. In Section 5 we showcase the fact that our theoretical results indeed translate into applied performance. We focus on two applications that fit within the framework of non-monotone submodular maximization subject to a knapsack constraint, namely *video recommendation* and *influence-and-exploit marketing*. We run experiments on real and synthetic data that indicate that SampleGreedy consistently performs better than FANTOM while being much faster. For AdaptiveGreedy we highlight the fact that its adaptive behavior results in a significant improvement over non-adaptive alternatives.

1.2 Related Work

There is an extensive literature on submodular maximization subject to knapsack or other constraints, going back several decades, see, e.g., [42, 46]. For a *monotone* submodular objective

[1] FANTOM can handle more general constraints, like a p-system constraint and ℓ knapsack constraints. Here we refer to its performance and running time when restricted to a single knapsack constraint.

subject to a knapsack constraint there is a deterministic $\frac{e}{e-1}$-approximation algorithm [33, 44] which is tight, unless P = NP [20].

On non-monotone submodular functions Lee et al. [35] provided a 5-approximation algorithm for k knapsack constraints, which was the first constant factor algorithm for the problem. Fadaei et al. [17] building on the approach of Lee et al. [35], reduced this factor to 4. One of the most interesting algorithms for a single knapsack constraint is the 6-approximation algorithm of Gupta et al. [27]. As this is a greedy combinatorial algorithm based on running Sviridenko's algorithm twice, it is often used as a subroutine by other algorithms in the literature, e.g., [12], despite its running time of $O(n^4)$. A number of continuous greedy approaches [23, 34, 8] led to the current best factor of e when a knapsack—or even a general downwards closed—constraint is involved. However, continuous greedy algorithms are impractical for most real-world applications. The fastest such algorithm for our setting is the $(e+\varepsilon)$-approximation algorithm of Chekuri et al. [10] requiring $O(n^3\varepsilon^{-4}\text{polylog}(n))$ function evaluations. Possibly the only algorithm that is directly comparable to our SampleGreedy in terms of running time is FANTOM by Mirzasoleiman et al. [39]. FANTOM achieves a $(1+\varepsilon)(p+1)(2p+2\ell+1)/p$-approximation for ℓ knapsack constraints and a p-system constraint in time $O(nrp\varepsilon^{-1}\log(n))$, where r is the size of the largest feasible solution.

As mentioned above, there is a number of recent results on randomizing simple greedy algorithms so that they work for non-monotone submodular objectives [6, 9, 26, 24, 22]. Our paper extends this line of work, as we are the first to successfully apply this approach for a knapsack constraint.

Golovin and Krause [25] introduced the notions of adaptive monotonicity and submodularity and showed it is possible to achieve guarantees with respect to the optimal adaptive policy that are similar to the guarantees one gets in the standard algorithmic setting with respect to an optimal solution. Our Section 4 fits into this framework as it was generalized by Gotovos et al. [26] for non-monotone objectives; they showed that a variant of the random greedy algorithm of Buchbinder et al. [6] achieves a $\frac{e}{e-1}$-approximation in the case of a cardinality constraint.

Implicitly related to our quest for few value oracle calls is the recent line of work on the adaptive complexity of submodular maximization that measures the number of sequential rounds of independent value oracle calls needed to obtain a constant factor approximation; see [4, 5, 18, 19] and references therein. To the best of our knowledge, nothing nontrivial is known for non-monotone functions and a knapsack constraint.

2 Preliminaries

In this section we formally introduce the problem of submodular maximization with a knapsack constraint in both the standard and the adaptive setting.

Let $A = \{1, 2, ..., n\}$ be a set of n items.

Definition 1 (Submodularity)**.** A function $v : 2^A \to \mathbb{R}$ is *submodular* if and only if

$$v(S \cup \{i\}) - v(S) \geq v(T \cup \{i\}) - v(T)$$

for all $S \subseteq T \subseteq A$ and $i \notin T$.

A function $v : 2^A \to \mathbb{R}$ is *non-decreasing* (often referred to as *monotone*), if $v(S) \leq v(T)$ for any $S \subseteq T \subseteq A$. We consider general (i.e., not necessarily monotone), normalized (i.e., $v(\emptyset) = 0$), non-negative submodular valuation functions. Since *marginal* values are extensively used, we adopt the shortcut $v(T \,|\, S)$ for the marginal value of set T with respect to a set S, i.e. $v(T \,|\, S) = v(T \cup S) - v(S)$. If $T = \{i\}$ we write simply $v(i \,|\, S)$. While this is the most standard definition of submodularity in this setting, there are alternative equivalent definitions that will be useful later on.

Theorem 1 (Nemhauser et al. [42]). *Given a function $v : 2^A \to \mathbb{R}$, the following are equivalent:*

1. $v(i \,|\, S) \geq v(i \,|\, T)$ *for all* $S \subseteq T \subseteq A$ *and* $i \notin T$.
2. $v(S) + v(T) \geq v(S \cup T) + v(S \cap T)$ *for all* $S, T \subseteq A$.
3. $v(T) \leq v(S) + \sum_{i \in T \setminus S} v(i \,|\, S) - \sum_{i \in S \setminus T} v(i \,|\, S \cup T \setminus \{i\})$ *for all* $S, T \subseteq A$.

Moreover, we restate a key result which connects random sampling and submodular maximization. The original version of the theorem was due to Feige et al. [21], although here we use a variant from Buchbinder et al. [6].

Lemma 1 (Lemma 2.2. of Buchbinder et al. [6]). *Let $f : 2^A \to \mathbb{R}$ be a submodular set function, let $X \subseteq A$ and let $X(p)$ be a sampled subset, where each element of X appears with probability at most p (not necessarily independent). Then:*

$$\mathbb{E}\left[f(X(p))\right] \geq (1-p) f(\emptyset).$$

We assume access to a *value oracle* that returns $v(S)$ when given as input a set S. We also associate a positive cost c_i with each element $i \in A$ and consider a given budget B. The goal is to find a subset of A of maximum value among the subsets whose total cost is at most B. Formally, we want some $S^* \in \arg\max\{v(S) \,|\, S \subseteq A,\, \sum_{i \in S} c_i \leq B\}$. Without loss of generality, we may assume that $c_i \leq B$ for all $i \in A$, since any element with cost exceeding B is not contained in any feasible solution and can be discarded.

We next present the adaptive optimization framework. On a high level, here we do know how the world works and what situations occur with which probability. However, which of those we will be actually dealing with is inferred over time by the bits of information we learn. Along with set A, we introduce the *state space* Ω which is endowed with some probability measure. By $\omega = (\omega_i)_{i \in A} \in \Omega$ we specify the *state* of each element in A. The adaptive valuation function v is then defined over $A \times \Omega$; the value over a subset $S \subseteq A$ depends on both the subset and ω. Due to the probability measure over Ω, $v(S, \omega)$ is a random variable. We define $v(S) = \mathbb{E}\left[v(S, \omega)\right]$, the expectation being with respect to ω. Like before, the costs c_i are deterministic and known in advance.

For each $\omega \in \Omega$ and $S \subseteq A$, we define the partial realization of state ω on S as the couple $(S, \omega_{|S})$, where $\omega_{|S} = (\omega_i)_{i \in S}$. It is natural to assume that the true value of a set S does not depend on the whole state, but only on $\omega_{|S}$, i.e., $v(S, \omega) = v(S, \psi)$, for all $\omega, \psi \in \Omega$ such that $\omega_{|S} = \psi_{|S}$. Therefore, sometimes we overload the notation and use $v(S, \omega_{|S})$ instead of $v(S, \omega)$. There is a clear partial ordering on the set of all possible partial realizations: $(S, \omega_{|S}) \subseteq (T, \omega_{|T})$ if $S \subseteq T$ and $\omega_{|T}$ coincides with $\omega_{|S}$ over all the elements of S. The marginal value of an element i given a partial realization $(S, \omega_{|S})$ is

$$v(i \,|\, (S, \omega_{|S})) = \mathbb{E}\left[v(\{i\} \cup S, \omega) - v(S, \omega) \,|\, \omega_{|S}\right].$$

We are now ready to introduce the concepts of adaptive submodularity and monotonicity.

Definition 2. The function $v(\cdot\,,\cdot)$ is *adaptive submodular* if $v(i \,|\, (S, \omega_{|S})) \geq v(i | (T, \omega_{|T}))$ for all partial realizations $(S, \omega_{|S}) \subseteq (T, \omega_{|T})$ and for any $i \notin T$. Further, $v(\cdot\,,\cdot)$ is *adaptive monotone* if $v(i \,|\, (S, \omega_{|S})) \geq 0$ for all partial realizations $(S, \omega_{|S})$ and for all $i \notin S$.

In Section 4 we assume access to a value oracle that given an element i and a partial realization returns the expected marginal value of i. Using the properties of conditional expectation, it is straightforward to show that if $v(\cdot, \cdot)$ is adaptive submodular, then its expected value $v(\cdot)$ is submodular. In analogy with [26], we assume v to be state-wise submodular, i.e., $v(\cdot, \omega)$ is a submodular set function for each $\omega \in \Omega$.

In this framework it is possible to define *adaptive policies* to maximize v. An adaptive policy is a function which associates with every partial realization a distribution on the next element to be added to the solution. The optimal solution to the adaptive submodular maximization problem is to find an adaptive policy that maximizes the expected value while respecting the knapsack constraint (the expectation being taken over Ω and the randomness of the policy itself).

3 The Algorithmic Idea

We present and analyze SampleGreedy, a randomized 5.83-approximation algorithm for maximizing a submodular function subject to a knapsack constraint. As we mentioned already, SampleGreedy is based on the modified density greedy algorithm of Wolsey [46]. Since the latter may perform arbitrarily bad for non-monotone objectives, we add a sampling phase, similar to the sampling phase of the Sample Greedy algorithm of Feldman et al. [24].

SampleGreedy first selects a random subset A' of A by independently picking each element with probability p. Then it runs Wolsey's algorithm only on A'. To formalize this second step, using $v(i)$ as a shorthand for $v(\{i\})$, let

$$j_1 \in \arg\max_{i \in A'} v(i)/c_i,$$

while, for $k \geq 1$,

$$j_{k+1} \in \arg\max_{i \in A' \setminus \{j_1, \dots, j_k\}} v(i \,|\, \{j_1, \dots, j_k\})/c_i.$$

If ℓ is the largest integer such that $\sum_{i=1}^{\ell} c_{j_i} \leq B$, then $S = \{j_1, \dots, j_\ell\}$. In the end, the output is the one yielding the largest value between S and an element from $\arg\max_{i \in A'} v(i)$.

We formally present this algorithm in pseudocode below. Notice that to simplify the analysis, instead of selecting the entire set A' immediately at the start of the algorithm, we defer this decision and toss a coin with probability of success p each time an item is considered to be added to the solution. Both versions of the algorithm behave identically.

Theorem 2. *For $p = \sqrt{2} - 1$,* SampleGreedy *is a $\left(3 + 2\sqrt{2}\right)$-approximation algorithm.*

Proof. For the analysis of the algorithm we are going to use the auxiliary set O, an extension of the set S that respects the knapsack constraint and uses feasible items from an optimal solution. In particular, let S^* be an optimal solution and let $s_1, s_2, \dots, s_r$ be its elements.

Then, O is a *fuzzy* set that is initially equal to S^* and during each iteration of the *while* loop it is updated as follows:

- If $r_i = 1$, then $O = O \cup \{i\}$. In case this addition violates the knapsack constraint, i.e., $c(O) > B$, then we repetitively remove items from $O \setminus S$ in increasing order with respect to their cost until the cost of O becomes exactly B. Note that this means that the last item removed may be removed only partially. More precisely, if $c(O) > B$ and $c(O \setminus \{s_j\}) \leq B$, where s_j is the item of S^* of maximum index in $O \setminus S$, then we keep a $\left(B - c(O) + c_j\right)/c_j$ fraction of s_j in O and stop its update for the current iteration.

- Else (i.e., if $r_i = 0$), $O = O \setminus \{i\}$.

SampleGreedy$(A, v, \mathbf{c}, B)$

1 $i^* \in \arg\max_{k \in A} v(k)$ /* best single item */
2 $S = \emptyset$ /* greedy solution */
3 $F = \{k \in A \,|\, v(k \,|\, S) > 0 \text{ and } c_k \le B\}$ /* initial set of feasible items */
4 $R = B$ /* remaining knapsack capacity */
5 **while** $F \neq \emptyset$ **do**
6 Let $i \in \arg\max_{k \in F} \dfrac{v(k \,|\, S)}{c_k}$
7 Let $r_i \sim \text{Bernoulli}(p)$ /* independent random bit */
8 **if** $r_i = 1$ **then**
9 $S = S \cup \{i\}$
10 $R = R - c_i$
11 $A = A \setminus \{i\}$
12 $F = \{k \in A \,|\, v(k \,|\, S) > 0 \text{ and } c_k \le R\}$
13 **return** $\max\{v(i^*), v(S)\}$

If an item j was considered (in line 6) in some iteration of the *while* loop, then let S_j and O_j denote the sets S and O, respectively, at the beginning of that iteration. Moreover, let O'_j denote O at the end of that iteration. If j was never considered, then S_j and O_j (or O'_j) denote the final versions of S and O, respectively. In fact, in what follows we exclusively use S and O for their final versions.

It should be noted that, for all $j \in A$, $S_j \subseteq O_j$ and also no item in $O_j \setminus S_j$ has been considered in any of the previous iterations of the *while* loop.

Before stating the next lemma, let us introduce some notation for the sake of readability. Note that, by construction, $O \setminus S$ is either empty or consists of a single fractional item $\hat{\imath}$. In case $O \setminus S = \emptyset$, by $\hat{\imath}$ we denote the last item removed from O. For every $j \in A$, we define $Q_j = O_j \setminus (O'_j \cup S \cup \{\hat{\imath}\})$. Note that if j was never considered during the execution of the algorithm, then $Q_j = \emptyset$.

Lemma 2. *For every realization of the Bernoulli random variables, it holds that*

$$v(S \cup S^*) \le v(S) + v(\hat{\imath}) + \sum_{j \in A} c(Q_j) \frac{v(j \,|\, S_j)}{c_j}.$$

Proof of Lemma 2. Assume that the random bits $r_1, r_2, \dots$ are fixed. Also, without loss of generality, assume the items are numbered according to the order in which they are considered by SampleGreedy, with the ones not considered by the algorithm numbered arbitrarily (but after the ones considered). That is, item j—if considered—is the item considered during the j^{th} iteration.

Consider now any round j of the while loop of SampleGreedy. An item is removed from O_j in two cases. First, it could be item j itself that was originally in S^* but $r_j = 0$ (and hence it will never get back in O_k for any $k > j$). Second, it could be some other item that was in S^* and is taken out to make room for the new item j. In the latter case the only possibility for the removed item to return in O_k for some $k > j$ is to be selected by the algorithm and inserted in S. We can hence conclude that $Q_j \cap Q_k = \emptyset$ for all $j \neq k$. In addition to that, it is clear that $S \cup S^* = S \cup \{\hat{\imath}\} \cup_{j=1} Q_j$.

Therefore, if items $1, 2, \dots, \ell$ where all the items ever considered, using submodularity and the fact that $S_j \subseteq S \subseteq S \cup_{r=j+1}^{\ell} Q_r$, we have

$$\begin{aligned} v(S \cup S^*) - v(S) - v(\hat{\imath}) &\leq v((S \cup S^*) \setminus \hat{\imath}) - v(S) = \sum_{j=1}^{\ell} v\left(Q_j \,|\, S \cup_{r=j+1}^{\ell} Q_r\right) \\ &\leq \sum_{j=1}^{\ell} v\left(Q_j \,|\, S_j\right) \leq \sum_{j=1}^{\ell} \sum_{x \in Q_j} \frac{v(x \,|\, S_j)}{c_x} \cdot c_x \\ &\leq \sum_{j=1}^{\ell} \sum_{x \in Q_j} \frac{v(j \,|\, S_j)}{c_j} \cdot c_x = \sum_{j=1}^{\ell} \frac{v(j \,|\, S_j)}{c_j} \cdot \sum_{x \in Q_j} c_x \\ &= \sum_{j=1}^{\ell} \frac{v(j \,|\, S_j)}{c_j} \cdot c(Q_j)\,, \end{aligned} \tag{1}$$

where in a slight abuse of notation we consider c_x to be the fractional (linear) cost if $x \in Q_j$ is a fractional item. While the first three inequalities directly follow from the submodularity of v, for the last inequality we need to combine the optimality of $v(j \,|\, S_j)/c_j$ at the step j was selected with the fact that every single item x appearing in the sum $\sum_{j=1}^{\ell} \sum_{x \in Q_j} v(x \,|\, S_j)$ was feasible (as a whole item) at that step. The latter is true because of the way we remove items from O. If x is removed, it is removed before (any part of) $\hat{\imath}$ is removed. Thus, x is removed when the available budget is still at least $c_{\hat{\imath}}$. Given that $c_x \leq c_{\hat{\imath}}$, we get that x is feasible until removed.

To conclude the proof of the Lemma it is sufficient to note that $c(Q_j) = 0$ for all items that were not considered. □

While the previous Lemma holds for *each* realization of the random coin tosses in the algorithm, we next consider inequalities holding in expectation over the randomness of the $\{r_i\}_{i=1}^{|A|}$ in SampleGreedy. The indexing of the elements is hence to be considered deterministic and fixed in advance, not as in the proof of Lemma 2.

Lemma 3. $\mathbb{E}\left[\sum_{j \in A} c(Q_j)\, \frac{v(j \,|\, S_j)}{c_j}\right] \leq \frac{\max\{p, 1-p\}}{p}\, \mathbb{E}\left[v(S)\right]$

Proof of Lemma 3. For all $i \in A$, we define G_i to be the random gain because of i at the time i is added to the solution ($G_i = v(i \,|\, S_i)$ if i is added and 0 otherwise)

Since $v(S) = \sum_{i \in A} G_i$, by linearity, it suffices to show that the following inequalities hold in expectation over the coin tosses:

$$c(Q_i)\, \frac{v(i \,|\, S_i)}{c_i} \leq \frac{\max\{p, 1-p\}}{p}\, G_i, \quad \forall i \in A. \tag{2}$$

In order to achieve that, following [24], let $\mathcal{E}_i$ be any event specifying the random choices of the algorithm up to the point i is considered (if i is never considered, $\mathcal{E}_i$ captures all the randomness). If $\mathcal{E}_i$ is an event that implies i is not considered, then the Eq. (2) is trivially true, due to $G_i = 0$ and $Q_i = \emptyset$. We focus now on the case $\mathcal{E}_i$ implies that i is considered. Analyzing the algorithm, it is clear that

$$\mathbb{E}\left[c(Q_i) \,|\, \mathcal{E}_i\right] \leq \begin{cases} 0 \cdot \mathbb{P}\left(r_i = 1\right) + c_i \cdot \mathbb{P}\left(r_i = 0\right) = (1-p) \cdot c_i\,, & \text{if } i \in O_i\,, \\ c_i \cdot \mathbb{P}\left(r_i = 1\right) + 0 \cdot \mathbb{P}\left(r_i = 0\right) = p \cdot c_i\,, & \text{otherwise.} \end{cases} \tag{3}$$

In short, $\mathbb{E}\left[c(Q_i) \,|\, \mathcal{E}_i\right] \leq \max\{p, 1-p\} \cdot c_i$. It is here that we use the fuzziness of O: without the fractional items it would be hopeless to bound $c(Q_t)$ with c_t.

At this point, we exploit the fact that $\mathcal{E}_i$ contains the information on S_i, i.e., $S_i = S_i(\mathcal{E}_i)$ deterministically. Recall that S_i is the solution set at the time item i is considered by the algorithm.

$$\begin{aligned}\mathbb{E}\left[G_i \,|\, \mathcal{E}_i\right] &= \mathbb{P}\left(i \in S \,|\, \mathcal{E}_i\right) v(i \,|\, S_i) = \mathbb{P}\left(r_i = 1\right) v(i \,|\, S_i) = p \cdot c_i \,\frac{v(i \,|\, S_i)}{c_i} \\ &\geq \frac{p}{\max\{p, 1-p\}}\, \mathbb{E}\left[c(Q_i) \,|\, \mathcal{E}_i\right] \frac{v(i \,|\, S_i)}{c_i} \\ &= \frac{p}{\max\{p, 1-p\}}\, \mathbb{E}\left[c(Q_i)\, \frac{v(i \,|\, S_i)}{c_i} \,\Big|\, \mathcal{E}_i\right].\end{aligned}$$

We can hence conclude the proof by using the law of total probability over $\mathcal{E}_i$ and the monotonicity of the conditional expectation:

$$\begin{aligned}\mathbb{E}\left[G_i\right] = \mathbb{E}\left[\mathbb{E}\left[G_i \,|\, \mathcal{E}_i\right]\right] \geq \mathbb{E}\left[\frac{p}{\max\{p, 1-p\}}\, \mathbb{E}\left[c(Q_i)\, \frac{v(i \,|\, S_i)}{c_i} \,\Big|\, \mathcal{E}_i\right]\right] = \\ = \frac{p}{\max\{p, 1-p\}}\, \mathbb{E}\left[c(Q_i)\, \frac{v(i \,|\, S_i)}{c_i}\right].\end{aligned}$$

□

Lemma 4. $v(S^*) \leq \dfrac{1}{1-p}\, \mathbb{E}\left[v(S \cup S^*)\right]$.

Proof of Lemma 4. Let S^* be an optimal set for the constrained submodular maximization problem. We define $g : 2^A \to \mathbb{R}_+$ as follows: $g(B) = v(B \cup S^*)$. It is a simple exercise to see that such function is indeed submodular, moreover $g(\emptyset) = v(S^*)$. If we now apply Lemma 1 to g, observing that the elements in the set S output by the algorithm are chosen with probability at most p, we conclude that:

$$\mathbb{E}\left[v(S \cup S^*)\right] = \mathbb{E}\left[g(S)\right] \geq (1-p) g(\emptyset) = (1-p) v(S^*).$$

□

Combining Lemmata 2, 3 and 4 we get

$$\begin{aligned}(1-p) v(S^*) &\leq \mathbb{E}\left[v(S \cup S^*)\right] \\ &\leq \mathbb{E}\left[v(S) + v(\hat{\imath}\,) + \sum_{j \in A} c(Q_j)\, \frac{v(j \,|\, S_j)}{c_j}\right] \\ &\leq \mathbb{E}\left[v(S)\right] + v(i^*) + \frac{\max\{p, 1-p\}}{p}\, \mathbb{E}\left[v(S)\right] \\ &= \max\left\{2, \tfrac{1}{p}\right\} \cdot \mathbb{E}\left[v(S)\right] + v(i^*)\,. \end{aligned} \tag{4}$$

By substituting $\sqrt{2} - 1$ for p, this yields $v(S^*) \leq (3 + 2\sqrt{2}) \max\{\mathbb{E}\left[v(S)\right], v(i^*)\}$. This establishes the claimed approximation factor. □

A naive implementation of SampleGreedy needs $\Theta(n^2)$ value oracle calls in the worst case. Indeed, in each iteration all the remaining elements have their marginals updated and for large enough B the greedy solution may contain a constant fraction of A. Applying lazy evaluations [37], however, we can cut the number of queries down to $O(n\varepsilon^{-1} \log(n/\varepsilon))$ losing only an ε in the approximation factor (see also [16]). To achieve this, instead of recomputing all the marginals at every step, we maintain an ordered queue of the elements sorted by their last known *densities* (i.e., their marginal value per cost ratios) and use it to get a *sufficiently good* element to add.

More formally, the lazy implementation of SampleGreedy maintains the elements in a priority queue in decreasing order of density, which is initialised using the ratios $v(i)/c_i$. At each step we pop the element on top of the queue. If its density with respect to the current solution is within a $1+\varepsilon$ factor of its old one, then it is picked by the algorithm, otherwise it is reinserted in the queue according to its new density and we pop the next element. Submodularity guarantees that the density of a picked element is at least $1/(1+\varepsilon)$ of the best density for that step. As soon as an element has been updated $\log(n/\varepsilon)/\varepsilon$ times, we discard it.

Theorem 3. *The lazy version of* SampleGreedy *achieves an approximation factor of* $3+2\sqrt{2}+\varepsilon$ *using* $O(n\varepsilon^{-1}\log(n/\varepsilon))$ *value oracle calls.*

Proof. For a given $\varepsilon \in (0,1)$ let $\varepsilon' = \varepsilon/6$. We perform lazy evaluations using ε', with log denoting the binary logarithm.

It is straightforward to argue about the number of value oracle calls. Since the marginal value of each element i has been updated at most $\frac{\log(n/\varepsilon')}{\varepsilon'}$ times, we have a total of at most $n\frac{\log(n/\varepsilon')}{\varepsilon'} = O(\frac{n\log(n/\varepsilon)}{\varepsilon})$ function evaluations.

The approximation ratio is also easy to show. There are two distinct sources of loss in approximation. We first bound the total value of the discarded elements due to too many updates. This value appears as the upper bound of an extra additive term in Section 3. Indeed, now besides $\sum_{j=1}^{\ell} v\left(Q_j \,|\, S \cup_{r=j+1}^{\ell} Q_r\right)$ we need to account for the elements of O that were ignored because of too many updates. Such elements, once they become "inactive" do not contribute to the cost of the current O and are never pushed out as new elements come into S. The definition of the Q_js in the proof of Theorem 2 should be adjusted accordingly. That is, if W_j are the elements of O that become inactive because they were updated too many times during iteration j, we have

$$v((S\cup S^*)\setminus \hat{\imath}\,) - v(S) \le \sum_{j=1}^{\ell} v\left(Q_j \,|\, S_j\right) + \sum_{j=1}^{\ell} v\left(W_j \,|\, S_j\right).$$

However, by noticing that for $x \in (0,1)$ it holds that $x \le \log(1+x)$, we have

$$\begin{aligned}
\sum_{j=1}^{\ell} v\left(W_j \,|\, S_j\right) &\le \sum_{i\in\bigcup_j W_j} (1+\varepsilon')^{-\frac{\log(n/\varepsilon')}{\varepsilon'}} v(i) \\
&\le \sum_{i\in\bigcup_j W_j} (1+\varepsilon')^{-\frac{\log(n/\varepsilon')}{\log(1+\varepsilon')}} \max_{k\in A} v(k) \\
&\le \sum_{i\in A} (1+\varepsilon')^{-\log_{1+\varepsilon'}(n/\varepsilon')} v(S^*) \\
&= \sum_{i\in A} \frac{\varepsilon}{6n} v(S^*) = \frac{\varepsilon}{6} v(S^*)\,.
\end{aligned}$$

For the second source of loss in approximation, recall that the marginals only decrease due to submodularity. So, we know that if some item j is considered during iteration j (following the renaming of Lemma 2), then $(1+\varepsilon')v(j\,|\,S_j)/c_j \ge \arg\max_{k\in F} v(k\,|\,S_j)/c_k$. The only difference this makes (compared to the proof of Theorem 2) is that in the last inequality of Section 3 we have an extra factor of $1+\varepsilon'$.

Combining the above, we get the following analog of Lemma 2:

$$v(S \cup S^*) \leq v(S) + v(\hat{\imath}\,) + \frac{\varepsilon}{6} v(S^*) + \sum_{j \in A} \Big(1 + \frac{\varepsilon}{6}\Big) c(Q_j)\, \frac{v(j \,|\, S_j)}{c_j},$$

which carries over to Eq. (4), while Lemmata 3 and 4 are not affected at all. It is then a matter of simple calculations to see that for $p = \sqrt{2}-1$, we still get $v(S^*) \leq (3+2\sqrt{2}+\varepsilon) \max\{\mathbb{E}[v(S)], v(i^*)\}$. □

Additionally, our analysis implies that SampleGreedy performs significantly better in the *large instance* scenario, i.e., when the value of the optimal solution is much larger than the value of any single element. While it is not expected to have exact knowledge of the factor δ in the following proposition, often some estimate is accessible. Especially for massive instances, it is reasonable to assume that δ is bounded by a very small constant.

Theorem 4. *If* $\max_{i \in A} v(i) \leq \delta \cdot \text{OPT}$ *for* $\delta \in (0, 1/2)$*, then* SampleGreedy *with* $p = \frac{1-\delta}{2}$ *is a* $(4 + \varepsilon_\delta)$*-approximation algorithm, where* $\varepsilon_\delta = \frac{4\delta(2-\delta)}{(1-\delta)^2}$.

Proof. Starting from Eq. (4) and exploiting the large instance property, we get:

$$(1-p)v(S^*) \leq \max\left\{2, \tfrac{1}{p}\right\} \cdot \mathbb{E}[v(S)] + v(i^*) \leq \max\left\{2, \tfrac{1}{p}\right\} \cdot \mathbb{E}[v(S)] + \delta \cdot v(S^*).$$

Rearranging the terms and assuming $p + \delta < 1$, we have:

$$v(S^*) \leq \frac{\max\left\{2, \frac{1}{p}\right\}}{1 - p - \delta}.$$

Optimizing for $p \in (0, 1-\delta)$ we get the desired statement. □

4 Adaptive Submodular Maximization

In this section we modify SampleGreedy to achieve a good approximation guarantee in the adaptive framework. Recall that the adaptive valuation function $v(\cdot, \cdot)$ depends on the state of the system which is discovered a bit at a time, in an adaptive fashion. Indeed, SampleGreedy is compatible with this framework and can be applied nearly as it is. We stick to the interpretation of SampleGreedy discussed right before Theorem 2. That is, there is no initial sampling phase. Instead, we directly begin to choose greedily with respect the density (marginal value with respect to the current solution over cost). Each time we are about to pick an element of A, we throw a p-biased coin that determines whether we keep or discard the element.

Here the main difference with the greedy part of SampleGreedy is that the marginals are to be considered with respect to the *partial realization* relative to the current solution. Moreover, since it is not possible to return the largest between $\max_{i \in A} v(i)$ and the result of the greedy exploration, the choice between these two quantities has to be settled before starting the exploration. Formally, at the beginning of the algorithm a p_0-biased coin is tossed to decide between the two. The pseudo-code for the resulting algorithm, AdaptiveGreedy, is given below.

Before proving that AdaptiveGreedy works as promised, we need some observations. Let us denote by S the output of a run of our algorithm, and S^* the output of a run of the optimal adaptive strategy. Fix a realization $\omega \in \Omega$. Now, Lemma 1 of [26] implies

$$\mathbb{E}[v(S \cup S^*, \omega) | \omega] \geq (1-p) \cdot v(S^*, \omega).$$

```
AdaptiveGreedy
Let r_0 ~ Bernoulli(p_0)
if r_0 = 1 then
    i* ∈ arg max_{k∈A} v(k)                          /* best single item in expectation */
    Observe ω_{i*} and return v(i*, ω_{i*})
S = ∅, R = B                                          /* greedy solution and remaining knapsack capacity */
F = {k ∈ A | v(k) > 0}                                /* initial set of candidate items */
while F ≠ ∅ do
    Let i ∈ arg max_{k∈F} v(k | (S, ω_{|S})) / c_k
    Let r_i ~ Bernoulli(p)                            /* independent random bit */
    if r_i = 1 then
        Observe ω_i :  S = S ∪ {i},  R = R − c_i
    A = A \ {i},    F = {k ∈ A | v(k | (S, ω_{|S})) > 0 and c_k ≤ R}
return S, v(S, ω_{|S})
```

Since ω (and therefore, S^*) is fixed, the only randomness is due to the coin flips in our algorithm. We stress that the union between S and S^* has to be intended in the following sense: run our algorithm, and independently, also the optimal one, both for the same realization ω. The previous inequality is true for any ω. So, by the law of total probability, we also have

$$\mathbb{E}\left[v(S \cup S^*)\right] \geq (1-p) \cdot \mathbb{E}\left[v(S^*)\right]. \tag{5}$$

For the next observation, assume our algorithm has picked (and therefore observed) exactly set S. That is, we know only $\omega_{|S}$. We number all items $a \in A$ with positive marginal with respect to S by decreasing ratio $\frac{v\left(a|(S,\omega_{|S})\right)}{c_a}$, i.e.,

$$a_1 = \arg\max_{a \in A} \left\{ \frac{v\left(a|(S,\omega_{|S})\right)}{c_a} \right\},$$

and so on. Note that this captures a notion of the *best-looking items after already adding S*.

For $k = \min\{i \in \mathbb{N} | \sum_{l=1}^{i} c_l \geq B\}$, we get, in analogy to Lemma 1 of Gotovos et al. [26],

$$\sum_{i=1}^{k} v\left(a_i|(S,\omega_{|S})\right) \geq \mathbb{E}\left[\sum_{a \in S^*} v\left(a|(S,\omega_{|S})\right) |\omega_{|S}\right] \geq v\left(S^*|(S,\omega_{|S})\right). \tag{6}$$

Note that it could be the case that k is not well defined, as there may not be enough elements with positive marginal to fill the knapsack. If that is the case, just consider k to be the number of elements with positive marginals.

The point of Eq. (6) is that, given $(S, \omega_| S)$, the set of elements $a_1 \ldots a_k$ is deterministic, while S^* is not, because it corresponds to the set opened by the best adaptive policy. Moreover, in the middle term notice that the conditioning influences the valuation, but *not* the policy, since we are assuming to run it obliviously. This is fundamental for the analysis.

Since this holds for any set S, we can again generalize to the expectation over all possible runs of the algorithm (fixing the coin flips or not, as they only influence S; the best adaptive policy or the best-looking items $a_1, a_2, \dots, a_k$ are not affected). So, we get

$$\mathbb{E}\left[\sum_{i=1}^{k} v\left(a_i|(S,\omega_{|S})\right)\right] \geq \mathbb{E}\left[v\left(S^*|(S,\omega_{|S})\right)\right]. \tag{7}$$

We remark that k above is a random variable which depends on S. We use these observations to prove the ratio of our algorithm.

Theorem 5. *For $p_0 = 1/3$ and $p = 1/6$,* AdaptiveGreedy *yields a 9-approximation of* OPT_Ω*, while its* lazy *version achieves a $(9+\varepsilon)$-approximation using $O(n\varepsilon^{-1}\log(n/\varepsilon))$ value oracle calls.*

Moreover, when $\max_{i\in A} v(i) \leq \delta \cdot \text{OPT}_\Omega$ for $\delta \in (0,1/2)$, then for $p_0 = 0$ and $p = (\sqrt{3-2\delta}-1)/2$, AdaptiveGreedy *yields a $(4+2\sqrt{3}+\varepsilon'_\delta)$-approximation, where $\varepsilon'_\delta \approx \frac{6\delta(2-\delta)}{(1-\delta)^2}$.*

Proof. For any run of the algorithm, i.e., a fixed set S, the corresponding partial realization $\omega_{|S}$ and the coin flips observed, define for convenience the set C as those items in $\{a_1,\dots,a_k\}$ that have been considered during the algorithm and then not added to S because of the coin flips. Define $U = \{a_1,\dots,a_k\} \setminus C$. Additionally, define C' to be the set of all items that are considered, but not chosen during the run of our algorithm which have positive expected marginal contribution to S. I.e., C captures the items from the *good-looking* set after choosing S that we missed due to coin tosses, and C' *all* items we missed for the same reason which should have had a positive contribution in hindsight. Note that $C \subseteq C'$.

We can then split the left hand side term of Eq. (7) into two parts: the sum over C (upper bounded by the sum over C'), and the sum over U. Now we control separately these terms using linear combinations of $v(S)$ and $v(i^*)$.

Lemma 5. $\mathbb{E}[v(S)] \geq p \cdot \mathbb{E}\left[\sum_{a\in C} v\left(a|(S,\omega_{|S})\right)\right]$

Proof. Since $C \subseteq C'$ and C' contains all considered elements with nonnegative expected contribution to S, it is sufficient to show $\mathbb{E}[v(S)] \geq p \cdot \mathbb{E}\left[\sum_{a\in C'} v\left(a|(S,\omega_{|S})\right)\right]$.

We proceed as in Lemma 3. Let's consider for each $a \in A$ all the events $\mathcal{E}_a$ capturing the story of a run of the algorithm up to the point element a is considered (all the history if it is never considered).

Let G_a be the marginal contribution of element a to the solution set S. If $\mathcal{E}_a$ corresponds to a story in which element a is not considered, then it does not contribute - neither in the left, nor in the right hand side of the inequality we are trying to prove. Else, let (S_a, ω_a) be the partial solution when it is indeed considered:

$$\mathbb{E}[G_a|\mathcal{E}_a] = p \cdot v(a|(S_a,\omega_a)) \geq p \cdot \mathbb{E}[v(a|(S,\omega_{|S}))|\mathcal{E}_a].$$

The statement follows from the law of total probability with respect to $\mathcal{E}_a$, and state-wise submodularity of v. □

Lemma 6. $\mathbb{E}[v(S)] + v(i^*) \geq \mathbb{E}\left[\sum_{a\in U} v\left(a|(S,\omega_{|S})\right)\right]$.

Proof. Now let us turn towards the items U that were not considered by the algorithm. The intuition behind the claim is that if they were not considered then they were not good enough, in expectation, to compare with S. The proof, though, has to deal with some probabilistic subtleties.

Let's start fixing a story of the algorithm, i.e., the coin tosses and (S,ω_S), $S = s_1, s_2, \dots, s_T$, numbered according to their insertion in S, i.e., s_i is the i^{th} element to be added to S. For the sake of simplicity let's also renumber the elements in U as $a_1, \dots, a_l$ respecting the order given by the marginals over costs.

There are two cases. If during the whole algorithm the elements in U have ratio $\frac{v\left(a|(S_i,\omega_{|S_i})\right)}{c_a}$ smaller than that of the item which was instead considered, then one can easily argue, by adaptive submodularity, that:

$$\sum_{a\in U} v\left(a|(S,\omega_{|S})\right) \le \sum_{t=1}^{T} v(s_t|(S_t,\omega_t)) + v(u_1|(S,\omega_{|S}) \le$$
$$\le \sum_{t=1}^{T} v(s_t|(S_t,\omega_t)) + v(u_1) \le \sum_{t=1}^{T} v(s_t|(S_t,\omega_t)) + v(i^*).$$

being $S_t = (s_1, \dots, s_{t-1})$ and ω_t the restriction of $\omega_{|S}$ to S_t. Note that the last element u_1 is added to account for the unspent budget by the solution. We claim that the above inequality holds also in the case in which there is an element in U whose marginal over cost is greater than that of some in S. Such an element can exist because of the budget constraint: during the algorithm it had better marginal over cost, but was discarded because there was not enough room for it. We observe there can exist at most one such element, due to the budget constraint and because its value is upper bounded by u_1, so the above formula still holds.

Once we know that, by law of total probability, we have

$$\mathbb{E}\left[\sum_{a\in U} v\left(a|(S,\omega_{|S})\right)\right] \le \mathbb{E}\left[v(S)\right] + v(i^*), \tag{8}$$

concluding the proof. □

Combining the two Lemmata we get:

$$(1+\tfrac{1}{p})\,\mathbb{E}\left[v(S)\right] + \mathbb{E}\left[v(i^*)\right] \ge \mathbb{E}\left[\sum_{a\in U} v\left(a|(S,\omega_{|S})\right)\right] + \mathbb{E}\left[\sum_{a\in C} v\left(a|(S,\omega_{|S})\right)\right] =$$
$$= \mathbb{E}\left[\sum_{a\in U\cup C} v\left(a|(S,\omega_{|S})\right)\right].$$

Equation Eq. (7) implies

$$(1+\tfrac{1}{p})\,\mathbb{E}\left[v(S)\right] + \mathbb{E}\left[v(i^*)\right] \ge \mathbb{E}\left[\sum_{a\in U\cup C} v\left(a|(S,\omega_{|S})\right)\right] \ge \mathbb{E}\left[v\left(S^*|(S,\omega_{|S})\right)\right].$$

Also, by Eq. (5) and some algebra:

$$\begin{aligned}\mathbb{E}\left[v\left(S^*|(S,\omega_{|S})\right)\right] &= \mathbb{E}\left[\mathbb{E}\left[v(S^*\cup S,\omega) - v(S,\omega)|\omega_S\right]\right] \\ &= \mathbb{E}\left[v(S^*\cup S)\right] - \mathbb{E}\left[v(S)\right] \\ &\ge (1-p)\cdot\mathbb{E}[v(S^*)] - \mathbb{E}\left[v(S)\right]\end{aligned}$$

All together, denoting as OPT the $\mathbb{E}[v(S^*)]$, we get:

$$(2p+1)\mathbb{E}[v(S)] + p\mathbb{E}[v(i^*)] \ge p(1-p)OPT \tag{9}$$

Let's call ALG the expected value of the solution output by the algorithm. Since the algorithm chooses with a coin flip either the best expected single item or S, it holds

$$ALG = (1 - p_0)v(S) + p_0 v(i^*)$$

Picking $p_0 = \frac{p}{3p+1}$,

$$ALG = \frac{2p+1}{3p+1}\mathbb{E}[v(S)] + \frac{p}{3p+1}\mathbb{E}[v(i^*)] \geq \frac{p(1-p)}{3p+1} OPT.$$

The right hand side is minimized for $p = \frac{1}{3}$, concluding the proof of the first part of the statement.

The lazy version of AdaptiveGreedy is analogous to the non-adaptive setting, both for the algorithm and the analysis, so we omit repeating the proof.

In order to prove the last part of the statement, we start from Eq. (9) and apply the large instance property:

$$(1-p)\mathbb{E}\left[v(S^*)\right] \leq \left(2 + \tfrac{1}{p}\right)\mathbb{E}\left[v(S)\right] + \mathbb{E}\left[v(i^*)\right] \leq \left(2 + \tfrac{1}{p}\right)\mathbb{E}\left[v(S)\right] + \delta \cdot \mathbb{E}\left[v(S^*)\right]$$

Rearranging terms and assuming $p + \delta < 1$ we have that:

$$\mathbb{E}\left[v(S^*)\right] \leq \frac{\left(2 + \frac{1}{p}\right)}{1 - p - \delta} \cdot \mathbb{E}\left[v(S)\right].$$

Optimizing for $p \in (0, 1-\delta)$, we get the claimed result. Specifically, for $p = (\sqrt{3 - 2\delta} - 1)/2$ the approximation factor is $(4 + 2\sqrt{3} + \varepsilon_\delta)$, with

$$\varepsilon_\delta = 2\left(\frac{\sqrt{3-2\delta}+1}{(1-\delta)^2} + \frac{1}{1-\delta} - \sqrt{3} - 2\right) \approx \frac{6\delta(2-\delta)}{(1-\delta)^2}.$$

□

5 Experiments

Out of the numerous applications of submodular maximization subject to a knapsack constraint, we evaluate SampleGreedy and AdaptiveGreedy on two selected examples, using real and synthetic graph topologies. Variants of these have been studied in a similar context; see [39].

As our algorithms are randomized, but extremely fast, we use the best output out of $n = 5$ iterations. A delicate point is tuning the probabilities of acceptance p (line 9 of AdaptiveGreedy) for improved performance. While the choices of p in Theorems 2 and 5 minimize our analysis of the theoretical worst-case approximation, there are two factors that suggest a value much closer to 1 works best in practice: the small value of any singleton solution, and the much better guarantee of Lemma 4 for most widely used non-monotone submodular objectives. We do not micro-optimize for p but rather choose uniformly at randomly from $[0.9, 1]$.

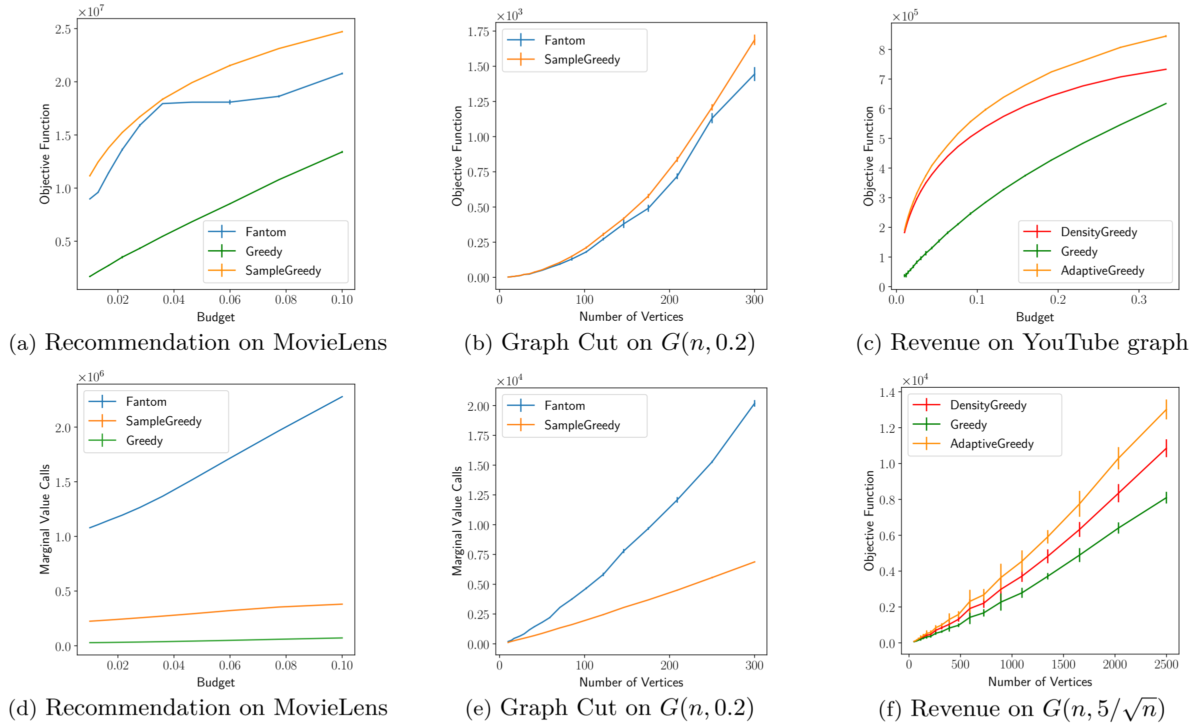
Video Recommendation: Suppose we have a large collection A of videos from various categories (represented as possibly intersecting subsets $C_1, \ldots, C_k \subseteq A$) and we want to design a recommendation system. When a user inputs a subset of categories and a target total length B, the system should return a set of videos from the selected categories of total duration at most B that maximizes an appropriate objective function. (Of course, instead of time here, we could use costs and a budget constraint.) Each video has a rating and there is some measure of similarity between any two videos. We use a weighted graph on A to model the latter: each edge $\{i, j\}$ between two videos i and j has a weight $w_{ij} \in [0, 1]$ capturing the percentage of their similarity. To pave the way for our $v(\cdot)$, we start from the auxiliary objective $f(S) = \sum_{i\in S}\sum_{j\in A} w_{ij} - \lambda \sum_{i\in S}\sum_{j\in S} w_{ij}$, for some $\lambda \geq 1$ [36, 39]. This is a *maximal marginal relevance* inspired objective [7] that rewards coverage, while penalizing similarity. For $\lambda = 1$, internal similarities are irrelevant and f becomes a cut function. However, one can penalize similarities even more severely as f is submodular for $\lambda \geq 1$ (e.g., Lin and Bilmes [36] use $\lambda = 5$).

In order to mimic the effect of a partition matroid constraint, i.e., the avoidance of many videos from the same category, we may use two parameters $\lambda \geq 1, \mu \geq 0$. While λ is as above, μ puts extra weight on similarities between videos that belong to the same category. That leads to a more general auxiliary objective $g(S) = \sum_{i\in S}\sum_{j\in A} w_{ij} - \sum_{i\in S}\sum_{j\in S}(\lambda + \chi_{ij}\mu)w_{ij}$, where χ_{ij} is equal to 1 if there exists ℓ such that $i, j \in C_\ell$ and 0 otherwise. To interpolate between choosing highly rated videos and videos that represent well the whole collection, here we use the submodular function $v(S) = \alpha \sum_{i\in S} \rho_i + \beta g(S)$ for $\alpha, \beta \geq 0$, where ρ_i is the rating of video i. We use $\lambda = 3$, $\mu = 7$ and set the parameters α, β so that the two terms are of comparable size.

We evaluate SampleGreedy on an instance based on the latest version of the MovieLens dataset [29], which includes 62000 movies, 13816 of which have *both* user-generated tags *and* ratings. We calculate the weights w_{ij} using these tags (with the L2 norm of the pairwise minimum tag vector, see Appendix A) while the costs are drawn independently from $U(0, 1)$. We compare against the FANTOM algorithm of Mirzasoleiman et al. [39] as it is the only other algorithm with a provable approximation guarantee that runs in reasonable time. Continuous greedy approaches [23] or the repeated greedy of Gupta et al. [27] are prohibitively slow. SampleGreedy consistently performs better than FANTOM for a wide range of budgets (Fig. 1a). Plotting the number of function evaluations against the budget, SampleGreedy is much faster (Fig. 1d) despite the fact that it is run 5 times!

Remark 1. The running time of FANTOM for fixed ε is $O(nr \log n)$, where r is the cardinality of the largest feasible solution. For a knapsack constraint this translates to $O(n^2 \log n)$. To be as fair as possible, we implemented FANTOM using lazy evaluations, which improves the number of evaluations of the objective function to $O(n \log^2 n)$ and is indeed much faster in practice, for the knapsack sizes we consider. Even so, our SampleGreedy is faster by a factor of $\Omega(\log n)$ which, including the improvement in the constants involved, still makes a huge difference. Note that in both Figs. 1d and 1e one can discern the superlinear increase of the function evaluations for FANTOM but not for SampleGreedy.

Influence-and-Exploit Marketing: Consider a seller of a single digital good (i.e., producing extra units of the good comes at no extra cost) and a social network on a set A of potential buyers. Suppose that the buyers influence each other and this is quantified by a weight w_{ij} on each edge $\{i, j\}$ between buyers i and j. Each buyer's value for the good depends on who owns it within her immediate social circle and how they influence her. A possible revenue-maximizing strategy for the seller is to first give the item for free to a selected set S of influential buyers (influence phase) and then extract revenue by selling to each of the remaining buyers at a price matching their value for the item due to the influential nodes (exploit phase). Here we further assume, similarly to the adaptation of this model by Mirzasoleiman et al. [39], that each buyer comes with a cost of convincing her to advertise the product to her friends. The seller has a budget B and the set S should be such that $\sum_{i\in S} c_i \leq B$.



(a) Recommendation on MovieLens (b) Graph Cut on $G(n, 0.2)$ (c) Revenue on YouTube graph

(d) Recommendation on MovieLens (e) Graph Cut on $G(n, 0.2)$ (f) Revenue on $G(n, 5/\sqrt{n})$

Figure 1: The four plots on the left compare the performance and the number of function evaluations of SampleGreedy and FANTOM on the video recommendation problem for the MovieLens dataset (a), (d) and on the maximum weighted cut problem on random graphs (b), (e). Since no $\varepsilon \leq 1$ affected the performance of FANTOM noticeably before becoming too computationally expensive, we used $\varepsilon = 1$ to achieve the maximum possible speedup. The plots on the far right illustrate the performance of AdaptiveGreedy (ignoring single item solutions, i.e., $p_0 = 0$) on the influence-and-exploit problem for two distinct topologies: the YouTube graph (c) and random graphs (f). All budgets are shown as fractions of the total cost.

We adopt the generalization of the *Concave Graph Model* of Hartline et al. [30] to non-monotone functions [2]. Each buyer $i \in A$ is associated with a non-negative concave function f_i. For any $i \in A$ and any set $S \subset A \setminus \{i\}$ of agents already owning the good, the value of i for it is $v_i(S) = f_i(\sum_{j \in S \cup \{i\}} w_{ij})$. The total potential revenue $v(S) = \sum_{i \in A \setminus S} v_i(S)$ that we aim to maximize is a non-monotone submodular function. Besides the theoretical guarantees for influence-and-exploit marketing in the Bayesian setting [30], there are strong experimental evidence of its performance in practice [2]. The problem generalizes naturally to different stochastic versions. We assume that the valuation function of each buyer i is of the form $f_i(x) = a_i\sqrt{x}$ where a_i is drawn independently from a Pareto Type II distribution with $\lambda = 1$, $\alpha = 2$. We only learn the exact value of a buyer when we give the good for free to someone in her neighborhood.

We evaluate AdaptiveGreedy on an instance based on the YouTube graph [47], containing 1,134,890 vertices. The (known) weights are drawn independently from $U(0, 1)$, and the costs are proportional to the sum of the weights of the incident edges. As AdaptiveGreedy is the first adaptive algorithm for the problem, we compare with non-adaptive alternatives like *Greedy*[2] and *Density Greedy*[3] for different values of the budget. AdaptiveGreedy outperforms the alternatives by up to 20% (Fig. 1c).

We observe similar improvements for Erdős-Rényi random graphs of different sizes and edge probability $5/\sqrt{n}$ and a fixed budget of 10% of the total cost (Fig. 1f).

[2]The simple greedy algorithm that in each step picks the element with the largest marginal value.

[3]The greedy part of Wolsey's algorithm [46].

Maximum Weighted Cut: Beyond the above applications, we would like to compare SampleGreedy to FANTOM with respect to both their performance and the number of value oracle calls as n grows. We turn to *weighted cut functions*—one of the most prominent subclasses of non-monotone submodular functions—on dense Erdős–Rényi random graphs with edge probability 0.2. The weights and the costs are drawn independently and uniformly from $[0, 1]$ and the budget is fixed to 15% of the total cost. Again SampleGreedy consistently performs better than FANTOM, albeit by 5–15% (Fig. 1b). In terms of running time, there is a large difference in favor of SampleGreedy (even for multiple runs), while the superlinear increase for FANTOM is evident (Fig. 1e).

6 Conclusions

The proposed random greedy method yields a considerable improvement over state-of-the-art algorithms, especially, but not exclusively, regarding the handling of huge instances. With all the subtleties of our work affecting solely our analysis, the algorithm remains strikingly simple and we are confident this will also contribute to its use in practice. Simultaneously, this very simplicity translates into a generality that can be employed to achieve comparably good results for a variety of settings; we demonstrated this in the case of the adaptive submodularity setting.

Specifically, we expect that our approach can be directly utilised to improve the performance and running time of algorithms that now use some variant of the algorithm of Gupta et al. [27]. Such examples include the distributed algorithm of da Ponte Barbosa et al. [12] and the streaming algorithm of Mirzasoleiman et al. [40] in the case of a knapsack constraint. We further suspect that the same algorithmic principle can be applied in the presence of incentives. This would largely improve the current state of the art in budget-feasible mechanism design for non-monotone objectives [11, 1].

Finally, a major question here is whether the same high level approach is valid even in the presence of additional combinatorial constraints. In particular, is it possible to achieve similar guarantees as FANTOM for a p-system and multiple knapsack constraints using only $O(n \log n)$ value queries?

References

[1] G. Amanatidis, P. Kleer, and G. Schäfer. Budget-feasible mechanism design for non-monotone submodular objectives: Offline and online. In *Proceedings of the 2019 ACM Conference on Economics and Computation, EC 2019, Phoenix, AZ, USA, June 24-28, 2019*, pages 901–919. ACM, 2019.

[2] M. Babaei, B. Mirzasoleiman, M. Jalili, and M. A. Safari. Revenue maximization in social networks through discounting. *Social Netw. Analys. Mining*, 3(4):1249–1262, 2013.

[3] A. Badanidiyuru, B. Mirzasoleiman, A. Karbasi, and A. Krause. Streaming submodular maximization: massive data summarization on the fly. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, pages 671–680. ACM, 2014.

[4] E. Balkanski and Y. Singer. The adaptive complexity of maximizing a submodular function. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 1138–1151. ACM, 2018.

[5] E. Balkanski, A. Rubinstein, and Y. Singer. An exponential speedup in parallel running time for submodular maximization without loss in approximation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 283–302. SIAM, 2019.

[6] N. Buchbinder, M. Feldman, J. Naor, and R. Schwartz. Submodular maximization with cardinality constraints. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1433–1452. SIAM, 2014.

[7] J. Carbinell and J. Goldstein. The use of MMR, diversity-based reranking for reordering documents and producing summaries. *SIGIR Forum*, 51(2):209–210, 2017.

[8] C. Chekuri, J. Vondrák, and R. Zenklusen. Submodular function maximization via the multilinear relaxation and contention resolution schemes. *SIAM J. Comput.*, 43(6):1831–1879, 2014.

[9] C. Chekuri, S. Gupta, and K. Quanrud. Streaming algorithms for submodular function maximization. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, volume 9134 of *Lecture Notes in Computer Science*, pages 318–330. Springer, 2015.

[10] C. Chekuri, T. S. Jayram, and J. Vondrák. On multiplicative weight updates for concave and submodular function maximization. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, ITCS 2015, Rehovot, Israel, January 11-13, 2015*, pages 201–210. ACM, 2015.

[11] N. Chen, N. Gravin, and P. Lu. On the approximability of budget feasible mechanisms. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 685–699, 2011.

[12] R. da Ponte Barbosa, A. Ene, H. L. Nguyen, and J. Ward. The power of randomization: Distributed submodular maximization on massive datasets. In *ICML*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 1236–1244. JMLR.org, 2015.

[13] A. Das and D. Kempe. Algorithms for subset selection in linear regression. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 45–54. ACM, 2008.

[14] A. Das and D. Kempe. Approximate submodularity and its applications: Subset selection, sparse approximation and dictionary selection. *Journal of Machine Learning Research*, 19(3): 1–34, 2018.

[15] J. Edmonds. Matroids and the greedy algorithm. *Math. Program.*, 1(1):127–136, 1971.

[16] A. Ene and H. L. Nguyen. A nearly-linear time algorithm for submodular maximization with a knapsack constraint. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 53:1–53:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[17] S. Fadaei, M. Fazli, and M. Safari. Maximizing non-monotone submodular set functions subject to different constraints: Combined algorithms. *Oper. Res. Lett.*, 39(6):447–451, 2011.

[18] M. Fahrbach, V. S. Mirrokni, and M. Zadimoghaddam. Submodular maximization with nearly optimal approximation, adaptivity and query complexity. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 255–273. SIAM, 2019.

[19] M. Fahrbach, V. S. Mirrokni, and M. Zadimoghaddam. Non-monotone submodular maximization with nearly optimal adaptivity and query complexity. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 1833–1842. PMLR, 2019.

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

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

[22] M. Feldman and R. Zenklusen. The submodular secretary problem goes linear. *SIAM J. Comput.*, 47(2):330–366, 2018.

[23] M. Feldman, J. Naor, and R. Schwartz. A unified continuous greedy algorithm for submodular maximization. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 570–579. IEEE Computer Society, 2011.

[24] M. Feldman, C. Harshaw, and A. Karbasi. Greed is good: Near-optimal submodular maximization via greedy optimization. In *Proceedings of the 30th Conference on Learning Theory, COLT 2017, Amsterdam, The Netherlands, 7-10 July 2017*, volume 65 of *Proceedings of Machine Learning Research*, pages 758–784. PMLR, 2017.

[25] D. Golovin and A. Krause. Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *J. Artif. Intell. Res.*, 42:427–486, 2011.

[26] A. Gotovos, A. Karbasi, and A. Krause. Non-monotone adaptive submodular maximization. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 1996–2003. AAAI Press, 2015.

[27] A. Gupta, A. Roth, G. Schoenebeck, and K. Talwar. Constrained non-monotone submodular maximization: Offline and secretary algorithms. In *Internet and Network Economics - 6th International Workshop, WINE 2010, Stanford, CA, USA, December 13-17, 2010. Proceedings*, volume 6484 of *LNCS*, pages 246–257. Springer, 2010.

[28] A. Gupta, V. Nagarajan, and S. Singla. Adaptivity gaps for stochastic probing: Submodular and XOS functions. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1688–1702. SIAM, 2017.

[29] F. M. Harper and J. A. Konstan. The MovieLens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4), Dec. 2015. ISSN 2160-6455.

[30] J. D. Hartline, V. S. Mirrokni, and M. Sundararajan. Optimal marketing strategies over social networks. In *Proceedings of the 17th International Conference on World Wide Web, WWW 2008, Beijing, China, April 21-25, 2008*, pages 189–198. ACM, 2008.

[31] D. Kempe, J. M. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. *Theory of Computing*, 11:105–147, 2015.

[32] R. Khanna, E. R. Elenberg, A. G. Dimakis, S. N. Negahban, and J. Ghosh. Scalable greedy feature selection via weak submodularity. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA*, volume 54 of *Proceedings of Machine Learning Research*, pages 1560–1568. PMLR, 2017.

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

[34] A. Kulik, H. Shachnai, and T. Tamir. Approximations for monotone and nonmonotone submodular maximization with knapsack constraints. *Math. Oper. Res.*, 38(4):729–739, 2013.

[35] J. Lee, V. S. Mirrokni, V. Nagarajan, and M. Sviridenko. Maximizing nonmonotone submodular functions under matroid or knapsack constraints. *SIAM J. Discrete Math.*, 23(4): 2053–2078, 2010.

[36] H. Lin and J. A. Bilmes. Multi-document summarization via budgeted maximization of submodular functions. In *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 2-4, 2010, Los Angeles, California, USA*, pages 912–920. The Association for Computational Linguistics, 2010.

[37] M. Minoux. Accelerated greedy algorithms for maximizing submodular set functions. In *Optimization Techniques*, pages 234–243, Berlin, Heidelberg, 1978. Springer Berlin Heidelberg.

[38] B. Mirzasoleiman, A. Karbasi, R. Sarkar, and A. Krause. Distributed submodular maximization: Identifying representative elements in massive data. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013, NIPS 2013, December 5-8, 2013, Lake Tahoe, Nevada, United States*, pages 2049–2057, 2013.

[39] B. Mirzasoleiman, A. Badanidiyuru, and A. Karbasi. Fast constrained submodular maximization: Personalized data summarization. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR*, pages 1358–1367. JMLR.org, 2016.

[40] B. Mirzasoleiman, S. Jegelka, and A. Krause. Streaming non-monotone submodular maximization: Personalized video summarization on the fly. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 1379–1386. AAAI Press, 2018.

[41] M. Mitrovic, E. Kazemi, M. Feldman, A. Krause, and A. Karbasi. Adaptive sequence submodularity. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 5353–5364, 2019.

[42] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions - I. *Math. Program.*, 14(1):265–294, 1978.

[43] R. Sipos, A. Swaminathan, P. Shivaswamy, and T. Joachims. Temporal corpus summarization using submodular word coverage. In *21st ACM International Conference on Information and Knowledge Management, CIKM'12, Maui, HI, USA, October 29 - November 02, 2012*, pages 754–763. ACM, 2012.

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

[45] S. Tschiatschek, R. K. Iyer, H. Wei, and J. A. Bilmes. Learning mixtures of submodular functions for image collection summarization. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, NIPS 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 1413–1421, 2014.

[46] L. A. Wolsey. Maximising real-valued submodular functions: Primal and dual heuristics for location problems. *Math. Oper. Res.*, 7(3):410–425, 1982.

[47] J. Yang and J. Leskovec. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems*, 42(1):181–213, 2015.

A Additional Details on Section 5

All graphs contain error bars, indicating the standard deviation between different runs of the experiments. This is usually insignificant due to the concentrating effect of the large size of the instances, despite the randomly initialized weights and inherent randomness of the algorithms used. Nevertheless, all results are obtained by running each experiment a number of times. For all algorithms involved, we use *lazy evaluations* with $\varepsilon = 0.01$.

Video Recommendation: We expand on the exact definition of the similarity measure that is only tersely described in the main text. Each movie i is associated with a tag vector $t^i \in [0,1]^{1128}$, where each coordinate contains a relevance score for that individual tag. These tag vectors are *not* normalized and have no additional structure, other than each coordinate being restricted to $[0,1]$. We define the similarity w_{ij} between two movies i, j as:

$$w_{ij} = \sqrt{\sum_{k=1}^{1128} \left(\min\{t^i_k, t^j_k\}\right)^2}.$$

In other words, it is the L2 norm of the *coordinate-wise minimum* of t^i and t^j. This metric was chosen so that if *both* movies have a high value in some tag, this counts as a much stronger similarity than one having a high value and the other a low one. For example, if we consider an inner product metric, any movie with all tags set to 1 would be as similar as possible to all other movies, even though it would include many tags that would be missing from the others. In particular, any movie would be appear more similar to the all 1 movie than to itself! Choosing the minimum of both tags avoids this issue. Another possibility would be to normalize each tag vector before taking the inner product, to obtain the *cosine similarity.* Although this alleviates some of the issues, there is some information loss as one movie could meaningfully have higher scores in all tags than another one; tags are not mutually exclusive. Ultimately any sensible metric has advantages and disadvantages and the exact choice has little bearing on our results. The similarity scores are then divided by their maximum as a final normalization step.

The experiment was repeated 5 times. The budget is represented as a fraction of the total cost starting at 1/100 and geometrically increasing to 1/10 in 10 steps. The total computation time was around 3 hours.

Influence-and-Exploit Marketing: For the YouTube graph, the experiment was repeated 5 times for a budget starting at 1/100 of the total cost and geometrically increasing to 1/3 in 20 steps, leading to a total computation time of 7 hours. For the Erdős–Rényi graph with n vertices and edge probability $5/\sqrt{n}$ it was repeated 10 times, for n starting at 50 and geometrically increasing to 2500 in 20 steps, taking approximately 10 minutes.

Maximum Weighted Cut: The experiment was repeated 10 times for n starting at 10 and increasing geometrically to 300 in 20 steps, requiring approximately 5 minutes.