Submodular Maximization Subject to a Knapsack Constraint: Combinatorial Algorithms with Near-Optimal Adaptive Complexity

Georgios Amanatidis [1] **Federico Fusco** [2] **Philip Lazos** [2] **Stefano Leonardi** [2]
Alberto Marchetti-Spaccamela [2] **Rebecca Reiffenhäuser** [2]

Abstract

The growing need to deal with massive instances motivates the design of algorithms balancing the quality of the solution with applicability. For the latter, an important measure is the *adaptive complexity*, capturing the number of sequential rounds of parallel computation needed. In this work we obtain the first *constant factor* approximation algorithm for non-monotone submodular maximization subject to a knapsack constraint with *near-optimal* $O(\log n)$ adaptive complexity. Low adaptivity by itself, however, is not enough: one needs to account for the total number of function evaluations (or value queries) as well. Our algorithm asks $\tilde{O}(n^2)$ value queries, but can be modified to run with only $\tilde{O}(n)$ instead, while retaining a low adaptive complexity of $O(\log^2 n)$. Besides the above improvement in adaptivity, this is also the first *combinatorial* approach with sublinear adaptive complexity for the problem and yields algorithms comparable to the state-of-the-art even for the special cases of cardinality constraints or monotone objectives. Finally, we showcase our algorithms' applicability on real-world datasets.

1. Introduction

Submodular optimization is a very popular topic that is relevant to various research areas as it captures the natural notion of *diminishing returns*. Its numerous applications include viral marketing (Hartline et al., 2008; Kempe et al., 2015), data summarization (Tschiatschek et al., 2014; Mirzasoleiman et al., 2016; Badanidiyuru et al., 2020), feature selection (Das & Kempe, 2008; 2018; Mirzasoleiman et al., 2020), and clustering (Mirzasoleiman et al., 2013). Prominent examples from combinatorial optimization are cut functions in graphs and coverage functions.

Submodularity is often implicitly associated with monotonicity, and many results rely on that assumption. However, non-monotone submodular functions do naturally arise in applications, either directly or from combining monotone submodular objectives and modular *penalization* or *regularization* terms (Hartline et al., 2008; Tschiatschek et al., 2014; Mirzasoleiman et al., 2016; Breuer et al., 2020; Amanatidis et al., 2020). Additional constraints, like cardinality, matroid, knapsack, covering, and packing constraints, are prevalent in applications and have been extensively studied. In this list, *knapsack* constraints are among the most natural, as they capture limitations on budget, time, or size of the elements. Like matroid constraints, they generalize cardinality constraints, yet they are not captured by the former.

The main computational bottleneck in submodular optimization comes from the necessity to repeatedly evaluate the objective function for various candidate sets. These so-called *value queries* are often notoriously heavy to compute, e.g., for exemplar-based clustering (Dueck & Frey, 2007), log-determinant of submatrices (Kazemi et al., 2018), and accuracy of ML models (Das & Kempe, 2008; Khanna et al., 2017). With real-world instances of these problems growing to enormous sizes, simply reducing the number of queries is not always sufficient and parallelisation has become an increasingly central paradigm. However, classic results in the area, often based on the greedy method, are inherently sequential: the intuitive approach of building a solution element-by-element contradicts the requirement of running *independent* computations on many machines in parallel. The degree to which an algorithm can be parallelized is measured by the notion of *adaptive complexity*, or adaptivity, introduced in Balkanski et al. (2018). It is defined as the number of sequential rounds of parallel computation needed to terminate. In each of these rounds, polynomially many value queries may be asked, but they can only depend on the answers to queries issued in past rounds.

Contribution. We design the first combinatorial randomized algorithms for maximizing a (possibly) non-monotone submodular function subject to a knapsack constraint that combine constant approximation, low adaptive complexity,

[1]University of Essex, United Kingdom. [2]Sapienza University of Rome, Italy. Correspondence to: Federico Fusco <fuscof@diag.uniroma1.it>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

and a small number of queries. In particular, we obtain

- a 9.465-approximation algorithm, ParKnapsack, that has $O(\log n)$ adaptivity and uses $O(n^2 \log^2 n)$ value queries. This is the *first* constant factor approximation to the problem with optimal adaptive complexity up to a $O(\log \log n)$ factor (Theorem 1).
- a variant of our algorithm with the same approximation, near-optimal $O(n \log^3 n)$ query complexity, and $O(\log^2 n)$ adaptivity (Theorem 2). This is the first constant factor approximation algorithm that uses only $\tilde{O}(n)$ queries and has *sublinear* adaptivity.
- 3-approximation algorithms for *monotone* objectives that combine $O(\log n)$ adaptivity with $O(n^2 \log^2 n)$ total queries, and $O(\log^2 n)$ adaptivity with $O(n \log^3 n)$ queries, respectively (Theorem 3). Even in the monotone setting, the latter is the first $O(1)$-approximation algorithm combining $\tilde{O}(n)$ queries and sublinear adaptivity.
- 5.83-approximation algorithms for *cardinality* constraints that match or surpass the state-of-the-art when it comes to the combination of approximation, adaptivity and total queries (Theorem 4).

See Table 1 for an overview of our results.

Technical Challenges. Like existing work for cardinality or matroid constraints, (e.g., Balkanski et al., 2019b; Breuer et al., 2020), in order to reduce the adaptive complexity we iteratively sample sequences of feasible elements and add large chunks of them to our solution. However, knapsack constraints do not allow for the elegant counting arguments used in the case of cardinality or matroid constraints. The reason is that while the latter can be interpreted as a 1-independence system, a knapsack constraint induces a $\Theta(n)$-independence system, leading to poor results when naively adjusting existing approaches. A natural and very successful way of circumventing the resulting difficulties is to turn towards a *continuous* version of the problem. This, however requires evaluating the objective function also for *fractional* sets, i.e., such algorithms require access to an oracle for the multilinear relaxation and its gradient. Typically, these values are estimated by sampling, requiring $\tilde{\Theta}(n^2)$ samples, see e.g. Chekuri & Quanrud (2019a). Our choice to avoid the resulting increase in query complexity and deal directly with the discreteness of the problem calls for specifically tailored algorithmic approaches. Most crucially, our main subroutine ThreshSeq needs to balance a suitable definition of *good quality* candidates with a way to also reduce the size (not simply by cardinality, but a combination of overall cost and absolute marginal values) of the candidate set by a constant factor in each adaptive round.

Both these goals are further hindered by our second main challenge, non-monotonicity. In presence of elements with negative marginals, not only is it harder to maintain a good quality of our solution, but size measures like the overall absolute marginals of our candidate sets are no longer inclusion-monotone. In fact, even one such element can arbitrarily deteriorate intuitive quality measures like the overall marginal density of the candidate set, causing a new adaptive round. Our approach combines carefully designed stopping times in ThreshSeq with a separate handling of the elements responsible for most of the above mentioned *discreteness issues*, i.e., elements with cost less than $1/n$ of the budget and elements of maximum value.

Related Work. Submodular maximization has been studied extensively since the seminal work of Nemhauser et al. (1978). For *monotone* submodular objectives subject to a knapsack constraint the $\frac{e}{e-1}$-approximation algorithm of Sviridenko (2004) is best-possible, unless P = NP (Feige, 1998). For the *non-monotone* case, a number of continuous greedy approaches (Feldman et al., 2011; Kulik et al., 2013; Chekuri et al., 2014) led to the current best factor of e when a knapsack, or any downward closed, constraint is involved. Combinatorial approaches (Gupta et al., 2010; Amanatidis et al., 2020) achieve somewhat worse approximation, but are often significantly faster and thus relevant in practice.

While the notion of adaptivity has been explored in the context of parallel algorithms and communication complexity for a long time, (e.g., Valiant, 1975; Duris et al., 1987), the adaptive complexity for submodular maximization was first studied by Balkanski & Singer (2018). In that work, for monotone objectives and a cardinality constraint, the authors achieved an $O(1)$ approximation with $O(\log n)$ adaptivity, along with an almost matching lower bound: to get an $o(\log n)$ approximation, adaptivity must be $\Omega\left(\frac{\log n}{\log \log n}\right)$. This result has been then improved (Balkanski et al., 2019a; Ene & Nguyen, 2019; Fahrbach et al., 2019a) and recently Breuer et al. (2020) achieved an optimal $\frac{e}{e-1}$-approximation in $O(\log n \log^2 \log k)$ adaptive rounds and $O(n \log \log k)$ query complexity, where k is the cardinality constraint.

The study of adaptivity for non-monotone objectives was initiated by Balkanski et al. (2018) again for a cardinality constraint, showing a constant approximation in $O(\log^2 n)$ adaptive rounds, later improved by (Fahrbach et al., 2019b; Ene & Nguyen, 2020; Kuhnle, 2021). Non-monotone maximization is also interesting in the unconstrained scenario. Recently, Ene et al. (2018) and Chen et al. (2019) achieved a $2 + \varepsilon$ approximation with constant adaptivity depending only on ε. Note that the algorithm of Chen et al. (2019) needs only $\tilde{O}(n)$ value queries, where the $\tilde{O}$ hides terms poly-logarithmic in ε^{-1} and n.

Richer constraints, e.g., matroids and multiple packing constraints, have also been studied (Balkanski et al., 2019b; Ene et al., 2019; Chekuri & Quanrud, 2019a;b). For knapsack constraints (as a special case of packing constraints) Ene et al. (2019) and Chekuri & Quanrud (2019a) provide low adaptivity results—$O(\log^2 n)$ for non-monotone and

Reference	Objective	Constraint	Approx.	Adaptive Complexity	Queries
Ene et al. (2019)	General	Knapsack	$\boldsymbol{e+\varepsilon}$	$O(\log^2 n)$	$\tilde{O}(n^2)$
Theorems 1 and 2 (this work)	General	Knapsack	$9.465+\varepsilon$	$\boldsymbol{O(\log n)} \parallel O(\log^2 n)$	$\tilde{O}(n^2) \parallel \boldsymbol{\tilde{O}(n)}$
Ene et al. (2019)	Monotone	Knapsack	$\boldsymbol{\frac{e}{e-1}+\varepsilon}$	$\boldsymbol{O(\log n)}$	$\tilde{O}(n^2)$
Chekuri & Quanrud (2019a)	Monotone	Knapsack	$\boldsymbol{\frac{e}{e-1}+\varepsilon}$	$\boldsymbol{O(\log n)}$	$\tilde{O}(n^2)$
Theorem 3 (this work)	Monotone	Knapsack	$3+\varepsilon$	$\boldsymbol{O(\log n)} \parallel O(\log^2 n)$	$\tilde{O}(n^2) \parallel \boldsymbol{\tilde{O}(n)}$
Ene & Nguyen (2020)	General	k-Cardinality	$\boldsymbol{e+\varepsilon}$	$\boldsymbol{O(\log n)}$	$\tilde{O}(nk^2)$
Kuhnle (2021)	General	k-Cardinality	$6+\varepsilon$	$\boldsymbol{O(\log n)}$	$\boldsymbol{\tilde{O}(n)}$
Kuhnle (2021)	General	k-Cardinality	$5.18+\varepsilon$	$O(\log^2 n)$	$\boldsymbol{\tilde{O}(n)}$
Theorem 4 (this work)	General	k-Cardinality	$5.83+\varepsilon$	$\boldsymbol{O(\log n)} \parallel O(\log n \log k)$	$\tilde{O}(nk) \parallel \boldsymbol{\tilde{O}(n)}$

Table 1: Our results—main result highlighted—compared to the state-of-the-art for low-adaptivity. Some lines present two algorithms: in each entry where the two differ, the first term refers to one algorithm, the second to the other, consistently. Bold indicates the best result(s) in each setting. In the last two columns the dependence on ε is omitted; in the last column only the leading terms are stated.

$O(\log n)$ for monotone—via continuous approaches (see Table 1; notice that the query complexity of these algorithms is stated with respect to queries to f and not to its multilinear extension). Chekuri & Quanrud (2019a) also provide two combinatorial algorithms for the monotone case: one with optimal approximation and adaptivity but $O(n^4)$ value queries, and one with linear query complexity, optimal adaptivity but an approximation factor parameterized by $max_{x \in \mathcal{N}} c(x)$ which can be arbitrarily bad.

2. Preliminaries

Let $f : 2^{\mathcal{N}} \to \mathbb{R}$ be a set function over a ground set $\mathcal{N}$ of n elements. For $S, T \subseteq \mathcal{N}$, $f(S \mid T)$ denotes the *marginal value* of S with respect to T, i.e., $f(S \cup T) - f(T)$. To ease notation, we write $f(x \mid T)$ instead of $f(\{x\} \mid T)$. The function f is *non-negative* if $f(S) \geq 0, \forall S \subseteq \mathcal{N}$, *monotone* if $f(S) \leq f(T), \forall S, T \subseteq \mathcal{N}$, and *submodular* if $f(x \mid T) \leq f(x \mid S), \forall S, T \subseteq \mathcal{N}$ with $S \subseteq T$ and $x \notin T$.

We study non-negative, possibly *non-monotone*, submodular maximization under a *knapsack constraint*. Formally, we are given a budget $B > 0$, a non-negative submodular function f and a non-negative additive cost function $c : 2^{\mathcal{N}} \to \mathbb{R}_{>0}$. The goal is to find $O^* \in \arg\max_{T \subseteq \mathcal{N} : c(T) \leq B} f(T)$. Let $OPT = f(O^*)$ denote the value of such an optimal set. Given a (randomized) algorithm for the problem, let ALG denote the expected value of its output. We say that the algorithm is a β-approximation algorithm if $ALG \cdot \beta \geq OPT$. Throughout this work we assume, without loss of generality, that $\max_{x \in \mathcal{N}} c(x) \leq B$.

We assume access to f through value queries, i.e., for each $S \subseteq \mathcal{N}$, an oracle returns $f(S)$ in constant time. Given such an oracle for f, the *adaptive complexity* or *adaptivity* of an algorithm is the minimum number of rounds in which the algorithm makes $O(\text{poly}(n))$ *independent* queries to the evaluation oracle. In each adaptive round the queries may *only* depend on the answers to queries from past rounds. With respect to the same oracle, the *query complexity* of an algorithm is the total number of value queries it makes.

We finally state some widely known properties of submodular functions that are extensively used in the rest of the paper. The first lemma summarizes two equivalent definitions of submodular functions shown by Nemhauser et al. (1978).

Lemma 1. *Let $f : 2^{\mathcal{N}} \to \mathbb{R}$ be a submodular function and S, T, U be any subsets of $\mathcal{N}$, with $S \subseteq T$. Then i) $f(U \mid T) \leq f(U \mid S)$, ii) $f(S \mid T) \leq \sum_{x \in S} f(x \mid T)$.*

The second lemma, Lemma 2.2 of Buchbinder et al. (2014), is an important tool for tackling non-monotonicity.

Lemma 2 (Sampling Lemma)**.** *Let $f : 2^{\mathcal{N}} \to \mathbb{R}$ be a submodular function, $X \subseteq \mathcal{N}$ and X_p be a random subset of X, where each element of X is contained with probability at most p. Then $\mathbb{E}[f(X_p)] \geq (1-p) f(\emptyset)$.*

Finally, we assume access to SUBMODMAX, an unconstrained submodular maximization oracle. For instance, this can be implemented via the combinatorial algorithm of Chen et al. (2019), which outputs a $(2+\varepsilon)$-approximation of $\max_{T \subseteq \mathcal{N}} f(T)$ for a given precision ε in $O(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$ adaptive rounds and linear query complexity. For our experiments, we use the much simpler 4-approximation of Feige et al. (2011), which has adaptive complexity of 1.

3. Non-Monotone Submodular Maximization

To achieve sublinear adaptivity we need to add large chunks of elements to the solution without using intermediate value queries. The sequence of elements that are candidates to be added to the current solution is randomly drawn in SAM-

PLESEQ. This subroutine receives as input a partial solution S, a set of feasible elements X and a budget, and outputs a sequence A each element of which is sequentially drawn uniformly at random among the remaining elements of X that do not cause $S \cup A$ to exceed the budget. We do, however, need to restrict ourselves to only adding a *suitable prefix* of A; with each element added, the original "good" quality of the leftover candidates in X can quickly deteriorate.

Algorithm 1 SAMPLESEQ(S, X, B)

Input: current solution S, set X of remaining elements and budget $B > 0$
$A \leftarrow [\ ],\ i \leftarrow 1$
while $X \neq \emptyset$ **do**
  Draw a_i uniformly at random from X
  $A \leftarrow [a_1, \dots, a_{i-1}, a_i]$
  $X \leftarrow \{x \in X \setminus \{a_i\} : c(x) + c(A) + c(S) \leq B\}$
  $i \leftarrow i + 1$
return $A = [a_1, a_2, \dots, a_d]$

The selection of the prefix of the sequence $A = [a_1, \dots, a_d]$ to be added to the current solution S is then done by THRESHSEQ. Given a threshold τ, we add to S a prefix $A_i = [a_1, \dots, a_i]$ such that for all $j < i$ the average contribution to $S \cup A_j$ of the elements in $X \setminus A_j$ is comparable to τ. Then the expected value of $f(A_i \mid S)$ should be comparable to $\tau\mathbb{E}\left[c(A_i)\right]$. In order to compute A_i in one single parallel round, one can *a posteriori* compute for each prefix A_j of A the *a priori* (with respect to the uniform samples) expected marginal value of a_{j+1}; with a_{j+1} drawn uniformly at random from the elements in $X \setminus A_j$ still fitting the budget, this means simply averaging over their marginal densities. Since all the value queries depend only on S and A, finding the prefix needs only a single adaptive round.

The crucial difficulty lies in the fact that limiting the expected marginal density is insufficient to bound the number of adaptive steps. In the worst case, a single very negative element could trigger this condition. We circumvent the resulting adaptive complexity of up to n by imposing two different stopping conditions. The *cost condition*, i.e. i^* in THRESHSEQ, is triggered once an ε-fraction of all remaining candidates' cost is due to elements that are no longer *good*, i.e., they now have marginal density below τ. The *value condition*, corresponding to j^* in THRESHSEQ, is triggered at most ℓ times, which happens whenever the elements with negative marginal value make up an ε-fraction of the entire leftover marginal value. Now, in each adaptive step, either the overall cost or the summed-up marginal contributions of the candidate set decrease by a factor of $(1 - \varepsilon)$. These observations are formalized below.

Lemma 3. *Let* $\kappa(X) = \max_{x,y \in X} c(x)/c(y)$. *Then* THRESHSEQ *runs in* $O\left(\frac{1}{\varepsilon}\log(n\kappa(X)) + \ell\right)$ *adaptive rounds and issues* $O\left(n^2\left(\frac{1}{\varepsilon}\log n\kappa(X) + \ell\right)\right)$ *value queries.*

Algorithm 2 THRESHSEQ$(X, \tau, \varepsilon, \ell, B)$

Input: set X of elements, threshold $\tau > 0$, precision $\varepsilon \in (0, 1)$, parameter ℓ and budget B
$S \leftarrow \emptyset$, ctr $\leftarrow 0$
$X \leftarrow \{x \in X : f(x) \geq \tau c(x)\}$
while $X \neq \emptyset$ and ctr $< \ell$ **do**
  $[a_1, a_2, \dots, a_d] \leftarrow$ SAMPLESEQ(S, X, B)
  for $i = 1, \dots, d$ **do**
    $A_i \leftarrow \{a_1, a_2, \dots, a_i\}$
    $X_i \leftarrow \{a \in X \setminus A_i : c(a) + c(S \cup A_i) \leq B\}$
    $G_i \leftarrow \{a \in X_i : f(a \mid S \cup A_i) \geq \tau \cdot c(a)\}$
    $E_i \leftarrow \{a \in X_i : f(a \mid S \cup A_i) < 0\}$
  $i^* \leftarrow \min\{i : c(G_i) \leq (1 - \varepsilon)c(X)\}$
  $j^* \leftarrow \min\{j : \sum_{x \in G_j} \varepsilon f(x|S \cup A_j) \leq \sum_{x \in E_j} |f(x|S \cup A_j)|\}$
  $k^* \leftarrow \min\{i^*, j^*\}$
  $S \leftarrow S \cup A_{k^*}$
  $X \leftarrow G_{k^*}$
  if $j^* < i^*$ **then**
    ctr $\leftarrow$ ctr $+ 1$
return S

Proof. The adaptive rounds correspond to iterations of the while loop. In fact, once a new sequence is drawn by SAMPLESEQ, all the value queries needed are deterministically induced by it and hence can be assigned to different independent machines. Gathering this information we can determine k^* and start another iteration of the while loop. Bounding the number of such iterations where the value condition is triggered is easy, since it is forced to be at most ℓ. For the cost condition we use the geometric decrease in the total cost of X: every time it is triggered, the total cost of the feasible elements X is decreased by at least a $(1 - \varepsilon)$ factor. At the beginning of the algorithm, that cost is at most Cn, with $C = \max_{x \in X} c(x)$, and it needs to decrease below $c = \min_{x \in X} c(x)$ to ensure that $X = \emptyset$. Call r the number of such rounds. In the worst case we need $Cn(1 - \varepsilon)^r < c$, meaning that the adaptivity is upper bounded by $\frac{1}{\varepsilon}\log\left(n\kappa(X)\right) + \ell$. Finally, notice that the query complexity is just a n^2 factor greater than the adaptivity: each adaptive round contains $O(n^2)$ value queries, since the length of the sequence output by SAMPLESEQ may be linear in n and for each prefix the value of the marginals of all the remaining elements has to be considered. □

Having settled the adaptive and query complexity of THRESHSEQ, we move to proving that our conditions ensure good expected marginal density.

Lemma 4. *For any* X, τ, $\varepsilon \in (0, 1)$, ℓ *and* b, *the random set* S *output by* THRESHSEQ *is such that* $c(S) \leq B$ *and* $\mathbb{E}\left[f(S)\right] \geq (1 - \varepsilon)^2 \tau \mathbb{E}\left[c(S)\right]$.

Proof. We first note that $c(S) \leq B$ with probability 1 since

SAMPLESEQ always returns feasible sequences. The algorithm adds a chunk of elements to the solution in each iteration of the *while* loop. This, along with the fact that each of these chunks is an ordered prefix of a sequence output by SAMPLESEQ, induces a total ordering on the elements in S. To facilitate the presentation of this proof, we imagine that the elements of S are added one after the other, according to this total order. Let us call the t-th such element s_t, and let $\mathcal{F}_t$ denote the filtration capturing the randomness of the algorithm up to, but excluding, the adding of s_t to its chunk's random sequence. We show that whenever any s_t is added, its expected marginal density is at least $(1-\varepsilon)^2\tau$.

Fix some s_t and consider the iteration of the while loop in which it is added to the solution. We denote with S_{old} the partial solution at the beginning of that while loop, with X the candidate set $\{x \in \mathcal{N} : f(x \,|\, S_{old}) \geq \tau \cdot c(x), c(x) + c(S_{old}) \leq B\}$ at that point, and with A the sequence drawn in that iteration by SAMPLESEQ. Let $A_{(t)}$ be the prefix of A up to, and excluding, s_t. Then $S_t = S_{old} \cup A_{(t)}$ is the set of all elements added to the solution before s_t. Note that, given $\mathcal{F}_t$, the sets X, S_{old} and $A_{(t)}$ are deterministic, while the rest of A is random. Recall that s_t is drawn uniformly at random from $X_{(t)} = \{x \in X \setminus A_{(t)} | c(S_t) + c(x) \leq B\}$. We need to show that $\mathbb{E}\left[f(s_t \,|\, S_t) \,|\, \mathcal{F}_t\right] \geq (1-\varepsilon)^2\tau\mathbb{E}\left[c(s_t) \,|\, \mathcal{F}_t\right]$, where the randomness is with respect to the uniform sampling in $X_{(t)}$.

If s_t is the first element in A, this holds since all the elements in X exhibit a marginal density greater that τ. If s_t is not the first element, it means that the value and cost condition were not triggered for the previous one. Call G and E the sets of the good and negative elements with respect to S_t, i.e., $G = \{x \in X_{(t)} : f(x \,|\, S_t) \geq \tau c(x)\}$ and $E = \{x \in X_{(t)} : f(x \,|\, S_t) < 0\}$, which are also deterministically defined by $\mathcal{F}_t$. Finally, let p_x be $\mathbb{P}\left(s_t = x \,|\, \mathcal{F}_t\right)$ which is equal to $|X_{(t)}|^{-1}$ for all $x \in X_{(t)}$ and zero otherwise, then

$$
\begin{aligned}
&\mathbb{E}[f(s_t \,|\, S_t) \,|\, \mathcal{F}_t] - (1-\varepsilon)^2\tau\mathbb{E}\left[c(s_t) \,|\, \mathcal{F}_t\right] = \\
&\quad = \sum_{x \in X} p_x f(x \,|\, S_t) - (1-\varepsilon)^2\tau \sum_{x \in X} p_x c(x) \\
&\quad \geq \sum_{x \in G \cup E} p_x f(x \,|\, S_t) - (1-\varepsilon)^2\tau \sum_{x \in X} p_x c(x) \\
&\quad = \varepsilon \sum_{x \in G} p_x f(x \,|\, S_t) - \sum_{x \in E} p_x |f(x \,|\, S_t)| \qquad (1) \\
&\qquad + (1-\varepsilon)\tau\Big[\sum_{x \in G} p_x c(x) - (1-\varepsilon)\sum_{x \in X} p_x c(x)\Big] \qquad (2) \\
&\qquad + (1-\varepsilon)\sum_{x \in G} p_x \Big[f(x \,|\, S_t) - \tau c(x)\Big] \geq 0 \,. \qquad (3)
\end{aligned}
$$

Expressions (1) and (2) are nonnegative since the value and cost conditions were not triggered before adding s_t. Expression (3) is nonnegative by definition of G. We have shown for all t and $\mathcal{F}_t$, the expected marginal density of the t-th element (if any) added by our algorithm is large enough. A careful recursive application of conditional expectation is then enough to get the desired bound. We refer the interested reader to the supplementary materials for the details. □

Having established that S has expected density comparable to our threshold τ, we move on to showing that when THRESHSEQ terminates, either a large portion of the budget is used up in expectation, or we can bound the value of good candidates that are left outside the solution.

Lemma 5. *When* THRESHSEQ *terminates we have* $f(S) \geq \varepsilon\ell\sum_{x \in G} f(x \,|\, S)$, *where* $G = \{x \in X \setminus S : f(x \,|\, S) \geq \tau c(x),\ c(x) + c(S) \leq B\}$.

Proof. THRESHSEQ terminates in one of two cases. Either X is empty, meaning that there are no elements still fitting in the budget whose marginal density is greater than τ—and in that case the inequality we want to prove trivially holds—or the value condition has been triggered ℓ times.

For the latter, suppose that the value condition was triggered for the ith time during iteration t_i of the while loop. Denote by S_{t_i} the solution at the end of that iteration. We are interested in the sets $X_{j^*}, G_{j^*}, E_{j^*}$ of that particular iteration of the while loop. In order to be consistent across iterations, we use $X_{(i)}$, $G_{(i)}$, and $E_{(i)}$ to denote these sets for iteration t_i. Since the value condition was triggered during t_i, we have $\varepsilon \sum_{x \in G_{(i)}} f(x \,|\, S_{t_i}) \leq \sum_{x \in E_{(i)}} |f(x \,|\, S_{t_i})|$. Clearly, $G_{(\ell)}$ is what we denoted by G in the statement and S_{t_ℓ} is S. Also notice that $E_{(j)} \cap E_{(k)} = \emptyset$ for $j \neq k$.

Now, by non-negativity of f and Lemma 1, we have

$$
0 \leq f\Big(S_{t_\ell} \cup \bigcup_{i=1}^{\ell} E_{(i)}\Big) \leq f(S_{t_\ell}) + \sum_{i=1}^{\ell} \sum_{x \in E_{(i)}} f(x \,|\, S_{t_i}) \,.
$$

Rearranging the terms and using the value condition, we get

$$
\begin{aligned}
f(S_{t_\ell}) &\geq \sum_{i=1}^{\ell} \sum_{x \in E_{(i)}} |f(x \,|\, S_{t_i})| \geq \sum_{i=1}^{\ell} \varepsilon \sum_{x \in G_{(i)}} f(x \,|\, S_{t_i}) \\
&\geq \ell\varepsilon \sum_{x \in G_{(\ell)}} f(x \,|\, S_{t_\ell}) \,.
\end{aligned}
$$

The last inequality follows from the submodularity of f and the fact that that $G_{(1)} \supseteq G_{(2)} \supseteq \ldots \supseteq G_{(\ell)}$. □

Lemma 5 still leaves a gap: how can we account for the elements which have marginal density greater than τ but are not considered due to the budget constraint? It can be the case that due to some poor random choices we initially filled the solution with low quality elements, preventing the algorithm at later stages to consider good elements with large costs. To handle this, we need the following simple lemma; we defer its proof to the supplementary material.

Lemma 6. *Suppose that* $\mathbb{E}[c(S)] < \frac{B}{2}(1-\varepsilon)$*. Then, running* THRESHSEQ $\frac{1}{\varepsilon}\log(\frac{1}{\varepsilon})$ *times, with probability at least* $(1-\varepsilon)$*, there is at least one run where* $c(S) < \frac{B}{2}$*.*

We can now present the full *parallel* algorithm PARKNAPSACK for the non-monotone case. It considers separately the set $\mathcal{N}_-$ of "small" elements, each with cost smaller than B/n, and the set of "large" elements $\mathcal{N}_+ = \mathcal{N} \setminus \mathcal{N}_-$. The set $\mathcal{N}_-$ is fed to the low adaptive complexity unconstrained maximization routine SUBMODMAX as discussed in Section 2. For the large elements, PARKNAPSACK samples each element of $\mathcal{N}_+$ with probability p to get a random subset H, and then it runs THRESHSEQ a logarithmic number of times on H, in parallel, for different guesses of the "right" threshold. The partition between $\mathcal{N}_+$ and $\mathcal{N}_-$ is critical in bounding the adaptivity of THRESHSEQ, as $\kappa(\mathcal{N}_+) \le n$.

Algorithm 3 PARKNAPSACK$(\mathcal{N}, f, \varepsilon, \alpha, p, B)$

```
Input: Ground set N, submodular function f, budget B, precision ε, parameter α and sampling probability p
N_− ← {x ∈ N : c(x) < B/n};  N_+ ← N \ N_−
x* ← max_{x∈N_+} f(x);   τ̂ ← αn f(x*)/B
ε̂ ← ε/125;  ℓ ← ε̂^{−2};  k ← ε̂^{−1} log(n)
S_− ← SUBMODMAX(N_−, ε̂)
H ← sample each element in N_+ independently at random with probability p
for i = 0, 1, ..., k in parallel do
    τ_i ← τ̂ · (1 − ε̂)^i
    for j = 1, 2, ..., ε̂^{−1} log(ε̂^{−1}) in parallel do
        S^i_j ← THRESHSEQ(H, τ_i, ε̂, ℓ, B)
return T ∈ argmax_{i,j} {f(S^i_j), f(x*), f(S_−)}
```

Theorem 1. *For* $\alpha = 2-\sqrt{3}$*,* $p = \frac{1-\alpha}{2}$ *and* $\varepsilon < \frac{1}{3}$*,* PARKNAPSACK *is a* $(9.465+\varepsilon)$*-approximation algorithm with* $O(\frac{1}{\varepsilon}\log n)$ *adaptivity and* $O(\frac{n^2}{\varepsilon^3}\log^2 n \log\frac{1}{\varepsilon})$ *total queries.*

Proof. Excluding the call to SUBMODMAX, the claim on the adaptivity follows directly from Lemma 3 with $\ell = O(\varepsilon^{-2})$, and the observation that $\kappa(\mathcal{N}_+) \le n$. The adaptivity is indeed only due to THRESHSEQ, since the guessing of the threshold, as well as the multiple runs of THRESHSEQ, happen independently in parallel. Relative to the query complexity, we have the bound in Lemma 3 multiplied by an extra $O(\frac{\log n}{\varepsilon^2}\log\frac{1}{\varepsilon})$ factor caused by the two *for* loops. SUBMODMAX does not affect these asymptotics since it has adaptivity bounded by $O(\frac{1}{\varepsilon})$ and linear query complexity.

Consider now the approximation guarantee. Call O^* the optimal solution, and O^+, O^- its intersections with $\mathcal{N}_+$ and $\mathcal{N}_-$ respectively. We can upper bound $f(O^-)$ with the unconstrained max on $\mathcal{N}_-$, since there are at most n elements in $\mathcal{N}_-$ whose cost is at most $\frac{B}{n}$. Using the combinatorial algorithm of Chen et al. (2019), we get

$$f(O^-) \le (2+\hat{\varepsilon}) \cdot f(S_-) \le (2+\hat{\varepsilon}) \cdot ALG\,. \quad (4)$$

Let $O \in \arg\max\{f(T) : T \subseteq \mathcal{N}_+,\ c(T) \le B\}$, i.e., O is an optimal solution in $\mathcal{N}_+$. Clearly $f(O^+) \le f(O)$, so we will upper bound the latter. Let $O_H = O \cap H$. By submodularity and monotonicity of $f(\cdot \cap O)$, we have $pf(O) \le \mathbb{E}[f(O_H)]$. Outside O, the function may be non-monotone, so we need Lemma 2. In particular, we apply it on the submodular function $g(\cdot) = f(\cdot \cup O)$. Since elements belong to H with probability p, for $S \subseteq H$ we get

$$\begin{aligned} p(1-p)f(O) &\le (1-p)\mathbb{E}[f(O_H)] \quad (5)\\ &\le \mathbb{E}[\mathbb{E}[f(S \cup O_H) \,|\, O_H]]\\ &= \mathbb{E}[f(S \cup O_H)]\,. \end{aligned}$$

Let $\tau^* = \alpha f(O)/B$. By the parallel guesses we have that there exists $\tau = \tau_i$ such that $(1-\hat{\varepsilon})\tau^* \le \tau < \tau^*$. This directly follows from the definitions of τ^* and $\hat{\tau}$ and the fact that $nf(x^*) \ge f(O) \ge f(x^*)$. We focus only on this particular τ and consider two cases, depending on $\mathbb{E}[c(S)]$, where S is the set outputted by THRESHSEQ for τ. If $\mathbb{E}[c(S)] \ge (1-\hat{\varepsilon})\frac{B}{2}$, then, from Lemma 4 we have

$$\begin{aligned} ALG &\ge \mathbb{E}[f(S)] \ge (1-\hat{\varepsilon})^2\tau\mathbb{E}[c(S)] \quad (6)\\ &\ge (1-\hat{\varepsilon})^3\alpha f(O)\frac{\mathbb{E}[c(S)]}{B} \ge (1-\hat{\varepsilon})^4\frac{\alpha}{2}f(O)\,. \end{aligned}$$

If $\mathbb{E}[c(S)] < (1-\hat{\varepsilon})\frac{B}{2}$ we need a more careful analysis, via Lemmata 5 and 6. Consider the multiple runs of THRESHSEQ corresponding to τ. Let $\mathcal{G}$ be the event that at least one of those runs outputs S with $c(S) < \frac{B}{2}$ and consider that solution; recall that $\mathbb{P}(\mathcal{G}) \ge (1-\hat{\varepsilon})$ from Lemma 6. What we want to bound is the total value of the elements of O_H which are not in S. The ones retaining a good marginal density with respect to S can be divided into two categories, depending on the reason why they were not added to S:

$$\begin{aligned} G &= \{x \in H : f(x \,|\, S) \ge \tau c(x), c(x) + c(S) \le B\}\,,\\ \tilde{G} &= \{x \in H : f(x \,|\, S) \ge \tau c(x),\ c(x) + c(S) > B\}\,. \end{aligned}$$

The total contribution of the elements in G can be bounded applying Lemma 5. For $\tilde{G} \cap O_H$ we know that it contains at most one element $\tilde{x}$, since we are conditioning on $\mathcal{G}$ and thus, if such $\tilde{x}$ exists, $c(\tilde{x}) > \frac{B}{2}$. Moreover, $f(\tilde{x}) \le f(x^*)$. Finally, we know that the marginal density of all the other elements in $O_H \setminus S$ is at most τ. Let $\mathcal{E}$ be the event that $\tilde{G} \cap O_H \neq \emptyset$ given $\mathcal{G}$ and q its probability. We have

$$\begin{aligned} f(S \cup O_H) &\le f(S) + \mathbf{1}_{\mathcal{E}} \cdot f(\tilde{x} \,|\, S)\\ &\quad + \sum_{x \in G} f(x \,|\, S) + \sum_{x \in O_H \setminus (G \cup \tilde{G})} f(x \,|\, S)\\ &\le f(S)(1+\hat{\varepsilon}) + \mathbf{1}_{\mathcal{E}} \cdot (f(\tilde{x} \,|\, S) - \tau c(\tilde{x})) + \tau c(O_H)\\ &\le f(S)(1+\hat{\varepsilon}) + \mathbf{1}_{\mathcal{E}} \cdot (f(x^*) - \tau\tfrac{B}{2}) + \tau c(O_H)\,. \end{aligned}$$

Keeping fixed H, let's apply the expectation on the randomness in THRESHSEQ, conditioning on $\mathcal{G}$ and recalling that both $f(S)$ and $f(x^*)$ are upper bounded by ALG:

$$\mathbb{E}\left[f(S \cup O_H)|\mathcal{G}\right] \leq (1+\hat{\varepsilon}+q)ALG + \tau c(O_H) - q\tau\tfrac{B}{2}\,.$$

Now move on to the expectation with respect to H. Note that by submodularity $f(S \cup O_H) \leq 2f(O)$. We have

$$\begin{aligned}\mathbb{E}\left[f(S \cup O_H)\right] &= \mathbb{E}\left[f(S \cup O_H)|\mathcal{G}\right]\mathbb{P}\left(\mathcal{G}\right) \\ &\quad + \mathbb{E}\left[f(S \cup O_H)|\mathcal{G}^C\right]\mathbb{P}\left(\mathcal{G}^C\right) \\ &\leq \mathbb{E}\left[f(S \cup O_H)|\mathcal{G}\right] + 2\hat{\varepsilon}f(O)\,.\end{aligned}$$

Putting together the last two inequalities and recalling that $\mathbb{E}\left[c(O_H)\right] = pc(O) \leq pB$, we have

$$\mathbb{E}\left[f(S \cup O_H)\right] \leq (2\hat{\varepsilon}+\alpha-q\tfrac{\alpha}{2})f(O) + (1+\hat{\varepsilon}+q)ALG\,.$$

Combining that with Equation (5), we finally obtain

$$f(O) \leq \frac{(1+q+\hat{\varepsilon})}{\left[p(1-p)-\alpha p+\frac{\alpha q}{2}-2\hat{\varepsilon}\right]}ALG\,. \qquad (7)$$

At this point we need to optimize the constants in Equations (4), (5) and (7), also using that $OPT \leq f(O^+) + f(O^-) \leq f(O) + f(O^-)$. Setting $p = \frac{1}{2}(\sqrt{3}-1)$, $\alpha = 2-\sqrt{3}$ and rescaling $\hat{\varepsilon} = \frac{\varepsilon}{125}$ we get, for small enough $\hat{\varepsilon}$ and *for any* value of $q \in (0,1)$ the desired bound: $OPT \leq (2(3+\sqrt{3})+\varepsilon)ALG$. □

3.1. Variants and Implications

As mentioned already, an interesting feature of our approach is that—with few modifications—yields a number of algorithms that match or improve the state-of-the-art. Due to space constraints we only sketch these modifications here and we defer the details to the supplementary material.

We begin with a discussion on the possible trade-offs between adaptivity and query complexity. THRESHSEQ can be adapted to spare $\Theta(\frac{n}{\log n})$ value queries at the cost of $O(\log n)$ extra adaptive rounds. The idea is to use *binary search* to locate k^* in the while loop of THRESHSEQ. Only a logarithmic number of prefixes needs to be sequentially considered, instead of all of them in parallel. To be able to binary search k^*, though, a carefully modified version of the value condition is implemented, since the one used in THRESHSEQ exhibits a multi-modal behaviour.

Theorem 2. *For $\varepsilon \in (0,1/3)$, it is possible to achieve a $(9.465+\varepsilon)$-approximation in $O(\frac{1}{\varepsilon}\log^2 n)$ adaptive rounds and $O(\frac{n}{\varepsilon^3}\log^3 n \log\frac{1}{\varepsilon})$ queries.*

Monotone Submodular Functions. For monotone objectives, the approximation ratio of PARKNAPSACK can be significantly improved. In particular, in THRESHSEQ, we do not need to address the value condition any more. Moreover, the small elements can be accounted for without any extra loss in the approximation. As in the case of Theorems 1 and 2, it is possible to trade a logarithmic loss in adaptivity for an almost linear gain in query complexity.

Theorem 3. *For $\varepsilon \in (0,1)$ it is possible to achieve a $3+\varepsilon$ approximation in $O(\frac{1}{\varepsilon}\log n)$ adaptive rounds and $O(\frac{n^2}{\varepsilon^3}\log^2 n \log\frac{1}{\varepsilon})$ queries* or *in $O(\frac{1}{\varepsilon}\log^2 n)$ adaptive rounds and $O(\frac{n}{\varepsilon^3}\log^3 n \log\frac{1}{\varepsilon})$ queries.*

Note that the variant using $\tilde{O}(n)$ queries is the first $O(1)$-approximation algorithm for the problem combining this few queries with sublinear adaptivity.

Cardinality Constraints. PARKNAPSACK can be directly applied to cardinality constraints for (possibly) non-monotone objectives. Again with some simple modifications, it is possible to achieve a much better approximation.

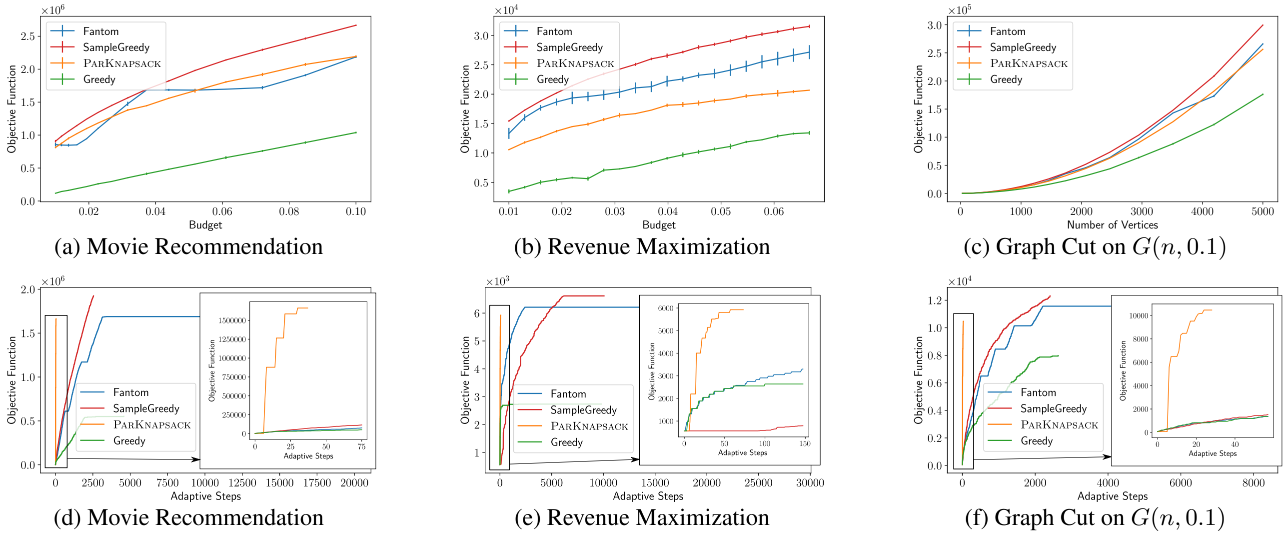
Theorem 4. *For $\varepsilon \in (0,2/5)$ it is possible to achieve a $5.83+\varepsilon$ approximation, in $O(\frac{1}{\varepsilon}\log n)$ adaptive rounds and $O(\frac{nk}{\varepsilon^3}\log n \log k \log\frac{1}{\varepsilon})$ queries,* or *in $O(\frac{1}{\varepsilon}\log n \log k)$ adaptive rounds and $O(\frac{n}{\varepsilon^3}\log n \log^2 k \log(\frac{1}{\varepsilon}))$ queries.*

Although we do not heavily adjust our algorithms to cardinality constraints, Theorem 4 is directly comparable to the very recent results of Ene & Nguyen (2020) and Kuhnle (2021) which are tailored for the problem.

Finally, it is natural to compare our PARKNAPSACK and the ADAPTIVE SEQUENCING of Balkanski et al. (2019b) for cardinality constraints and monotone objectives. While the corresponding versions of our SAMPLESEQ and their RANDOM SEQUENCE coincide, the full algorithms do not. The reason is that the different thresholds in ADAPTIVE SEQUENCING are used for the same solution, and thus the algorithm cannot be parallelized to the extent that PARKNAPSACK can. Each call to our THRESHSEQ uses *a single* threshold and so all of them can run in parallel. The result is that PARKNAPSACK is $O(\log n)$-adaptive while ADAPTIVE SEQUENCING is $O(\log n \log k)$-adaptive.

4. Experiments

We evaluate the performance of PARKNAPSACK on real datasets and real-world applications, as is often the case in the related literature (Mirzasoleiman et al., 2016; Fahrbach et al., 2019b; Amanatidis et al., 2020; Breuer et al., 2020; Kuhnle, 2021). All three objectives we use are non-monotone submodular. In our first set of experiments (Figure 1), we compare against the state-of-the-art of *fast* algorithms for non-monotone submodular maximization subject to a knapsack constraint, in order to demonstrate that PARKNAPSACK produces almost equally good solutions with an exponential improvement on the adaptivity. We



(a) Movie Recommendation (b) Revenue Maximization (c) Graph Cut on $G(n, 0.1)$

(d) Movie Recommendation (e) Revenue Maximization (f) Graph Cut on $G(n, 0.1)$

Figure 1: Each of the columns contains two plots, corresponding to the same submodular problem. The top row contains the objective function value for different budget or instance sizes. The bottom row focuses on one vertical slice of the top one: for a specific budget and instance size, the objective value is presented as a function of the number of adaptive rounds, indicating that if we required to stop after a small number of rounds all other algorithms would perform extremely poorly. In all cases, the results are consistent: PARKNAPSACK has comparable performance, with drastically improved adaptivity.

provide two kinds of figures: objective versus budget (or instance size) and objective versus adaptive steps, for a given instance, as in Balkanski et al. (2018) and Fahrbach et al. (2019b). We refer to the supplementary material for the exact setup and implementation details, but we note that we use the version of PARKNAPSACK from Theorem 2 to ensure $\tilde{O}(n)$ query complexity. The benchmarks we use are plain GREEDY, FANTOM of Mirzasoleiman et al. (2016) and SAMPLEGREEDY of Amanatidis et al. (2020). The last two have the state-of-the-art performance in terms of objective value for knapsack constraints among algorithms with practical running times, i.e., among algorithms with subquadratic query complexity. On the other hand, these algorithms are not designed for low adaptivity but, the only alternative, i.e., continuous methods, are impractical for the instance sizes we consider. The GREEDY algorithm builds a solution step by step by adding the element with the highest marginal value, until the budget is exhausted. While this naive approach has no theoretical guarantees, it is very fast and often has acceptable performance in practice. FANTOM builds on GREEDY and is robust for intersecting p-systems and knapsack constraints providing a $10(1 + \epsilon)$-approximation for our setting. Finally, SAMPLEGREEDY greedily selects elements according to their marginal value per cost ratio, but only adds them to the solution with some probability. This leads to a 5.83-approximation. These algorithms need $O(n \log n)$ queries and adaptive steps, when implemented using with *lazy evaluations* (Minoux, 1978).

The obvious question here is how PARKNAPSACK performs against the only other low-adaptivity algorithm for non-monotone submodular objectives and a knapsack constraint, namely Algorithm 3 of (Ene et al., 2019)—*the ENV algorithm* for short. The main reason the ENV algorithm was not included in the experiments of Figure 1 is that it is infeasible to run it for instances of that size. Note that that the ENV algorithm uses the multilinear extension F of the objective f and each evaluation of F requires $\tilde{\Theta}(n^2)$ evaluations of f. So, in our second set of experiments (Figure 2), we focus on smaller instances of Maximum Weighted Cut and directly compare PARKNAPSACK and ENV with respect to their adaptivity and the quality of their output. From the experiments it results that PARKNAPSACKoutperforms ENV both in terms of the adaptive complexity and the objective. We use $\varepsilon \leq 0.1$ for ENV, because its performance deteriorates quickly as ε grows, even after some tuning. The issue is that, by design, the ENV algorithm does not perform much better than its theoretical bound in terms of approximation, even on well-behaved instances like we have here. Note that the approximation ratio of ENV is 3.7 for $\varepsilon = 0.1$, but 29 for $\varepsilon = 1/3$.

Movie Recommendation. Given a set of movies A, a list of genres C_i such that $C_1 \cup C_2 \cup \ldots \cup C_k = A$ and a list of user generated keyword tags t_{iu} and ratings r_{iu}, where $i \in A$ and u is the id of a user, a movie recommendation system aims to use this information to provide a short list of diverse options that match certain preferences. The MovieLens dataset (Harper & Konstan, 2016) provides a very large set of movies that include user gen-

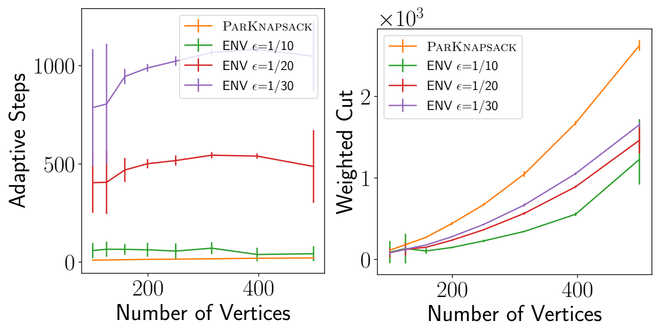


Figure 2: The ENV algorithm actually finds a *fractional* solution, which can be then rounded using some approximation preserving scheme. We have chosen to omit this rounding and report the performance of the fractional solution, to present a version of the algorithm that is as close to the original as possible.

erated tags and ratings. We calculate the similarity between two movies (following the procedure of Amanatidis et al. (2020); see also the supplementary material) and produce a weighted complete graph, where each vertex is a movie. For $i, j \in A$ the weight w_{ij} represents their similarity. In addition, we use χ_{ij} to indicate if the two movies share a genre. Putting everything together, the objective function is: $v(S) = \alpha \sum_{i\in S} r_i + \beta(\sum_{i\in S}\sum_{j\in A} w_{ij} - \sum_{i\in S}\sum_{j\in S}(\lambda + \chi_{ij}\mu)w_{ij})$ for $\lambda, \mu, \alpha, \beta \geq 0$ where r_i represents the average rating of movie i. This is a weighted average of the ratings of the movies in S and a modified *maximal marginal relevance* (Carbinell & Goldstein, 2017). The second part is similar to a max cut (in fact it is a max cut for $\lambda = 1$ and $\mu = 0$), but allows the internal edges to be penalized differently, depending on whether the movies are similar or belong to the same genre. For the experiments we consider a subset of 5000 movies and set $\alpha = \beta = 0.5, \lambda = 3$ and $\mu = 7$. Each movie is assigned a cost sampled uniformly from $[0, 1]$ and the total budget ranges from 0.01 to 0.1 of the total cost.

Revenue Maximization. Representing a social network as a weighted graph, where each edge signifies how much one user is influenced by another, our goal is to select a subset S of users who are given a product to advertise, in order to maximize the revenue from sales. We use the YouTube Network (Yang & Leskovec, 2015), and consider the subgraph induced by selecting its Top 5000 communities, which has 39841 vertices and 224235 edges. We assign edge weights w_{ij} sampled uniformly in $[0, 1]$ and each user $i \in V$ has a suggestibility parameter α_i drawn from a Pareto Type II distribution with $\lambda = 1, \alpha = 2$. The objective to maximize is: $v(S) = \sum_{i\in V\setminus S} \alpha_i \sqrt{\sum_{j\in S} w_{ij}}$. Each user is assigned a cost proportional to their incident edges, with the budget ranging from 0.01 to 0.1 of the total cost.

Maximum Weighted Cut. Given an Erdős–Rényi graph $G(n, p)$ where n is the number of vertices and p the probability of including each edge, the objective is to find a *cut of maximum weight*. Fixing $p = 0.1$, we let $n \in \{30, \dots, 5000\}$ (with an exponential step) and assign random edge weights and costs sampled uniformly from $[0, 1]$, while the budget is fixed at 15% of the total cost.

5. Conclusions

In this paper we close the gap for the adaptive complexity of non-monotone submodular maximization subject to a knapsack constraint, up to a $O(\log \log n)$ factor. Our algorithm, ParKnapsack, is combinatorial and can be modified to achieve trade-offs between adaptivity and query complexity. In particular, it may use nearly linear queries, while achieving an exponential improvement on adaptivity compared to existing algorithms with subquadratic query complexity.

Acknowledgment

This work was supported by the ERC Advanced Grant 788893 AMDROMA "Algorithmic and Mechanism Design Research in Online Markets", the MIUR PRIN project ALGADIMAR "Algorithms, Games, and Digital Markets", and the NWO Veni project No. VI.Veni.192.153.

References

Amanatidis, G., Fusco, F., Lazos, P., Leonardi, S., and Reiffenhäuser, R. Fast adaptive non-monotone submodular maximization subject to a knapsack constraint. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

Badanidiyuru, A., Karbasi, A., Kazemi, E., and Vondrák, J. Submodular maximization through barrier functions. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

Balkanski, E. and Singer, Y. 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*, pp. 1138–1151. ACM, 2018.

Balkanski, E., Breuer, A., and Singer, Y. Non-monotone submodular maximization in exponentially fewer iterations. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information*

Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada, pp. 2359–2370, 2018.

Balkanski, E., Rubinstein, A., and Singer, Y. An exponential speedup in parallel running time for submodular maximization without loss in approximation. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pp. 283–302. SIAM, 2019a.

Balkanski, E., Rubinstein, A., and Singer, Y. An optimal approximation for submodular maximization under a matroid constraint in the adaptive complexity model. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pp. 66–77. ACM, 2019b.

Breuer, A., Balkanski, E., and Singer, Y. The FAST algorithm for submodular maximization. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pp. 1134–1143. PMLR, 2020.

Buchbinder, N., Feldman, M., Naor, J., and Schwartz, R. 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*, pp. 1433–1452. SIAM, 2014.

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

Chekuri, C. and Quanrud, K. Submodular function maximization in parallel via the multilinear relaxation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pp. 303–322. SIAM, 2019a.

Chekuri, C. and Quanrud, K. Parallelizing greedy for submodular set function maximization in matroids and beyond. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pp. 78–89. ACM, 2019b.

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

Chen, L., Feldman, M., and Karbasi, A. Unconstrained submodular maximization with constant adaptive complexity. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pp. 102–113. ACM, 2019.

Das, A. and Kempe, D. 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*, pp. 45–54. ACM, 2008.

Das, A. and Kempe, D. Approximate submodularity and its applications: Subset selection, sparse approximation and dictionary selection. *J. Mach. Learn. Res.*, 19:3:1–3:34, 2018.

Dueck, D. and Frey, B. J. Non-metric affinity propagation for unsupervised image categorization. In *IEEE 11th International Conference on Computer Vision, ICCV 2007, Rio de Janeiro, Brazil, October 14-20, 2007*, pp. 1–8. IEEE Computer Society, 2007.

Duris, P., Galil, Z., and Schnitger, G. Lower bounds on communication complexity. *Inf. Comput.*, 73(1):1–22, 1987.

Ene, A. and Nguyen, H. L. Submodular maximization with nearly-optimal approximation and adaptivity in nearly-linear time. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pp. 274–282. SIAM, 2019.

Ene, A. and Nguyen, H. L. Parallel algorithm for non-monotone DR-submodular maximization. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pp. 2902–2911. PMLR, 2020.

Ene, A., Nguyen, H. L., and Vladu, A. A parallel double greedy algorithm for submodular maximization. *CoRR*, abs/1812.01591, 2018.

Ene, A., Nguyen, H. L., and Vladu, A. Submodular maximization with matroid and packing constraints in parallel. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pp. 90–101. ACM, 2019.

Fahrbach, M., Mirrokni, V. S., and Zadimoghaddam, M. 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*, pp. 255–273. SIAM, 2019a.

Fahrbach, M., Mirrokni, V. S., and Zadimoghaddam, M. 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*, pp. 1833–1842. PMLR, 2019b.

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

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

Feldman, M., Naor, J., and Schwartz, R. 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*, pp. 570–579. IEEE Computer Society, 2011.

Gupta, A., Roth, A., Schoenebeck, G., and Talwar, K. 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 *Lecture Notes in Computer Science*, pp. 246–257. Springer, 2010.

Harper, F. M. and Konstan, J. A. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5 (4):19:1–19:19, 2016.

Hartline, J. D., Mirrokni, V. S., and Sundararajan, M. 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*, pp. 189–198. ACM, 2008.

Kazemi, E., Zadimoghaddam, M., and Karbasi, A. Scalable deletion-robust submodular maximization: Data summarization with privacy and fairness constraints. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 2549–2558. PMLR, 2018.

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

Khanna, R., Elenberg, E. R., Dimakis, A. G., Negahban, S. N., and Ghosh, J. 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*, pp. 1560–1568. PMLR, 2017.

Kuhnle, A. Nearly linear-time, parallelizable algorithms for non-monotone submodular maximization. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pp. 8200–8208. AAAI Press, 2021.

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

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

Mirzasoleiman, B., Karbasi, A., Sarkar, R., and Krause, A. 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. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pp. 2049–2057, 2013.

Mirzasoleiman, B., Badanidiyuru, A., and Karbasi, A. 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 Workshop and Conference Proceedings*, pp. 1358–1367. JMLR.org, 2016.

Mirzasoleiman, B., Bilmes, J. A., and Leskovec, J. Coresets for data-efficient training of machine learning models. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pp. 6950–6960. PMLR, 2020.

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

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

Tschiatschek, S., Iyer, R. K., Wei, H., and Bilmes, J. A. 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, December 8-13 2014, Montreal, Quebec, Canada, pp. 1413–1421, 2014.

Valiant, L. G. Parallelism in comparison problems. *SIAM J. Comput.*, 4(3):348–355, 1975.

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