Online and Streaming Algorithms for Constrained k-Submodular Maximization

Fabian Christian Spaeh,[1] **Alina Ene,**[1] **Huy Nguyen**[2]

[1]Department of Computer Science, Boston University
[2]Khoury College of Computer Science, Northeastern University
fspaeh@bu.edu, aene@bu.edu, hu.nguyen@northeastern.edu

Abstract

Constrained k-submodular maximization is a general framework that captures many discrete optimization problems such as ad allocation, influence maximization, personalized recommendation, and many others. In many of these applications, datasets are large or decisions need to be made in an online manner, which motivates the development of efficient streaming and online algorithms. In this work, we develop single-pass streaming and online algorithms for constrained k-submodular maximization with both monotone and general (possibly non-monotone) objectives subject to cardinality and knapsack constraints. Our algorithms achieve provable constant-factor approximation guarantees which improve upon the state of the art in almost all settings. Moreover, they achieve the fastest known running times and have optimal space usage. We experimentally evaluate our algorithms on instances for ad allocation and other applications, where we observe that our algorithms are practical and scalable, and construct solutions that are comparable in value even to offline greedy algorithms.

1 Introduction

We develop algorithms for maximizing a k-submodular function f subject to cardinality or knapsack constraints. k-Submodular functions capture the property of diminishing returns under an allocation of elements from a ground set V to k parts. Specifically, we are trying to find k disjoint subsets $(S_1, \dots, S_k)$ of V such that $f(S_1, \dots, S_k)$ is maximized. We consider constrained k-submodular maximization where we enforce individual constraints for each part: Each part $a \in \{1, \dots, k\}$ has a specified budget n_a and we are only allowed to allocate at most $|S_a| \leq n_a$ items.

k-Submodular functions model several important applications such as ad allocation. In this problem, ad impressions arrive online and we have to allocate them immediately to one of k advertisers. At the same time, each advertiser $a \in \{1, \dots, k\}$ is willing to pay for at most n_a ad impressions (Feldman et al. 2009). The advertising platform makes an allocation to maximize advertiser satisfaction, for which natural objectives such as user exposure show diminishing returns. Another important application is in personalized recommendation, which motivates the study of general (possibly non-monotone) objectives. Mirzasoleiman, Badanidiyuru, and Karbasi (2016) show how to model a recommender system through cut functions in dissimilarity graphs, which are general k-submodular. Related tasks such as document or image summarization (Lin and Bilmes 2011; Gomes and Krause 2010) can be modeled through similar objectives. Additional motivation on influence maximization, sensor placement, and video summarization is given in the works of Ohsaka and Yoshida (2015a) and Feldman, Karbasi, and Kazemi (2018). These further applications also motivate the study of more general packing constrains such as knapsack, where items take up non-uniform space in the budget constraints.

The datasets used in all of these applications are typically large and even offline greedy algorithms are often impractical. This necessitates streaming algorithms which only make one or multiple passes over the data and require little space. Furthermore, applications such ad allocation require us to make decisions online: Upon arrival, we need to immediately allocate a user to an advertiser without the chance of re-allocating later. Algorithms that work simultaneously in the online and streaming setting are therefore desirable and well-studied (Ene and Nguyen 2022; Feldman et al. 2021). We contribute to this line of work by designing algorithms with improved approximation guarantees. We make further progress with novel algorithms for more general settings.

Our Contributions and Techniques: Our key contributions for the online and streaming setting are:

1. We introduce a novel, general algorithm and analysis framework. Algorithms in our framework are combinatorial and very efficient, have optimal space and attain the best known running time for our setting. Our framework does not only allow us to improve upon prior results in a unified way, but also handles much more general non-monotone objectives and budget constraints.

2. Our framework allows us to address settings that were previously out of reach. For general (possibly non-monotone) k-submodular maximization, we are the first to obtain a constant-factor approximation, which was previously not even known in the offline setting (Xiao et al. 2022). We are also first to provide an algorithm for k-submodular maximization subject to individual knapsack constraints (here, items do not have uniform size in the budget constraints), where prior work is restricted to a common constraint (Pham et al. 2022; Ha, Pham, and Tran 2024).

Objective	Reference	Setting	Approx.	Time	Space
monotone	(Ohsaka and Yoshida 2015b)	offline	$\frac{1}{3}$	$O(mkr)$	$O(m)$
	(Ene and Nguyen 2022)	online, streaming	$\geq \frac{1}{4}$ ≈ 0.2953 as $n \to \infty$	$O(mk)$	$O(r)$
	Theorem 1 **(This paper)**	online, streaming	$\geq \frac{1}{4}$ ≈ 0.3178 as $n \to \infty$	$O(mk)$	$O(r)$
general	(Xiao et al. 2022)	offline	$\frac{1}{4+\max_a n_a}$	$O(rmk)$	$O(r)$
	Theorem 2 **(This paper)**	online, streaming	$\geq \frac{1}{8}$ ≈ 0.1589 as $n \to \infty$	$O(mk)$	$O(r)$

Table 1: Comparison of algorithms for k-submodular maximization with cardinality constraints. We let $n = \min_{a \in [k]} n_a$ denote the minimum budget, $r = \sum_{a \in [k]} n_a$ the total budget, and $m = |V|$.

3. Our algorithms achieve provable constant factor approximation guarantees that improve upon the state of the art for k-submodular maximization (cf. Table 1). Due to their versatility, they also apply to the related and practically relevant setting of submodular maximization with a partition matroid. Here, we improve upon the guarantee of Feldman, Karbasi, and Kazemi (2018) for general objectives and close the gap to less efficient streaming algorithms (cf. Table 4). We achieve this by introducing new techniques that are suited for general (possibly non-monotone) objectives.

These contributions are in the context of individual constraints, which is what we discuss in the main body. Starting with Ohsaka and Yoshida (2015a), there has been a lot of work for a common constraint on the support, e.g. $|S_1 \cup S_2 \cup \cdots \cup S_k| \leq n$ for a cardinality constraint Ha, Pham, and Tran (2024); Yu et al. (2023); Wang and Zhou (2021); Sakaue (2017); Nguyen and Thai (2020a). Such constraints cannot capture individual constraints. However, applying our framework to the common budget setting yields efficient online and streaming algorithms, which we outline in Section 3.3 and formally discuss in Appendix D.

Related Work: The main focus of our work is on individual constraints and we therefore discuss only work related to this setting in the main body. However, our techniques also apply to the setting with a common constraint, and we also obtain further results for knapsack constraints. We discuss this in Appendix D along the relevant related work.

Offline algorithms for k-submodular maximization. Most results for k-submodular maximization are in the offline setting. For monotone objectives, Ohsaka and Yoshida (2015a) provide a greedy algorithm that achieves a $\frac{1}{3}$-approximation. The only prior work on general k-submodular maximization is due to Xiao et al. (2022) and in the offline setting. Their greedy approach obtains a $\frac{1}{4+\max_a n_a}$-approximation, which decreases with the maximum budget.

Online and streaming algorithms for k-submodular maximization. We consider the challenging and practically relevant online and streaming setting. The only work in this space is due to Ene and Nguyen (2022) and applies only to monotone objectives. For monotone objectives, our algorithms achieve an improved approximation guarantee as shown in Table 1. We give the first algorithm for general objectives with constant-factor approximation guarantees. Such a result was not even known in the offline setting.

Submodular maximization with a partition matroid. A related and important setting is submodular maximization with a partition matroid. Feldman et al. (2009) provide a seminal online algorithm for linear objectives based on the primal-dual approach. For general submodular objectives under a matroid constraint, Feldman et al. (2022) give a streaming algorithm based on the continuous extension of a submodular function, whose evaluation is costly. Their algorithm also maintains multiple solutions and is thus not suited for the online setting. Our discrete algorithms achieve the same guarantees when the minimum budget tends to infinity, which is a necessary assumption for the online setting (Feldman et al. 2009). In the online setting, Feldman, Karbasi, and Kazemi (2018) give a discrete algorithm for general objectives under p-matchoid constraints. Their algorithm subsamples items which is also a technique we employ, but we obtain an improved approximation ratio.

2 Preliminaries

k-Submodular functions. We define a partial allocation $(k+1)^V$ as the set of all k-tuples $(X_1, \ldots, X_k)$ of disjoint subsets $X_a \subseteq V, X_a \cap X_b = \emptyset$ for all $a, b \in [k]$ where $[k] := \{1, 2, \ldots, k\}$. For a k-tuple $\mathbf{X} \in (k+1)^V$, we define the support $\mathbf{supp}(\mathbf{X}) := X_1 \cup \cdots \cup X_k$ as all allocated items. Given another k-tuple $\mathbf{Y} \in (k+1)^V$, we define the intersection $\mathbf{X} \sqcap \mathbf{Y}$ of two k-tuples through $(\mathbf{X} \sqcap \mathbf{Y})_a := X_a \cap Y_a$ for all $a \in [k]$, and the union as $(\mathbf{X} \sqcup \mathbf{Y})_a := (X_a \cup Y_a) \setminus \bigcup_{b \neq a}(X_b \cup Y_b)$. Given these operations, we say $f\colon (k+1)^V \to \mathbb{R}$ is k-submodular if

$$f(\mathbf{X}) + f(\mathbf{Y}) \geq f(\mathbf{X} \sqcap \mathbf{Y}) + f(\mathbf{X} \sqcup \mathbf{Y})$$

for all $\mathbf{X}, \mathbf{Y} \in (k+1)^V$. We define the marginal gain of adding element t to part a of $\mathbf{X}$ as

$$\Delta_{t,a} f(\mathbf{X}) := f\left((X_1, \ldots, X_a \cup \{t\}, \ldots, X_k)\right) - f(\mathbf{X}).$$

We further write $\mathbf{X} \preceq \mathbf{Y}$ if $X_a \subseteq Y_a$ for all $a \in [k]$ and say f is monotone if $f(\mathbf{X}) \leq f(\mathbf{Y})$ for all $\mathbf{X} \preceq \mathbf{Y}$. To obtain a notion of diminishing returns, we say that f is orthant submodular if

$$\Delta_{t,a} f(\mathbf{X}) \geq \Delta_{t,a} f(\mathbf{Y})$$

for all $\mathbf{X} \preceq \mathbf{Y}$ and $t \notin \mathbf{supp}(\mathbf{Y})$. We say f is pairwise monotone if for all $t \notin \mathbf{supp}(\mathbf{X})$ and $a \neq b$,

$$\Delta_{t,a} f(\mathbf{X}) + \Delta_{t,b} f(\mathbf{X}) \geq 0.$$

We know that f is k-submodular if and only if f is orthant submodular and pairwise monotone (Ward and Zivný 2016). A function is called submodular if it is 1-submodular.

Problem definition. In k-submodular maximization, we are given a k-submodular function and budgets $n_1, \dots, n_k$ for every part. The goal is to find a solution that maximizes f while allocating at most n_a items to every part a, and we denote the optimum solution as $\mathbf{S}^*$. More general knapsack constraints and the setting with a common constraint are defined in Appendix D.

We consider both problems in the (single-pass) streaming model. Here, all items of V arrive in an arbitrary (possibly adversarial) order and the task is to generate a solution at the end of the stream, while using as little space as possible. Our algorithms simultaneously apply to the online setting with free disposal (Feldman et al. 2009). Here, items also arrive one at a time, but we are required to maintain a single solution after each arrival. Additionally, we are only allowed to add the arriving item to the solution or dispose (i.e. remove) an item from the solution, but cannot re-allocate.

Examples of k-submodular functions. We now give examples of k-submodular functions that arise in the applications to ad allocation and recommender systems discussed in the introduction and our experiments. The well-studied submodular welfare problem is a special case of k-submodular maximization. Here, the goal is to maximize the welfare $f(\mathbf{X}) = \sum_{a=1}^{k} h_a(X_a)$ where $h_a \colon 2^V \to \mathbb{R}_+$ is the valuation for agent $a \in [k]$. If the h_a's are submodular then f is orthant submodular and if they are monotone, then f is monotone. Such instances appear for ad allocation where advertiser satisfaction can be modeled through a function h_a that expresses, for example, the coverage of an ad campaign. If $h_a = h$ where h is a submodular function that is *symmetric* (i.e., $h(X) = h(V \setminus X)$ for all $X \subseteq V$), then f is general k-submodular (i.e., it is pairwise monotone and orthant submodular). Such instances arise from graph cut functions in applications such as recommender systems.

3 k-Submodular Maximization

In the following, we illustrate our framework on monotone k-submodular maximization, as it is a fundamental setting that allows us to convey the core ideas and novelties of our algorithm and analysis. Section 3.1 contains an algorithm description, intuition, and analysis overview. In Section 3.2, we give a short overview of how additional novel ideas allow us to adapt our framework to obtain the first constant factor approximation algorithm for general k-submodular maximization. Algorithms and analysis for both settings are described in full detail in Appendix A. The algorithm and analysis framework that we develop in this section is general and with some additional work applies to related settings, and we outline these contributions in Section 3.3.

3.1 Monotone Objectives

Our algorithm for maximizing a monotone k-submodular function is shown in Algorithm 1. As shown in Table 1, this improves upon the approximation guarantee of Ene and Nguyen (2022).

Algorithm 1 Monotone k-submodular maximization.

```
Parameters: coefficients $\{g_a(i)\}_{a \in [k], i \in [n_a]}$
Input: a monotone $k$-submodular function $f$ and the budgets $\{n_a\}_{a \in [k]}$
$\mathbf{S} = (S_1, \dots, S_k) \leftarrow (\emptyset, \dots, \emptyset)$
$\beta_a \leftarrow 0$ for all $a \in [k]$
for $t = 1, 2, \dots, |V|$:
    let $w_{t,a} = \Delta_{t,a} f(\mathbf{S})$ for all $a \in [k]$
    let $a = \arg\max_{a \in [k]} \{w_{t,a} - \beta_a\}$
    if $w_{t,a} - \beta_a \geq 0$:
        if $|S_a| < n_a$:
            $S_a \leftarrow S_a \cup \{t\}$
        else:
            let $t' = \arg\min_{i \in S_a} w_{i,a}$
            $S_a \leftarrow (S_a \setminus \{t'\}) \cup \{t\}$
        let $w_a(i)$ be the $i$-th largest weight in $\{w_{t,a} \colon t \in S_a\}$
                                        and $w_a(i) = 0$ for $i > |S_a|$
        $\beta_a \leftarrow \sum_{i=1}^{n_a} w_a(i) g_a(i)$
return S
```

The algorithm maintains thresholds β_a that ensure high gain and balance the allocation. On arrival of each item t, we evaluate its marginal gains for each part a with respect to the current solution $\mathbf{S}$. We denote these marginal gains as weights $w_{t,a}$. The algorithm decides whether to allocate the item and to which part using the discounted weights $w_{t,a} - \beta_a$ for all parts $a \in [k]$. The algorithm chooses the part with largest discounted weight and allocates only if the discounted weight is non-negative. Thus, β_a acts as a threshold that the weight of item t has to pass to be added to the solution. A critical design consideration is how to set the thresholds β_a, and we set them to a carefully designed linear combination of the weights of currently allocated items along the coefficients

$$g_a(i) \coloneqq \frac{c_a}{n_a}\left(1 + \frac{d_a}{n_a}\right)^{i-1} \quad \text{for} \quad c_a \coloneqq \frac{1 + d_a}{\left(1 + \frac{d_a}{n_a}\right)^{n_a} - 1}$$

for all $i \in [n_a]$ with constants d_a, which we will specify in Theorem 1 according to the budget n_a. If the chosen part is at capacity, we dispose of the item with lowest weight.

Intuition. Our basic design can be understood for linear objectives, where the weights $w_{t,a}$ and the item values are identical. Let us consider this setting first. Here, a natural approach is a greedy allocation rule where the discounted weight is the value of the new item minus the value of the item that will be disposed (or 0 if no disposal is required). A crucial downside is that the greedy scheme fails to balance the allocation among the parts, which is necessary to defend against future inputs. We illustrate this with an example in Appendix A.2 due to Feldman et al. (2009). A solution is to consider the allocation by discounting the value of a new item by an exponential average over the values of all items currently allocated to a part.

In the submodular setting, an important difficulty is that the change in objective value when adding and disposing an item is no longer fixed after we modify the current solution. We therefore use the marginal gain $w_{t,a}$ when an item ar-

n_a	1	2	3	≥ 4
d_a	1	1.0642	1.0893	1.1461
$\frac{1}{Q_a}$	0.25	≥ 0.2781	≥ 0.2896	$\geq 0.3178\left(1-\frac{0.7681}{n_a}\right)$

Approximation guarantee $\min_a \frac{1}{Q_a}$		
$\min_a n_a$	≤ 3	≥ 4
approx	≥ 0.25	$\geq 0.3178\left(1-\frac{0.7681}{\min_a n_a}\right)$

Table 2: Parameter choices and approximation guarantee for monotone k-submodular maximization. As shown in Equation (3), Q_a reflects the approximation ratio per part, and the overall approximation guarantee is thus given as $\min_a \frac{1}{Q_a}$.

rives as a proxy for this gain or loss. To account for the fact that the loss of disposing an item might be larger than the proxy, we require new items to pass a higher threshold as in the linear setting. As such, it no longer suffices to set the thresholds β_a as a weighted average. Instead, we require a carefully-designed linear combination of the gains to set the thresholds β_a. The appropriate coefficients for each part a are given through the functions $g_a(i)$ and we derive them via a careful and novel analysis which we outline below.

Comparison to previous work. Our algorithm follows a primal-dual design and keeps a threshold for each part. Ene and Nguyen (2022) set thresholds depending on the marginal gains of all previously allocated items, even those that were already disposed. In contrast, we use a different scheme for setting the thresholds using linear combinations of the gains of only the items in the current solution. As such, our approach is related to Feldman et al. (2009) with the notable difference that we no longer use a convex combination of the gains, which is crucial for submodular objectives as discussed above. Our analysis is another significant departure from both prior works: The analysis of Feldman et al. (2009) strongly leverages the special structure of linear functions and does not apply to submodular objectives. Ene and Nguyen (2022) use a global analysis that is tailored to their specific threshold update scheme. In contrast, we use a different approach for updating the thresholds and analyze it via a novel local analysis outlined below. This makes our approach general and flexible, and it allows us to handle both monotone and non-monotone objectives as well as more general packing constraints. It further allows us to choose the coefficients that go into the thresholds, tailored to the specific budget in each part. Hence, we obtain better approximations in challenging settings when budgets are imbalanced. This was not considered in previous works but is important for applications such as ad allocation, which we also showcase experimentally in Table 3.

As is common in works on submodular optimization, we state the running time of our Algorithm 1 in terms of the number of function evaluations. We can easily see that this is $O(|V| \cdot k)$: we consider each item in V exactly once and evaluate the k-submodular function k times per item. The algorithm only stores a single feasible solution in memory, and this dominates the space requirement. The space is therefore optimal, and the running time is the best-known for algorithms in our setting (Ene and Nguyen 2022). We obtain the following approximation guarantee.

Theorem 1. *We make the following choices for the parameters $\{d_a\}_{a\in[k]}$. Let $d = 1.1461$, which is an approximate solution to the equation $e^d - d - 2 = 0$. We set $d_a = d$ if $n_a > n_0 := 3$, and we set d_a as shown in Table 2 if $n_a \leq n_0$. We obtain the approximation guarantees shown in Table 2. Note that the approximation is at least 0.25 for any minimum budget and it tends to ≥ 0.3178 as the minimum budget tends to infinity.*

Overview of the analysis. We now provide an analysis overview for the approximation ratio of Algorithm 1. A complete analysis is in Section A.2 of the appendix. Analyses for other algorithms in this work follow the same proof framework, but require further non-trivial modifications.

We denote with superscript (t) all quantities of the algorithm at the end of iteration t. We denote all quantities at the end of the stream without superscript. Let $T_a^{(t)} = \bigcup_{i=1}^{t} S_a^{(i)}$ be all items that were allocated to a in the first t iterations.

Lower bound on $f(\mathbf{S})$. Our goal is to relate $f(\mathbf{S})$ to the optimum $f(\mathbf{S}^*)$. However, comparing both is difficult as there is no direct relationship between the allocation $\mathbf{S}$ created by our algorithm and the optimum solution $\mathbf{S}^*$. What we can do is to relate both to marginal gains (weights) and thresholds used in the algorithm, and then leverage the algorithm's structure to compare both. Using orthant submodularity, we can construct the following lower bound on the value of the solution $\mathbf{S}$ (Lemma 3):

$$f(\mathbf{S}) \geq \sum_a \sum_{t\in S_a} w_{t,a}. \tag{1}$$

Upper bound on $f(\mathbf{S}^*)$. An upper bound on the optimum value is harder to obtain, since our marginal gains are with respect to the current solution $\mathbf{S}^{(t)}$, and it is unclear how to relate this to the optimum. For submodular functions ($k = 1$), a common approach is to upper bound $f(S^*)$ by $f(S \cup S^*)$ and analyze the latter via the marginal gains. However, this strategy no longer works for k-submodular functions since they are only defined on allocations where each item appears in at most one part. We thus consider a sequence of intermediate solutions $\mathbf{O}^{(t)}$ that agree with $\mathbf{T}^{(t)}$ on items $\{1, \ldots, t\}$ and with $\mathbf{S}^*$ on $\{t+1, \ldots, |V|\}$, and analyze the change in function value $f(\mathbf{O}^{(t-1)}) - f(\mathbf{O}^{(t)})$ for each iteration. With some additional care where we critically use the allocation choice of Algorithm 1, we obtain the following guarantee (Lemma 4):

$$f(\mathbf{S}^*) \leq \sum_a \left(\sum_{t\in T_a} \left(2w_{t,a} - \beta_a^{(t-1)}\right) + n_a\beta_a \right). \tag{2}$$

Due to Equations (1) and (2), we can compare terms on a per-part basis for each $a \in [k]$, and provide a bound Q_a on the ratio between the two such that

$$\sum_{t\in T_a} \left(2w_{t,a} - \beta_a^{(t-1)}\right) + n_a\beta_a \leq Q_a \sum_{t\in S_a} w_{t,a}. \tag{3}$$

This gives us $f(\mathbf{S}^*) \leq Qf(\mathbf{S})$ where we try to make $Q \coloneqq \max_{a\in[k]} Q_a$ as small as possible. Our analysis yields

$$Q_a = (1 + d_a)\left(1 + \tfrac{1}{(1+d_a/n_a)^{n_a}-1}\right) \tag{4}$$

for parameters d_a, which we can optimize according to the budgets as outlined below. First, however, we show how to obtain the per-part approximation ratios Q_a.

Deriving the approximation ratios Q_a. Note that the LHS of (3) has the weights $\{w_{t,a} : t \in T_a\}$ of all of the items ever allocated to a (including discarded items) and the thresholds. In contrast, the RHS of (3) has only the weights $\{w_{t,a} : t \in S_a\}$ of the final solution. Thus we need to relate the weights of the discarded items and the thresholds to the items in the final solution. To this end, we use a primal potential tracking the lower bound (1) and a dual potential tracking the upper bound (2):

$$P_t \coloneqq \sum_{i\in S_a^{(t)}} w_i, \;\; D_t \coloneqq \sum_{i\in T_a^{(t)}} \left(2w_{ai} - \beta_a^{(i-1)}\right) + n_a\beta_a^{(t)}.$$

We interpret the dual D_t as follows: $2w_{at} - \beta_a^{(t-1)}$ is the cost of reallocating an item to the part chosen by the optimum solution, and we use $n_a\beta_a^{(t)}$ to account for items in S_a^* that have not arrived yet by paying the current threshold $\beta_a^{(t)}$ for each of them. Our analysis relates the change in the dual to the change in the primal, in each iteration. If $t \notin T_a$, we experience no change in either primal nor dual. If $t \in T_a$, the change is

$$P_t - P_{t-1} = w_{t,a} - \min_{i\in S_a^{(t-1)}} w_{i,a},$$

$$D_t - D_{t-1} = 2w_{t,a} - \beta_a^{(t-1)} + n_a\big(\beta_a^{(t)} - \beta_a^{(t-1)}\big).$$

To relate the two, we use several properties maintained by the algorithm: we only allocate the item if the discounted gain is non-negative (i.e., $w_{t,a} \geq \beta_a^{(t-1)}$) and our threshold is a combination of the largest weights with exponential coefficients. We can then upper bound the change in thresholds $\beta_a^{(t)} - \beta_a^{(t-1)}$ (Lemma 6) using only the weights of the new item $w_{t,a}$ and the disposed item $\min_{i\in S_a^{(t-1)}} w_{i,a}$, with appropriate coefficients. Setting c_a appropriately makes the two coefficients equal and gives us the desired comparison. This agrees with the intuition that c_a describes exactly how much additional gain we require from new items in order to account for the potential loss through the disposal, which is expressed in the dual potential. This definition results in the desired value of Q_a as stated in Equation 4.

Setting parameters d_a. It remains to choose the parameters d_a to optimize the approximation guarantee. In the large budget case, we can approximate $(1 + d_a/n_a)^{n_a} \approx \exp(d_a)$ which does not depend on the budget. Thus we can use the same parameter d for all parts and set it to the value that maximizes the approximation guarantee. In order to account for all budgets, including very small ones, we analyze the error incurred from approximating $(1 + d_a/n_a)^{n_a}$ by $\exp(d_a)$ (Lemma 7) and derive appropriate choices d_a tailored to the budgets n_a. As a result, we can handle the challenging setting where budgets can be very different, and obtain approximations that improve with the budget.

3.2 Non-Monotone Objectives

The maximization of general (possibly non-monotone) k-submodular functions is challenging even in the offline setting, where prior work does not achieve constant-factor approximation guarantees (Xiao et al. 2022). In this section we show how to use our framework along with fundamentally new techniques leveraging special properties of k-submodular functions to obtain the first constant factor approximation guarantees for general objectives, even in the more difficult online and streaming setting. The problem is different from the monotone k-submodular case as adding items with positive marginal gain may result in a decreased objective at the end. However, the core difficulty lies in the difference to non-monotone submodular maximization ($k = 1$) as approaches for this setting, such as subsampling and twice-greedy, do not have counterparts for non-monotone k-submodular maximization. This difference becomes apparent when comparing a solution created by our algorithm to the optimum solution: Even if the support of both solutions is identical, our solution can still have a lower objective due to misallocating items to the wrong parts.

Due to this difference, our approach is to consider the case $k = 1$ and $k \geq 2$ separately. For $k = 1$, we use our algorithm for a partition matroid constraint (Section 3.3) and thus consider only the case $k \geq 2$ here. A key insight is that if the maximum budget $\max_a n_a$ is not too large compared to the total budget $\sum_a n_a$, we can use pairwise monotonicity in a delicate adaptation of Algorithm 1. We thus consider the regime when the maximum budget is not too large first. From this, we then derive an algorithm for all budgets.

Algorithm for $\max_a n_a \leq \frac{1}{2}\sum_a n_a$. A serious complication of Algorithm 1 for non-monotone objectives it that we can no longer bound the difference in function value after re-allocating item t according to the optimum solution. To hedge against the loss in objective due to misallocation, we also need to take the thresholds of all other parts into account (for more details, we refer the reader to the proof of Lemma 9 in the appendix), so we make a modification to the allocation: In each iteration t, we choose the part that maximizes the following modified discounted gain:

$$a \leftarrow \arg\max_{a\in[k]} \left\{ \Delta_{t,a} f(\mathbf{S}^{(t-1)}) - \beta_a^{(t-1)} - \min_{a'\neq a} \beta_{a'}^{(t-1)} \right\}.$$

The full pseudocode and analysis can be found in Appendix B.1. We obtain:

Theorem 2. *When setting the parameters $\{d_a\}_{a\in[k]}$ as in Theorem 1, the modified algorithm achieves an approximation guarantee that is $\frac{1}{2}$ of the approximation in Theorem 1.*

Algorithm for All Budgets. If $\max_a n_a > \frac{1}{2}\sum_a n_a$, we can still obtain a constant-factor approximation (in expectation). Note that we either extract a lot of value from the part with maximum budget, or we can decrease the maximum budget and still obtain a good fraction of the original value. We mimic this idea by creating two solutions. For the first solution, we only allocate to the part with maximum budget while not exceeding the respective budget constraint. For the second solution, we solve the original problem, but reduce the budget of the maximum advertiser such that we

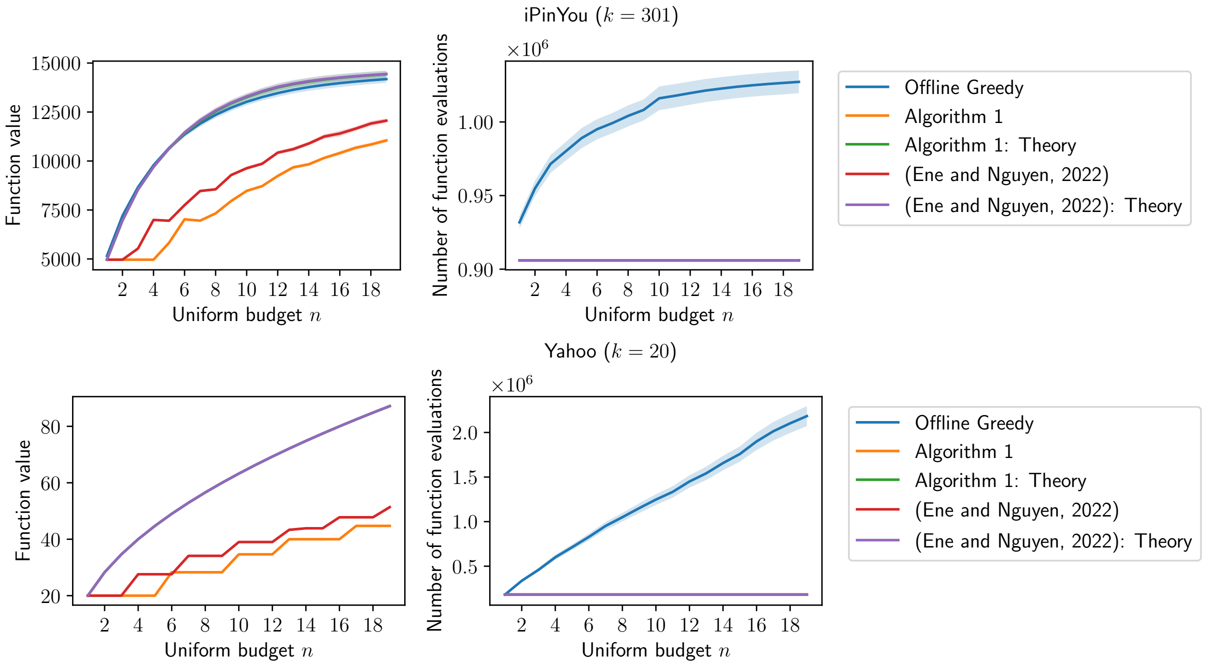

Figure 1: Ad allocation on the iPinYou (two plots to the left) and Yahoo instance (two plots to the right). We report mean and standard deviation over all days in the datasets, while varying a uniform budget $n_a = n$ for all $a \in [k]$. Note that the online algorithms using modified parameter choices coincide with offline greedy on the Yahoo instance. Also, all algorithms except offline greedy use the same number of function evaluations. We indicate runs with the theoretical parameters (e.g. "Algorithm 1: Theory" is Algorithm 1 using the theoretically optimal parameters).

can again apply Theorem 2. We select the better of the two solutions. This is only a streaming algorithm as we create multiple solutions, but we can also obtain an online algorithm by choosing a solution randomly. We defer a full description and analysis of this algorithm to Appendix B.2.

3.3 Additional Results

Our techniques can be applied in a unified way to obtain further results in the following settings.

Submodular maximization with a partition matroid (Appendix C). We close the gap between discrete and continuous methods which evaluate the multilinear extension and are therefore inefficient in practice (cf. Table 4). As Feldman, Karbasi, and Kazemi (2018), we subsample items to obtain a discrete algorithm, but improve upon their approximation guarantees by carefully choosing the subsampling probability in coordination with the coefficients $g_a(i)$. Their algorithm also has the previously best approximation guarantees for $k = 1$, and we even improve upon that.

k-Submodular maximization with knapsack constraints (Appendix D.2). We extend our approach using the density $\rho_{t,a} = \Delta_{t,a} f(\mathbf{S})/u_{t,a}$ in place of the marginal gain $\Delta_{t,a} f(\mathbf{S})$. An immediate problem of having items of any size is that we are not able to fill up each constraint exactly. To obtain an efficient discrete algorithm using optimal space, we maintain an infeasible solution $\tilde{\mathbf{S}}$ that overflows by at most a single item per part. Our update rule for β_a becomes a careful generalization that accounts for irregular item sizes and depends on $\tilde{\mathbf{S}}$.

Common Constraints (Appendix D.3). Recall that for a common constraint, we have only a single constraint on the support, e.g. $|S_1 \cup \cdots \cup S_k| \leq n$ or a single knapsack constraint. Our techniques also apply to this setting and we obtain improved online algorithms.

4 Experiments

We show the practicality of our algorithms for k-submodular maximization on instances for ad allocation, influence maximization, and max-cut, exemplifying the applications mentioned in the introduction. Further results are in Appendix E. We demonstrate the performance of our algorithms, which almost close the gap to offline greedy algorithms in function value while retaining the efficiency of online algorithms such as the ones due to Ene and Nguyen (2022).

Instances. Here, we briefly discuss our experiments. We defer a detailed description to Appendix E.

Ad Allocation. We consider the problem of allocating ad impressions to k advertisers (Mehta 2013). Ad impressions $t \in V$ arrive online and must be allocated immediately to advertisers $a \in [k]$ with budget constraints $|S_a| \leq n_a$. We use data from the iPinYou ad exchange (Zhang, Yuan, and Wang 2014) and a Yahoo dataset (Yahoo 2011). Our objective (total advertiser satisfaction) is a specific k-submodular function which we detail in Appendix E. We also create an imbalanced instance on the iPinYou data by sampling advertiser budgets n_a uniformly from $[10]$, with results in Table 3.

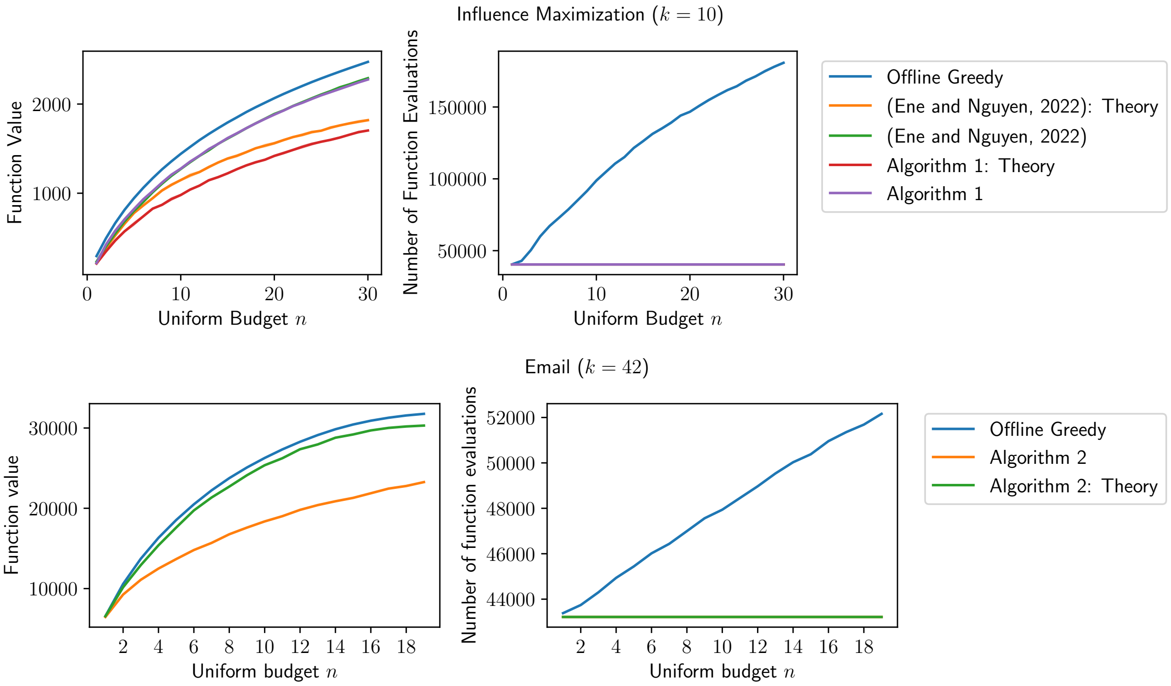

Figure 2: Influence maximization with k topics (top) and max-k-cut on the Email instance (bottom). We vary a uniform budget $n_a = n$ for $a \in [k]$ and report mean and standard deviation over 5 runs.

Algorithm	Theory	Modified
Algorithm 1	7499.13 ± 68.22	10236.05 ± 220.22
(Ene and Nguyen 2022)	5698.33 ± 88.57	9681.85 ± 152.87
Offline Greedy	10427.58 ± 214.04	

Table 3: Ad allocation on the iPinYou instance with imbalanced budgets. We report mean and standard deviation over 7 days. We use theoretical and modified parameter choices.

Influence Maximization with k Topics and Sensor Placement with k Measurements. We use the same experimental setup as Ene and Nguyen (2022) to create instances for monotone k-submodular maximization. The results for influence maximization and sensor placement are in Figure 2 and Figure 3 of Appendix E, respectively.

Max-k-Cut: The max-k-cut problem asks, given a graph $G = (V, E)$ and cardinality constraints $n_1, \ldots, n_k$ to find $\mathbf{S} \in (k+1)^V$ maximizing the total cut size defined as $f(\mathbf{S}) := \sum_{a \in [k]} \delta_G(S_a)$ where $\delta_G(S) := |\{\{u, v\} \in E : u \in S, v \notin S\}|$. Note that f is a general k-submodular function since we use a single submodular and symmetric function for each part δ_G. Our results on the Email network (Leskovec and Krevl 2014) are in Figure 2.

Algorithms. We use Algorithm 1 for the monotone instance ad allocation and Algorithm 2 for the general instance max-k-cut. We use two parameter choices for the online algorithms: First, we set $\{d_a\}_{a \in [k]}, \{c_a\}_{a \in [k]}$ to the optimal theoretical choice as the minimizer of Q_a in Lemma 5. Second, we modify these parameters by reducing each c_a to $\frac{1}{4}$ of the the previous choice to make the algorithms less conservative. The problems we consider are NP-complete, so instead of reporting the approximation ratio, we compare our algorithms with the greedy algorithms of Ohsaka and Yoshida (2015a) for monotone and Xiao et al. (2022) for general objectives. Both greedy algorithms work in the *offline* setting and access elements in arbitrary order, as opposed to the more constrained streaming and online setting we consider. We implement both using lazy evaluations. We also run the algorithm of Ene and Nguyen (2022) on monotone instances. The theoretical and modified parameter choices coincide with the ones used in their experiments. Details about the machine we used are in Appendix E.

Discussion. Our algorithms with modified parameter choices are able to match or even outperform the objective value of offline greedy on the ad-allocation instances in Figure 1, with significantly less function evaluations. This is surprising as greedy is an offline algorithm, therefore less restricted than our streaming algorithms and thus expected to perform better. The instances on influence maximization and max-k-cut (Figure 2) are more challenging, but we still almost match the objective value of offline greedy with about 8% less in objective. In the settings where the algorithms of Ene and Nguyen (2022) apply (i.e. the monotone instances in Figures 1 and 2 (top)), we recover their strong practical results. Table 3 shows that our algorithm also performs well if budgets are imbalanced and comes close to offline greedy. It outperforms the algorithm of Ene and Nguyen (2022) that does not adapt to the individual budgets in each part.

Acknowledgments

FS and AE were supported in part by NSF CAREER grant CCF-1750333, NSF grant III-1908510, and an Alfred P. Sloan Research Fellowship. HN was supported in part by NSF grant CCF-2311649.

References

Badanidiyuru, A.; Mirzasoleiman, B.; Karbasi, A.; and Krause, A. 2014. Streaming submodular maximization: massive data summarization on the fly. In *KDD*, 671–680. ACM.

Ene, A.; and Nguyen, H. L. 2022. Streaming Algorithm for Monotone k-Submodular Maximization with Cardinality Constraints. In *ICML*, volume 162 of *Proceedings of Machine Learning Research*, 5944–5967. PMLR.

Feldman, J.; Korula, N.; Mirrokni, V. S.; Muthukrishnan, S.; and Pál, M. 2009. Online Ad Assignment with Free Disposal. In *WINE*, volume 5929 of *Lecture Notes in Computer Science*, 374–385. Springer.

Feldman, M.; Karbasi, A.; and Kazemi, E. 2018. Do Less, Get More: Streaming Submodular Maximization with Subsampling. In *NeurIPS*, 730–740.

Feldman, M.; Liu, P.; Norouzi-Fard, A.; Svensson, O.; and Zenklusen, R. 2022. Streaming Submodular Maximization Under Matroid Constraints. In *ICALP*, volume 229 of *LIPIcs*, 59:1–59:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Feldman, M.; Norouzi-Fard, A.; Svensson, O.; and Zenklusen, R. 2021. Streaming Submodular Maximization with Matroid and Matching Constraints. *arXiv preprint arXiv:2107.07183*.

Gomes, R.; and Krause, A. 2010. Budgeted Nonparametric Learning from Data Streams. In *ICML*, 391–398. Omnipress.

Ha, D. T. K.; Pham, C. V.; and Tran, T. D. 2024. Improved approximation algorithms for k-submodular maximization under a knapsack constraint. *Comput. Oper. Res.*, 161: 106452.

Lavastida, T.; Moseley, B.; Ravi, R.; and Xu, C. 2021. Using Predicted Weights for Ad Delivery. In *ACDA*, 21–31. SIAM.

Leskovec, J.; and Krevl, A. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.

Lin, H.; and Bilmes, J. A. 2011. A Class of Submodular Functions for Document Summarization. In *ACL*, 510–520. The Association for Computer Linguistics.

Mehta, A. 2013. Online Matching and Ad Allocation. *Found. Trends Theor. Comput. Sci.*, 8(4): 265–368.

Mirzasoleiman, B.; Badanidiyuru, A.; and Karbasi, A. 2016. Fast Constrained Submodular Maximization: Personalized Data Summarization. In *ICML*, volume 48 of *JMLR Workshop and Conference Proceedings*, 1358–1367. JMLR.org.

Nguyen, L.; and Thai, M. T. 2020a. Streaming k-Submodular Maximization under Noise subject to Size Constraint. In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, 7338–7347. PMLR.

Nguyen, L.; and Thai, M. T. 2020b. Streaming K-submodular maximization under noise subject to size constraint. In *International Conference on Machine Learning (ICML)*, 7338–7347. PMLR.

Ohsaka, N.; and Yoshida, Y. 2015a. Monotone k-Submodular Function Maximization with Size Constraints. In *NIPS*, 694–702.

Ohsaka, N.; and Yoshida, Y. 2015b. Monotone k-Submodular Function Maximization with Size Constraints. In *Neural Information Processing Systems NeurIPS*, 694–702.

Pham, C. V.; Ha, D. K. T.; Hoang, H. X.; and Tran, T. D. 2022. Fast Streaming Algorithms for k-Submodular Maximization under a Knapsack Constraint. In *DSAA*, 1–10. IEEE.

Sakaue, S. 2017. On maximizing a monotone k-submodular function subject to a matroid constraint. *Discret. Optim.*, 23: 105–113.

Spaeh, F.; and Ene, A. 2023. Online Ad Allocation with Predictions. In *Advances in Neural Information Processing Systems*, volume 36, 17265–17295.

Tang, Z.; Wang, C.; and Chan, H. 2022. On maximizing a monotone *k*-submodular function under a knapsack constraint. *Oper. Res. Lett.*, 50(1): 28–31.

Wang, B.; and Zhou, H. 2021. Multilinear extension of k-submodular functions. *CoRR*, abs/2107.07103.

Ward, J.; and Zivný, S. 2016. Maximizing *k*-Submodular Functions and Beyond. *ACM Trans. Algorithms*, 12(4): 47:1–47:26.

Xiao, H.; Liu, Q.; Zhou, Y.; and Li, M. 2022. Non-monotone k-Submodular Function Maximization with Individual Size Constraints. In *CSoNet*, volume 13831 of *Lecture Notes in Computer Science*, 268–279. Springer.

Yahoo. 2011. Yahoo! Webscope.

Yu, K.; Li, M.; Zhou, Y.; and Liu, Q. 2023. On maximizing monotone or non-monotone k-submodular functions with the intersection of knapsack and matroid constraints. *J. Comb. Optim.*, 45(3): 93.

Zhang, W.; Yuan, S.; and Wang, J. 2014. Real-Time Bidding Benchmarking with iPinYou Dataset. *CoRR*, abs/1407.7073.