Non-monotone Sequential Submodular Maximization

Shaojie Tang[1], **Jing Yuan** [2]

[1]Naveen Jindal School of Management, University of Texas at Dallas
[2] Department of Computer Science and Engineering, University of North Texas
shaojie.tang@utdallas.edu, jing.yuan@unt.edu

Abstract

In this paper, we study a fundamental problem in submodular optimization known as sequential submodular maximization. The primary objective of this problem is to select and rank a sequence of items to optimize a group of submodular functions. The existing research on this problem has predominantly concentrated on the monotone setting, assuming that the submodular functions are non-decreasing. However, in various real-world scenarios, like diversity-aware recommendation systems, adding items to an existing set might negatively impact the overall utility. In response, we propose to study this problem with non-monotone submodular functions and develop approximation algorithms for both flexible and fixed length constraints, as well as a special case with identical utility functions. The empirical evaluations further validate the effectiveness of our proposed algorithms in the domain of video recommendations.

Introduction

Submodular optimization is a central problem in machine learning with various applications in a wide range of fields, including data summarization (Lin and Bilmes 2011), sparse reconstruction (Das and Kempe 2011), active learning (Golovin and Krause 2011; Tang and Yuan 2022), and viral marketing (Tang and Yuan 2020). These formulations aim to select a subset of items that maximizes a submodular function. However, in many real-world applications, the objective is not only to select items but also to rank them in a specific order (Azar and Gamzu 2011; Tschiatschek, Singla, and Krause 2017; Tang and Yuan 2021b). This motivates the study of *sequential submodular maximization* (Asadpour et al. 2022; Zhang, Tatti, and Gionis 2022), a fundamental problem in submodular optimization. This problem involves selecting and ranking a group of k items from a ground set V. The goal is to maximize the weighted summation of k submodular functions, denoted as $f_1, \cdots, f_k : 2^V \rightarrow \mathbb{R}^+$, where each function f_j takes the first j items from the ranking sequence as input. Formally, the objective is to find a feasible sequence $\pi = \{\pi_1, \cdots, \pi_k\}$ over items in V that maximizes the value of $F(\pi) \overset{\text{def}}{=} \sum_{j\in[k]} \lambda_j \cdot f_j(\pi_{[j]})$ where λ_j denotes the weight of function j and $\pi_{[j]} \overset{\text{def}}{=} \{\pi_1, \cdots, \pi_j\}$ denotes the first j items of π.

This problem, which captures the position-bias in item selection, has a wide range of applications, including sequential active learning and recommendation systems (Zhang, Tatti, and Gionis 2022). For instance, it can be applied to tackle challenges in ranking products on online retail platforms (Asadpour et al. 2022). Platforms like Amazon and Airbnb face the task of not only selecting a subset of products or rooms to showcase but also arranging them in a vertical list format to provide customers with an organized and customer-friendly browsing experience. Shoppers scroll through this list, depending on their level of patience, and may potentially make a purchase from the products displayed. The platform's objective is to optimize the selection and ranking of products to maximize the likelihood of a purchase. Interestingly, these applications can be framed as a problem of sequential submodular maximization. In this context, the parameters in $F(\pi)$ can be interpreted as follows: We denote the set of products as V, the window size of displayed products as k, λ_j represents the proportion of customers with a specific patience level j (e.g., a customer with patience level j is willing to view the first j products $\pi_{[j]}$). The function $f_j(\pi_{[j]})$ denotes the likelihood of purchase for customers with a patience level of j after seeing the first j products $\pi_{[j]}$. Typically, f_j is described as a submodular function. In this case, $F(\pi)$ captures the expected probability of purchase when a customer is shown a sequence of products π.

While previous research on sequential submodular maximization has primarily focused on the monotone setting, where submodular functions are assumed to be non-decreasing, the non-monotonicity of submodular functions becomes more apparent in many real-world scenarios, including feature selection (Das and Kempe 2008), maximum cut (Gotovos, Karbasi, and Krause 2015), profit maximization (Tang and Yuan 2021a), and data summarization (Mirzasoleiman, Badanidiyuru, and Karbasi 2016). One such example involves designing a *diversity-aware* recommendation system for a vast assortment of products spanning different categories (Lin and Bilmes 2010; Mirzasoleiman, Badanidiyuru, and Karbasi 2016; Amanatidis et al. 2020; Carbonell and Goldstein 1998). The system's primary goal is to generate a sequence of products that not only have high ratings but also effectively represent the entire collection. To address this tradeoff, a commonly adopted objective function is submodular but not monotone. More details about this application can be found in the experiment section. This highlights the need to develop algorithms that can handle non-monotone submodular functions efficiently.

Here are the main contributions of this paper: 1. We are the first to explore the sequential submodular maximization problem with non-monotone submodular functions. We investigate two variants: one with a flexible length constraint, where the goal is to find a sequence of items with a length of at most k, and another with a fixed length constraint, where the aim is to find a sequence of items with an exact length of k. 2. For the flexible length constraint, we introduce efficient constant-factor algorithms. In the case of the fixed length constraint, we develop an algorithm with an approximation ratio dependent on the ratio k/n, where n is the size of the ground set. When all k utility functions are identical, we provide constant-factor approximation algorithms.

Additional Related Work. The traditional non-monotone submodular optimization, where the objective is to select a *set* of items to maximize a non-monotone submodular function, has been extensively studied in the literature (Gharan and Vondrák 2011; Buchbinder et al. 2014). The best-known result for this problem, subject to a cardinality constraint, is a 0.385-approximation algorithm (Buchbinder and Feldman 2019). It is crucial to emphasize that our work differs from this traditional setting. Instead of aiming to find a *set* of items where the order does not matter, our goal is to find a *sequence* of items to maximize a group of submodular functions. It will become evident later that their problem can be seen as a special case of our setting. The focus on sequences introduces a new dimension of complexity in comparison to the traditional set-based approaches. However, we draw inspiration from previous studies (Amanatidis et al. 2020; Tang 2021) and incorporate a sampling technique to tackle the challenges arising from non-monotonicity. While there are other studies that have explored position bias in submodular optimization (Tschiatschek, Singla, and Krause 2017; Alaei, Makhdoumi, and Malekian 2010), our problem formulation significantly differs from theirs.

Preliminaries and Problem Statement

Notations. We first introduce some useful notations. Throughout the remainder of this paper, we denote the set $\{1, 2, \ldots, m\}$ as $[m]$ for any positive integer m. Given a function f, we use $f(i \mid S)$ to denote the marginal utility of adding i to S, i.e., $f(i \mid S) \overset{\text{def}}{=} f(S \cup \{i\}) - f(S)$. We say a function f is submodular if and only if $f(i \mid X) \geq f(i \mid Y)$ for any two sets X and Y such that $X \subseteq Y$ and any item $i \notin Y$. Let $\pi = \{\pi_1, \cdots, \pi_k\}$ be a sequence of items, we define the operation $\pi \oplus i$ as the concatenation of i to π, that is, $\pi \oplus i \overset{\text{def}}{=} \{\pi_1, \cdots, \pi_k, i\}$.

Now we are ready to introduce our research problem. Given k non-monotone submodular functions $f_1, \cdots, f_k : 2^V \to \mathbb{R}^+$ and non-negative coefficients $\lambda_1, \cdots, \lambda_k$, the objective of the Non-Monotone Sequential Submodular Maximization (NSM) problem is to find a *feasible* sequence $\pi = \{\pi_1, \cdots, \pi_k\}$ over items in V that maximizes the value of $F(\pi)$. Here $F(\pi) \overset{\text{def}}{=} \sum_{j \in [k]} \lambda_j \cdot f_j(\pi_{[j]})$ where $\pi_{[j]} \overset{\text{def}}{=} \{\pi_1, \cdots, \pi_j\}$ denotes the first j items of π. In this paper, we adopt a non-standard notation, employing π to represent both a sequence of items and the set of items comprising this sequence. To simplify notation, define $\pi_j = \emptyset$ if $|\pi| < j$ where $|\pi|$ denotes the number of *non-empty* items contained in π.

In this paper, we consider two types of feasibility constraints. The first type, known as NSM with Flexible Length, imposes a constraint where feasible sequences can contain at most k items. The second type, known as NSM with Fixed Length, requires that all feasible sequences contain a fixed number k of items from V. Both problems have additional restrictions: the same item cannot appear multiple times in a feasible sequence, and there should be no empty slots between two items. These constraints ensure that each item is considered at most once and maintain the sequential nature of the sequence.

NSM with Flexible Length

A formal description of NSM with Flexible Length can be written as follows:

P.1 $\max_\pi F(\pi)$ subject to $|\pi| \leq k$.

We first provide a negative result by showing that it is impossible to find a 0.491-approximation algorithm for **P.1**. This result can be easily shown by setting $\lambda_1 = \lambda_2 = \cdots, = \lambda_{k-1} = 0$ and $\lambda_k = 1$ in **P.1**, thereby reducing the problem to maximizing a non-monotone submodular function f_k over a *set* of at most k items, which does not allow for a 0.491-approximation solution (Gharan and Vondrák 2011). Thus, we establish the following lemma.

Lemma 1 *It is impossible to achieve a* 0.491-*approximation for* ***P.1***.

Algorithm Design. Next, we present the design of our algorithm, referred to as Sampling-Greedy, which builds upon a simple greedy approach that selects items based on their marginal utility. However, since our utility function is non-monotone, employing a straightforward greedy strategy may result in suboptimal selections with low utility. To address this challenge, we draw inspiration from (Amanatidis et al. 2020) and introduce a sampling phase to the greedy algorithm, extending its guarantees to the non-monotone setting. We provide a detailed description of our algorithm below, which consists of two phases:

1. RANDOM SUBSET SELECTION: We begin by selecting a random subset, denoted as R, from the ground set V. Each item $i \in V$ is independently included in R with a probability $p \in [0, 1]$, where p is a parameter to be optimized later. This random subset serves as the initial pool of items for subsequent processing.

2. GREEDY ALGORITHM ON R: We run a greedy algorithm only on R. This algorithm operates iteratively, augmenting the current sequence by selecting an item that provides the greatest incremental utility. To be precise, consider a specific round of the greedy algorithm, let π, whose

Algorithm 1: Sampling-Greedy

```
1: $E = V, \pi = \emptyset, t = 1, Q = \{i \in E \mid \sum_{j\in\{1,\cdots,k\}} \lambda_j \cdot f_j(\{i\}) > 0\}$
2: while $t \leq k$ and $Q \neq \emptyset$ do
3:     consider $z = \arg\max_{i\in Q} \sum_{j\in\{t,\cdots,k\}} \lambda_j \cdot f_j(i \mid \pi)$
4:     $E = E \setminus \{z\}$
5:     let $\Phi_z \sim \mathrm{Bernoulli}(p)$
6:     if $\Phi_z = 1$ then
7:         $\pi = \pi \oplus z, t \leftarrow t + 1$
8:         $Q = \{i \in E \mid \sum_{j\in\{t,\cdots,k\}} \lambda_j \cdot f_j(i \mid \pi) > 0\}$
9:     end if
10: end while
11: return $\pi$
```

initial value is $\emptyset$, denote the current solution. Let $z \in \arg\max_{i\in R} [F(\pi \oplus i) - F(\pi)]$ denote the item that has the largest marginal utility on top of π. If $F(\pi \oplus z) - F(\pi) > 0$, then we append z to π (i.e., $\pi = \pi \oplus z$) and proceed to the next iteration. Note that $F(\pi \oplus i) - F(\pi) = \sum_{j\in\{|\pi|+1,\cdots,k\}} \lambda_j \cdot f_j(i \mid \pi)$, observing that appending i to π only affects the utility of those functions in $\{f_{|\pi|+1}, \cdots, f_k\}$. This construction process continues until one of two stopping criteria is met: either the sequence reaches a length of $\min\{k, |R|\}$, or the marginal utility of the remaining items becomes non-positive. This ensures that further additions would not contribute positively to the overall utility.

In order to facilitate analysis, we present an alternative approach to implementing Sampling-Greedy (a detailed description of this alternative is listed in Algorithm 1). Unlike the original implementation, where the entire set R is sampled at the start of the algorithm, our alternative defers this decision. Instead, we employ a coin toss, with a success probability of p, after *considering* each item. This coin toss determines whether the item should be added to the solution. It is straightforward to confirm that both versions of the algorithm produce same output distributions.

Performance Analysis. We next analyze the approximation ratio of Sampling-Greedy. As in Algorithm 1, we introduce a random variable $\Phi \in \{0,1\}^V$ to denote the outcome of the coin tosses, e.g., $\Phi_i \in \{0,1\}$ represents the coin toss of item i.

Let $\pi^* = \{\pi^*_1, \cdots, \pi^*_l\}$ denote the optimal solution of **P.1**, where $l \leq k$. In the context of a specific run of our algorithm, where the coin tosses Φ is realized and the corresponding sequence returned by our algorithm is denoted as $\pi(\Phi)$, we *partition* the optimal solution π^* into three sets as follows:

1. Set $O_{\text{cons.}}(\Phi)$: This set contains all items π^*_j in the optimal sequence π^* that have been considered by our algorithm before (including when) position j being filled, but were not picked due to the random coin flip Φ. That is,

$$O_{\text{cons.}}(\Phi) \overset{\text{def}}{=} \{\pi^*_j \in \pi^* \mid \pi^*_j \text{ was considered by Algorithm 1 during the selection of } \pi(\Phi)_{[j]} \text{ but } \pi^*_j \notin \pi(\Phi)_{[j]}\},$$

where $\pi(\Phi)_{[j]}$ denotes the first j items of $\pi(\Phi)$.

2. Set $O_{\text{not cons.}}(\Phi)$: This set consists of all items π^*_j in π^* that have not been considered during our algorithm before (including when) position j being filled. That is,

$$O_{\text{not cons.}}(\Phi) \overset{\text{def}}{=} \{\pi^*_j \in \pi^* \mid \pi^*_j \text{ was not considered by Algorithm 1 during the selection of } \pi(\Phi)_{[j]}\}.$$

3. Set $O_{\text{ovlpd.}}(\Phi) = \pi^* \setminus (O_{\text{not cons.}}(\Phi) \cup O_{\text{cons.}}(\Phi))$: This set contains those items π^*_j in π^* that have been added to $\pi(\Phi)$ before (including when) position j being filled. That is, $O_{\text{ovlpd.}}(\Phi) \overset{\text{def}}{=} \{\pi^*_j \in \pi^* \mid \pi^*_j \in \pi(\Phi)_{[j]}\}$.

Let's consider a specific example to illustrate the aforementioned three sets. Suppose we have an optimal sequence $\pi^* = \{a, b, c, d\}$. And let's assume that the sequence considered by our algorithm is $\{b, c, d, e, f, g\}$, and the final sequence picked by our algorithm, based on the coin tosses Φ, is $\pi(\Phi) = \{c, e, f, g\}$. For this example, $O_{\text{cons.}}(\Phi) = \{b, d\}$, this is because our algorithm considered b (resp. d) before filling position 2 (resp. 4), but did not pick b (resp. d); $O_{\text{not cons.}}(\Phi) = \{a\}$, this is because our algorithm did not consider a before filling position 1; $O_{\text{ovlpd.}}(\Phi) = \{c\}$, this is because our algorithm picked c before filling position 3.

By defining this partition of π^*, we can effectively analyze the influence of the coin tosses Φ (or, equivalently, the random subset R in the original implementation of our algorithm) on the resultant sequence and its connection to the optimal solution.

To simplify notation, we drop the random variable Φ from $\pi(\Phi)$, $O_{\text{cons.}}(\Phi)$, $O_{\text{not cons.}}(\Phi)$ and $O_{\text{ovlpd.}}(\Phi)$ if it is clear from the context.

Before analyzing the performance of our algorithm, we introduce some additional notations. Recall that we define $\pi_j = \emptyset$ if $j > |\pi|$. Given a random output π from our algorithm and an optimal solution π^*, we define $F(\pi \uplus \pi^*) = \sum_{j\in[k]} \lambda_j \cdot f_j(\pi_{[j]} \cup \pi^*_{[j]})$ as the utility of the "union" of π and π^*. Here, $\pi_{[j]}$ (resp. $\pi^*_{[j]}$) denotes all items from π (resp. π^*) that are placed up to position j. Intuitively, $\pi \uplus \pi^*$ can be interpreted as a virtual sequence where both π_j and π^*_j are added to position j for all $j \in [k]$. Similarly, we define $F(\pi \uplus \pi^*_{\text{ovlpd.}}) = \sum_{j\in[k]} \lambda_j \cdot f_j(\pi_{[j]} \cup (\pi^*_{[j]} \cap O_{\text{ovlpd.}}))$ as the utility of the union of π and $O_{\text{ovlpd.}} \subseteq \pi^*$. Furthermore, we define $F(\pi \uplus \pi^*_{\text{cons.}}) = \sum_{j\in[k]} \lambda_j \cdot f_j(\pi_{[j]} \cup (\pi^*_{[j]} \cap O_{\text{cons.}}))$ as the utility of the union of π and $O_{\text{cons.}} \subseteq \pi^*$. Finally, we define $F(\pi \uplus \pi^*_{\text{not cons.}}) = \sum_{j\in[k]} \lambda_j \cdot f_j(\pi_{[j]} \cup (\pi^*_{[j]} \cap O_{\text{not cons.}}))$ as the utility of the union of π and $O_{\text{not cons.}} \subseteq \pi^*$.

Throughout this section, all expectations are taken over the coin tosses Φ. We first provide two technical lemmas.

Lemma 2 *Let $p \in [0,1]$ denote the sampling probability of each item, π denote the output from our algorithm, and π^* denote the optimal solution, we have*

$$(2 + \frac{1}{p}) \cdot \mathbb{E}[F(\pi)] \geq \mathbb{E}[F(\pi \uplus \pi^*)]. \quad (1)$$

Proof: By the definition of $O_{\text{ovlpd.}}$, all items from $\pi^*_{[j]} \cap O_{\text{ovlpd.}}$ must appear in $\pi_{[j]}$ for all $j \in [k]$. Hence, $\pi_{[j]} \cup$

$(\pi^*_{[j]} \cap O_{\text{ovlpd.}}) = \pi_{[j]}$. It follows that for all $j \in [k]$,

$$\begin{aligned} f_j(\pi_{[j]} \cup (\pi^*_{[j]} \cap O_{\text{ovlpd.}}) \cup (\pi^*_{[j]} \cap O_{\text{not cons.}}) \\ \cup (\pi^*_{[j]} \cap O_{\text{cons.}})) \\ = f_j(\pi_{[j]} \cup (\pi^*_{[j]} \cap O_{\text{not cons.}}) \cup (\pi^*_{[j]} \cap O_{\text{cons.}})). \quad (2) \end{aligned}$$

Therefore, $F(\pi \uplus \pi^*) = \sum_{j\in[k]} \lambda_j \cdot f_j(\pi_{[j]} \cup \pi^*_{[j]}) = \sum_{j\in[k]} \lambda_j \cdot f_j(\pi_{[j]} \cup (\pi^*_{[j]} \cap O_{\text{ovlpd.}}) \cup (\pi^*_{[j]} \cap O_{\text{not cons.}}) \cup (\pi^*_{[j]} \cap O_{\text{cons.}})) = \sum_{j\in[k]} \lambda_j \cdot f_j(\pi_{[j]} \cup (\pi^*_{[j]} \cap O_{\text{not cons.}}) \cup (\pi^*_{[j]} \cap O_{\text{cons.}}))$ where the second equality is by the fact that $O_{\text{cons.}}$, $O_{\text{not cons.}}$ and $O_{\text{ovlpd.}}$ is a partition of π^*, and the third equality is by equality (2).

For simplicity, let $F(\pi \uplus \pi^*_{\text{not cons.}} \uplus \pi^*_{\text{cons.}}) = \sum_{j\in[k]} \lambda_j \cdot f_j(\pi_{[j]} \cup (\pi^*_{[j]} \cap O_{\text{not cons.}}) \cup (\pi^*_{[j]} \cap O_{\text{cons.}}))$, the above equality indicates that $F(\pi \uplus \pi^*) = F(\pi \uplus \pi^*_{\text{not cons.}} \uplus \pi^*_{\text{cons.}})$.

It follows that

$$\begin{aligned} \mathbb{E}[F(\pi \uplus \pi^*)] &= \mathbb{E}[F(\pi)] + \mathbb{E}[F(\pi \uplus \pi^*_{\text{not cons.}}) - F(\pi)] \\ &\quad + \mathbb{E}[F(\pi \uplus \pi^*) - F(\pi \uplus \pi^*_{\text{not cons.}})] \\ &= \mathbb{E}[F(\pi)] + \mathbb{E}[F(\pi \uplus \pi^*_{\text{not cons.}}) - F(\pi)] \\ &\quad + \mathbb{E}[F(\pi \uplus \pi^*_{\text{not cons.}} \uplus \pi^*_{\text{cons.}}) - F(\pi \uplus \pi^*_{\text{not cons.}})] \end{aligned} \quad (3)$$

where the second equality is by the observation that $F(\pi \uplus \pi^*) = F(\pi \uplus \pi^*_{\text{not cons.}} \uplus \pi^*_{\text{cons.}})$.

Observe that

$$\begin{aligned} & F(\pi \uplus \pi^*_{\text{not cons.}} \uplus \pi^*_{\text{cons.}}) - F(\pi \uplus \pi^*_{\text{not cons.}}) \\ &= \sum_{j\in[k]} \lambda_j \cdot f_j(\pi_{[j]} \cup (\pi^*_{[j]} \cap O_{\text{not cons.}}) \cup (\pi^*_{[j]} \cap O_{\text{cons.}})) \\ &\quad - \sum_{j\in[k]} \lambda_j \cdot f_j(\pi_{[j]} \cup (\pi^*_{[j]} \cap O_{\text{not cons.}})) \\ &\le \sum_{j\in[k]} \lambda_j \cdot f_j(\pi_{[j]} \cup (\pi^*_{[j]} \cap O_{\text{cons.}})) - \sum_{j\in[k]} \lambda_j \cdot f_j(\pi_{[j]}) \\ &= F(\pi \uplus \pi^*_{\text{cons.}}) - F(\pi) \end{aligned}$$

where the inequality is by the assumption that f_j is submodular for all $j \in [k]$, and the observations that $\pi_{[j]} \subseteq \pi_{[j]} \cup (\pi^*_{[j]} \cap O_{\text{not cons.}})$, and $\pi^*_{[j]} \cap O_{\text{cons.}}$ does not overlap with $\pi_{[j]} \cup (\pi^*_{[j]} \cap O_{\text{not cons.}})$. Here $\pi^*_{[j]} \cap O_{\text{cons.}}$ does not overlap with $\pi_{[j]} \cup (\pi^*_{[j]} \cap O_{\text{not cons.}})$ is because $O_{\text{cons.}} \cap O_{\text{not cons.}} = \emptyset$ (this is by the definitions of these two sets) and $O_{\text{cons.}} \cap \pi = \emptyset$ (this is due to the fact that any item that is considered but not picked by our algorithm must not appear in π, noting that each item can be considered only once).

This, together with (3), implies that

$$\begin{aligned} \mathbb{E}[F(\pi \uplus \pi^*)] \le \mathbb{E}[F(\pi)] + \mathbb{E}[F(\pi \uplus \pi^*_{\text{not cons.}}) - F(\pi)] \\ + \mathbb{E}[F(\pi \uplus \pi^*_{\text{cons.}}) - F(\pi)]. \end{aligned} \quad (4)$$

To prove this lemma, it suffices to show that $p \cdot \mathbb{E}[F(\pi \uplus \pi^*_{\text{cons.}}) - F(\pi)] \le \mathbb{E}[F(\pi)]$ and

$$\mathbb{E}[F(\pi \uplus \pi^*_{\text{not cons.}}) - F(\pi)] \le \mathbb{E}[F(\pi)]. \quad (5)$$

The proofs for these two inequalities are provided in the technical report (Tang and Yuan 2023). □

Lemma 3 *Let $p \in [0, 1]$ denote the sampling probability of each item, π denote the output from our algorithm, and π^* denote the optimal solution, we have $\mathbb{E}[F(\pi \uplus \pi^*)] \ge (1-p) \cdot F(\pi^*)$.*

Proof: We begin by presenting a result that links random sampling to submodular maximization.

Lemma 4 *(Lemma 2.2 of (Buchbinder et al. 2014)). Consider a submodular set function $f : 2^V \to \mathbb{R}$. Let X be a subset of V, and let $X(p)$ denote a sampled subset obtained by including each item of X independently with a probability of at most p (not necessarily independent). The expected value of $f(X(p))$ is at least $(1-p)$ times the value of $f(\emptyset)$. In other words, $\mathbb{E}[f(X(p))] \ge (1-p)f(\emptyset)$.*

We define $h_j : 2^V \to \mathbb{R}$ as follows: $h_j(T) = f_j(T \cup \pi^*_{[j]})$. It can be easily verified that h_j is a submodular function, and $h_j(\emptyset) = f_j(\pi^*_{[j]})$. By applying the above lemma to the function h_j and considering that the items in $\pi_{[j]}$ are chosen with a probability of at most p, we can conclude that: $\mathbb{E}[f_j(\pi_{[j]} \cup \pi^*_{[j]})] = \mathbb{E}[h_j(\pi_{[j]})] \ge (1-p) \cdot h_j(\emptyset) = (1-p) \cdot f_j(\pi^*_{[j]})$ for all $j \in [k]$.

The following chain proves this lemma: $\mathbb{E}[F(\pi \uplus \pi^*)] = \mathbb{E}[\sum_{j\in[k]} \lambda_j \cdot f_j(\pi_{[j]} \cup \pi^*_{[j]})] = \sum_{j\in[k]} \lambda_j \cdot \mathbb{E}[f_j(\pi_{[j]} \cup \pi^*_{[j]})] \ge \sum_{j\in[k]} \lambda_j \cdot (1-p) \cdot f_j(\pi^*_{[j]}) = (1-p) \cdot \sum_{j\in[k]} \lambda_j \cdot f_j(\pi^*_{[j]}) = (1-p) \cdot F(\pi^*)$. □

Lemmas 2 and 3 imply the following main theorem.

Theorem 1 *Let $p \in [0, 1]$ be the sampling probability of each item, π be the output from our algorithm, and π^* denote the optimal solution, we have $\mathbb{E}[F(\pi)] \ge \frac{p(1-p)}{2p+1} \cdot F(\pi^*)$.*

Corollary 1 *Let $p = \frac{\sqrt{3}-1}{2}$, we have $\mathbb{E}[F(\pi)] \ge 0.134 \cdot F(\pi^*)$.*

Remark 1: For the case when all utility functions f_j are monotone and submodular, Algorithm 1, by setting $p = 1$, can achieve an improved approximation ratio of $1/2$. This is because if $p = 1$, then $O_{\text{cons.}} = \emptyset$, observing that when $p = 1$, all considered items must be added to the solution. It follows that inequality (4) is reduced to $\mathbb{E}[F(\pi \uplus \pi^*)] \le \mathbb{E}[F(\pi)] + \mathbb{E}[F(\pi \uplus \pi^*_{\text{not cons.}}) - F(\pi)]$. This can be rewritten as $\mathbb{E}[F(\pi \uplus \pi^*)] \le \mathbb{E}[F(\pi \uplus \pi^*_{\text{not cons.}})]$. This, together with inequality (5), implies that $\mathbb{E}[F(\pi)] \ge \mathbb{E}[F(\pi \uplus \pi^*)]/2$. Moreover, if f_j are monotone, it is easy to verify that $\mathbb{E}[F(\pi \uplus \pi^*)] \ge F(\pi^*)$. It follows that $\mathbb{E}[F(\pi)] \ge \mathbb{E}[F(\pi \uplus \pi^*)]/2 \ge F(\pi^*)/2$.

Remark 2: When all utility functions are homogeneous (i.e., $\exists f : f_j = f$ for all $j \in [k]$), Algorithm 1 becomes independent of the specific values of λ_j. This is because in Line 3 of Algorithm 1, finding z that maximizes $\sum_{j=t}^{k} \lambda_j \cdot f_j(i \mid \pi)$ is equivalent to maximizing $f(i \mid \pi)$, assuming $f_j = f$ for all $j \in [k]$. Thus, the selection of z becomes independent of λ_j, showcasing the algorithm's robustness against the knowledge of λ_j. In the context of recommendation systems, where λ_j represents the distribution of customers' patience levels, which may only be estimated approximately or remain unknown, our algorithm consistently delivers high-quality solutions.

NSM with Fixed Length

We next study the case with fixed length constraints. A formal description of NSM with Fixed Length can be written as follows:

P.2 $\max_\pi F(\pi)$ subject to $|\pi| = k$.

By setting $\lambda_1 = \lambda_2 = \cdots , = \lambda_{k-1} = 0$ and $\lambda_k = 1$ in **P.2**, our problem reduces to maximizing a non-monotone submodular function f_k over a *set* of exactly k items, which does not admit a 0.491-approximation solution (Gharan and Vondrák 2011). Hence, we establish the following lemma.

Lemma 5 *It is impossible to achieve a* 0.491-*approximation for* ***P.2***.

Approximation Algorithms for P.2

The basic idea of our algorithm is to first apply Algorithm 1 to calculate a sequence π with a maximum size of k, then, if required, we supplement π with additional backup items to ensure that it reaches a size of exactly k.

We provide a detailed description of our algorithm below:

1. Apply Algorithm 1 to calculate a sequence π with a maximum size of k.
2. If $|\pi| = k$, then return π as the final output. Otherwise, randomly select $k-|\pi|$ items B from the set $V\backslash\pi$, and append them to π in an arbitrary order. The resulting modified set π is then returned as the final output π^{p2}.

Note that problem **P.1** is a relaxed problem of **P.2**, observing that any feasible solution of **P.2** is also a feasible solution of **P.1**. As a consequence, we have the following lemma.

Lemma 6 *Let π^* and π^o denote the optimal solution of* ***P.1*** *and* ***P.2*** *respectively, we have $F(\pi^*) \geq F(\pi^o)$.*

Now we are ready to present the performance bound of our algorithm. The proof of this theorem is moved to the technical report (Tang and Yuan 2023).

Theorem 2 *Let π^{p2} denote the solution returned from our algorithm, we have $\mathbb{E}_{\Phi,B}[F(\pi^{p2})] \geq (1-\frac{k}{n})\cdot\frac{p(1-p)}{2p+1}\cdot F(\pi^o)$ where the expectation is taken over the random coin tosses Φ (phase 1) and the random backup set B (phase 2).*

If we set $p = \frac{\sqrt{3}-1}{2}$ in the first phase, we have the following corollary.

Corollary 2 *Let $p = \frac{\sqrt{3}-1}{2}$, we have $\mathbb{E}_{\Phi,B}[F(\pi^{p2})] \geq (1-\frac{k}{n})\cdot 0.134\cdot F(\pi^o)$.*

The above corollary implies that when the size constraint k remains relatively small compared to the overall number of items n, our algorithm is capable of achieving a good approximation of the optimal result. This observation is particularly relevant in scenarios such as recommendation systems where k is often much smaller than n.

Constant-Factor Approximation Algorithm for Homogeneous Functions

Next, we will examine an important special case of **P.2**, where we assume that all utility functions are homogeneous. This means that there exists a submodular function $f : 2^V \rightarrow \mathbb{R}^+$ such that $f_j = f$ for all $j \in [k]$. A formal description of this special case is listed in below.

P.2 $\max_\pi \sum_{j\in[k]} \lambda_j \cdot f(\pi_{[j]})$ subject to $|\pi| = k$.

Next, we present constant-factor approximation algorithms for this special case, which represents an improvement over the general case where we can only achieve an approximation ratio based on k/n.

We consider two cases: when $k < n/2$ and when $k \geq n/2$. The case when $k < n/2$ is straightforward, as one can directly apply our algorithm developed for the general case (as presented in the previous section) to achieve a constant-factor approximation. This is because when $k < n/2$, we have $1-k/n > 1/2$. This, together with Corollary 2, implies the following corollary.

Corollary 3 *Assume $k < n/2$, let $p = \frac{\sqrt{3}-1}{2}$, we have $\mathbb{E}_{\Phi,B}[F(\pi^{p2})] \geq \frac{0.134}{2}\cdot F(\pi^o)$ where π^o is the optimal solution of* ***P.2***.

The rest of this section focuses on the case when $k \geq n/2$. For the sake of simplicity, let's assume that n is an even number. Considering the optimal solution π^o, we can partition its utility $F(\pi^o)$ into two parts: $F(\pi^o) = \sum_{j\in[\frac{n}{2}]} \lambda_j \cdot f(\pi^o_{[j]}) + \sum_{j\in\{\frac{n}{2}+1,\cdots,k\}} \lambda_j \cdot f(\pi^o_{[j]})$. Here, the first part represents the utility obtained from the first $n/2$ functions, while the second part represents the utility obtained from the remaining $k - n/2$ functions. We can further divide the analysis into two subcases based on the relationship between the utilities from these two parts. Intuitively, if the first part dominates the overall utility, it suffices to find a sequence that maximizes the utility from the first part. Otherwise, if the second part contributes significantly to the overall utility, our focus is on finding a solution maximizes the second part. Although we initially have no information about the relationship between the two parts of the utility, given that π^o is unknown, we can make a guess regarding this relation and consider both possibilities. We can evaluate the performance of each case and choose the one that yields the best overall utility as the final output.

The Case When $\sum_{j\in[\frac{n}{2}]} \lambda_j \cdot f(\pi^o_{[j]}) \geq \sum_{j\in\{\frac{n}{2}+1,\cdots,k\}} \lambda_j \cdot f(\pi^o_{[j]})$ We first study the case when $\sum_{j\in[\frac{n}{2}]} \lambda_j \cdot f(\pi^o_{[j]}) \geq \sum_{j\in\{\frac{n}{2}+1,\cdots,k\}} \lambda_j \cdot f(\pi^o_{[j]})$, that is, the first $n/2$ functions contributes more than half of the total utility. In this scenario, our objective is to find a sequence that maximizes the utility from the first part (a formal description of this problem is listed in **P.2a**).

P.2a $\max_\pi \sum_{j\in[\frac{n}{2}]} \lambda_j \cdot f(\pi_{[j]})$ subject to $|\pi| = k$.

Let π^a denote the optimal solution of **P.2a**, we have

$$\sum_{j\in[\frac{n}{2}]} \lambda_j \cdot f(\pi^a_{[j]}) \geq \sum_{j\in[\frac{n}{2}]} \lambda_j \cdot f(\pi^o_{[j]}) \geq \frac{1}{2}\cdot F(\pi^o) \quad (6)$$

where the first inequality is because π^o is a feasible solution of **P.2a** and the assumption that π^a is an optimal solution of

P.2a, and the second inequality is by the fact that $F(\pi^o) = \sum_{j\in[\frac{n}{2}]} \lambda_j \cdot f(\pi^o_{[j]}) + \sum_{j\in\{\frac{n}{2}+1,\cdots,k\}} \lambda_j \cdot f(\pi^o_{[j]})$ and the assumption that $\sum_{j\in[\frac{n}{2}]} \lambda_j \cdot f(\pi^o_{[j]}) \geq \sum_{j\in\{\frac{n}{2}+1,\cdots,k\}} \lambda_j \cdot f(\pi^o_{[j]})$.

Now we are ready to present our algorithm. It is easy to verify that if we replace the constraint $|\pi| = k$ in **P.2a** by $|\pi| = \frac{n}{2}$, it does not affect the value of the optimal solution. This is because those items placed after position $n/2$ do not affect the utility of the first $n/2$ functions anyway. A formal description of this variant is listed in **P.2b**.

P.2b $\max_\pi \sum_{j\in[\frac{n}{2}]} \lambda_j \cdot f(\pi_{[j]})$ subject to $|\pi| = \frac{n}{2}$.

Let π^b denote the optimal solution of **P.2b**, we have

$$\sum_{j\in[\frac{n}{2}]} \lambda_j \cdot f(\pi^b_{[j]}) = \sum_{j\in[\frac{n}{2}]} \lambda_j \cdot f(\pi^a_{[j]}) \geq \frac{1}{2} \cdot F(\pi^o) \qquad (7)$$

where the equality is because, as previously discussed, **P.2a** and **P.2b** share the same value of the optimal solution, and the inequality is by inequality (6).

We can utilize our algorithm designed for the general case to solve **P.2b** and derive a solution denoted as π. Considering that the size constraint in **P.2b** is $\frac{n}{2}$, we can substitute $\frac{n}{2}$ into Corollary 2 and conclude that π provides a $\frac{0.134}{2}$-approximation solution for **P.2b**, that is,

$$\sum_{j\in[\frac{n}{2}]} \lambda_j \cdot f(\pi_{[j]}) \geq \frac{0.134}{2} \cdot \sum_{j\in[\frac{n}{2}]} \lambda_j \cdot f(\pi^b_{[j]}). \qquad (8)$$

However, π might not be a feasible solution for the original problem **P.2a**, as its size might be less than k. To ensure feasibility, we can append an arbitrary set of $k - \frac{n}{2}$ items from $V \setminus \pi$ to π to obtain the final solution π'. This step ensures that the solution satisfies the size constraint and makes it feasible for **P.2a**. More importantly, this step does not affect the utility of π, given that any items placed after position $n/2$ do not affect the utility of the first $n/2$ functions. As a result, we achieve a $\frac{0.134}{4}$-approximation for **P.2**, that is, $F(\pi') \geq \sum_{j\in[\frac{n}{2}]} \lambda_j \cdot f(\pi'_{[j]}) = \sum_{j\in[\frac{n}{2}]} \lambda_j \cdot f(\pi_{[j]}) \geq \frac{0.134}{2} \cdot \sum_{j\in[\frac{n}{2}]} \lambda_j \cdot f(\pi^b_{[j]}) \geq \frac{0.134}{4} \cdot F(\pi^o)$ where the equality is because, as previously discussed, any items placed after position $n/2$ do not affect the utility of the first $n/2$ functions, the second inequality by inequality (8) and the third inequality is by inequality (7). This leads to the following lemma.

Lemma 7 *There exists a $\frac{0.134}{4}$-approximation solution for P.2, assuming $\sum_{j\in[\frac{n}{2}]} \lambda_j \cdot f(\pi^o_{[j]}) \geq \sum_{j\in\{\frac{n}{2}+1,\cdots,k\}} \lambda_j \cdot f(\pi^o_{[j]})$.*

The Case When $\sum_{j\in[\frac{n}{2}]} \lambda_j \cdot f(\pi^o_{[j]}) < \sum_{j\in\{\frac{n}{2}+1,\cdots,k\}} \lambda_j \cdot f(\pi^o_{[j]})$ For this case, we manage to design an algorithm that achieves an approximation ratio of $\frac{0.134}{4}$. We move this part to the technical report (Tang and Yuan 2023).

Putting It All Together Recall that we develop $\frac{0.134}{4}$-approximation algorithms for the case when $\sum_{j\in[\frac{n}{2}]} \lambda_j \cdot f(\pi^o_{[j]}) \geq \sum_{j\in\{\frac{n}{2}+1,\cdots,k\}} \lambda_j \cdot f(\pi^o_{[j]})$ and $\sum_{j\in[\frac{n}{2}]} \lambda_j \cdot f(\pi^o_{[j]}) < \sum_{j\in\{\frac{n}{2}+1,\cdots,k\}} \lambda_j \cdot f(\pi^o_{[j]})$ respectively. Although we lack prior knowledge of the optimal solution, selecting the superior solution between the aforementioned options guarantees achieving an approximation ratio of $\frac{0.134}{4}$.

Experimental Evaluation

We conduct experiments on real-world datasets to evaluate the impact of user type distributions in the context of video recommendation. Suppose we are dealing with a large video library, denoted as V, containing videos spanning multiple categories, each categorized as potentially overlapping subsets, namely $C_1, C_2, \ldots, C_m \subseteq V$. When a user provides a set of category preferences, our platform's objective is to generate a sequence of videos, denoted as π, from those specified categories that maximizes the expected user engagement $\sum_{j\in[k]} \lambda_j \cdot f_j(\pi_{[j]})$. Recall that k denotes the maximum window size of displayed videos, and λ_j represents the proportion of users with a specific patience level j who are willing to view the first j videos $\pi_{[j]}$. Denote by $\mathcal{U}$ the space of user types, each user type $u \in \mathcal{U}$ is specified by a pair $(j, f_j(\cdot))$. The user considers the first j videos in the list and obtains utility $f_j(\pi_{[j]})$. The platform lacks precise knowledge of the user's exact type but is aware of the type distribution $\mathcal{D}$. We consider a common characterization of $f_j(\cdot)$ as a submodular function. Each video s has a rating ρ_s and we denote by $w_{st} \in [0, 1]$ some measure of the percentage of similarity between any two videos s and t. In introducing our function $f_j(\cdot)$, we adopt the approach outlined in (Amanatidis et al. 2020). We start by considering the auxiliary objective $g_j(\pi_{[j]}) = \sum_{s\in\pi_{[j]}} \sum_{t\in V} w_{st} - \eta \sum_{s\in\pi_{[j]}} \sum_{t\in\pi_{[j]}} w_{st}$, for some $\eta \geq 1$ (Mirzasoleiman, Badanidiyuru, and Karbasi 2016). This objective takes inspiration from maximal marginal relevance (Carbonell and Goldstein 1998), emphasizing coverage while penalizing similarity. To balance between highly-rated videos and those representing the entire collection, we employ the submodular function $f_j(\pi_{[j]}) = \alpha \sum_{s\in\pi_{[j]}} \rho_s + \beta g_j(\pi_{[j]})$ for $\alpha, \beta \geq 0$.

Datasets. We evaluate our algorithms and benchmarks on the latest MovieLens dataset (Harper and Konstan 2015), consisting of $62,423$ movies, of which $13,816$ have both user-generated tags and ratings. Similarities, represented as w_{st}, are computed from these tags using pairwise minimum tag vectors with more details forthcoming.

Algorithms and Parameters. We compare our proposed sampling-based greedy algorithms (labeled as SG) against two baselines, namely, COVDIV and QUALITY, under both flexible length and fixed length settings. COVDIV iteratively selects an item with the largest marginal utility in $g_j(\pi_{[j]})$, i.e., the marginal relevance inspired objective, until no more items with positive marginal utility can be found. COVDIV returns a sequence of items ranked in the same order as they are selected. QUALITY is a simple ranking method that orders individual items in non-increasing quality. Here quality can be measured by average ratings or scores pre-

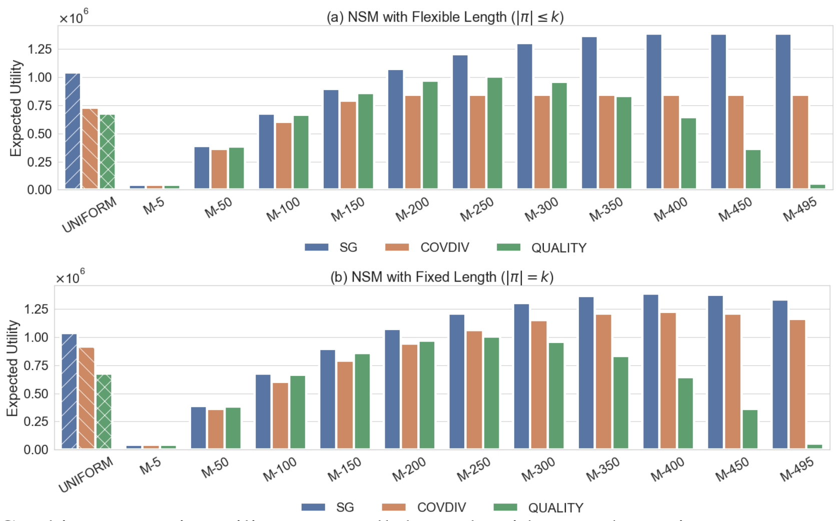

Figure 1: SG achieves superior utility among all three algorithms under various user type distributions.

dicted by the recommender systems. Due to space limitation and the results being similar for these metrics, we report the results with average rating metric for QUALITY. For user type distribution $\mathcal{D}$, we use the normal mass function that approximates the normal distribution $\mathcal{N}(\mu, \sigma^2)$ to set the values for $\lambda_j, j \in [k]$. We explore the impact of user type distribution on the performance of the algorithms by varying the value of μ and σ. Each movie, denoted as i, is linked to a tag vector $t^i \in [0, 1]^{1128}$. Within this vector, each component represents the relevance score for an individual tag. We employ a widely accepted model to quantify the similarity w_{ij} between two videos, i and j, defined as $w_{ij} = \sqrt{\sum_{l=1}^{1128}(\min\{t_l^i, t_l^j\})^2}$ (Amanatidis et al. 2020). This metric calculates the L2 norm of the element-wise minimum between t^i and t^j. We set $\eta = 35$ and adjust the parameters α and β to ensure that the two components in $f_j(\cdot)$ are roughly equal in magnitude. In each experimental set, we perform 100 rounds and present the average results as follows.

Experimental Results. We measure the performance of the algorithms in terms of their expected utility with respect to various user type distributions. As shown in Figure 1, we test under a uniform user type distribution where $\lambda_j = 1/k, j \in [k]$, labeled as UNIFORM on the x-axis. We also report the results under the approximated normal distribution with varying mean, μ, labeled as M-μ on the x-axis. In order to distinguish these two types of distributions, we add hatches on the bars for the uniform distribution. In our experiments, we set $k = 500$. Figure 1(a) and (b) show the results for NSM with flexible length, i.e., $|\pi| \leq k$, and that for NSM with fixed length, i.e., $|\pi| = k$, respectively. It shows in Figure 1(a) that SG outperforms the benchmarks under all tested user type distributions. While the benchmarks yield an expected utility of 7.29×10^5 and 6.75×10^5 respectively under the uniform distribution, SG yields an expected utility over 1.04×10^6, a 43% increase. Under the approximated normal distribution, the increase of μ indicates a higher number of videos viewed by average users, leading to an increase in the expected utility for the algorithms. QUALITY always returns a sequence of size k as adding more videos always increases the sum of ratings. However, our objective function is non-monotone, similar videos added by QUALITY result in a lower expected utility as μ further increases. SG and COVDIV only add items with positive marginal utility. In our experiments, SG returns a sequence of around 400 videos, and COVDIV returns around 200 videos. As μ further increases, their overall expected utility converge since most users will view all the videos listed. We observe that SG outperforms both benchmarks under all test settings. While QUALITY solely considers the ratings of the items, COVDIV only considers their capability of representing the whole collection. SG shows a superior balance between the two. This result validates the superiority of our proposed algorithm over the benchmarks. Figure 1(b) illustrates the results for NSM with fixed length, which means all three algorithms return a sequence of 500 videos. The results for QUALITY remain the same. We observe that under the uniform distribution, SG yields a lower expected utility while COVDIV yields a higher one, compared with the case of flexible length. We also observe that under approximated normal distribution, the expected utility of SG starts to decrease as μ goes over 400, due to the negative contribution from the lastly added videos. The expected utility of COVDIV peaks at $\mu = 250$, and then declines. The reason is that for $200 < \mu < 250$, the increase in the total rating is enough to compensate for the negative contribution from $g_j(\cdot)$, which does not hold any more as μ further increases. Notably, SG outperforms the others, underscoring our method's advantage.

Acknowledgements

This work was supported in part by CAHSI-Google institutional Research Program.

References

Alaei, S.; Makhdoumi, A.; and Malekian, A. 2010. Maximizing sequence-submodular functions and its application to online advertising. *arXiv preprint arXiv:1009.4153*.

Amanatidis, G.; Fusco, F.; Lazos, P.; Leonardi, S.; and Reiffenhäuser, R. 2020. Fast Adaptive Non-Monotone Submodular Maximization Subject to a Knapsack Constraint. In *Advances in neural information processing systems*.

Asadpour, A.; Niazadeh, R.; Saberi, A.; and Shameli, A. 2022. Sequential Submodular Maximization and Applications to Ranking an Assortment of Products. *Operations Research*.

Azar, Y.; and Gamzu, I. 2011. Ranking with submodular valuations. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, 1070–1079. SIAM.

Buchbinder, N.; and Feldman, M. 2019. Constrained submodular maximization via a nonsymmetric technique. *Mathematics of Operations Research*, 44(3): 988–1005.

Buchbinder, N.; Feldman, M.; Naor, J.; and Schwartz, R. 2014. Submodular maximization with cardinality constraints. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, 1433–1452. SIAM.

Carbonell, J.; and Goldstein, J. 1998. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, 335–336.

Das, A.; and Kempe, D. 2008. Algorithms for subset selection in linear regression. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, 45–54.

Das, A.; and Kempe, D. 2011. Submodular meets spectral: greedy algorithms for subset selection, sparse approximation and dictionary selection. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, 1057–1064.

Gharan, S. O.; and Vondrák, J. 2011. Submodular maximization by simulated annealing. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, 1098–1116. SIAM.

Golovin, D.; and Krause, A. 2011. Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *Journal of Artificial Intelligence Research*, 42: 427–486.

Gotovos, A.; Karbasi, A.; and Krause, A. 2015. Non-monotone adaptive submodular maximization. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

Harper, F. M.; and Konstan, J. A. 2015. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4): 1–19.

Lin, H.; and Bilmes, J. 2010. Multi-document summarization via budgeted maximization of submodular functions. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 912–920.

Lin, H.; and Bilmes, J. 2011. A class of submodular functions for document summarization. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, 510–520.

Mirzasoleiman, B.; Badanidiyuru, A.; and Karbasi, A. 2016. Fast Constrained Submodular Maximization: Personalized Data Summarization. In *ICML*, 1358–1367.

Tang, S. 2021. Beyond pointwise submodularity: Non-monotone adaptive submodular maximization in linear time. *Theoretical Computer Science*, 850: 249–261.

Tang, S.; and Yuan, J. 2020. Influence maximization with partial feedback. *Operations Research Letters*, 48(1): 24–28.

Tang, S.; and Yuan, J. 2021a. Adaptive Regularized Submodular Maximization. In *32nd International Symposium on Algorithms and Computation (ISAAC 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

Tang, S.; and Yuan, J. 2021b. Cascade Submodular Maximization: Question Selection and Sequencing in Online Personality Quiz. *Production and Operations Management*, 30(7): 2143–2161.

Tang, S.; and Yuan, J. 2022. Optimal Sampling Gaps for Adaptive Submodular Maximization. In *AAAI*.

Tang, S.; and Yuan, J. 2023. Non-monotone Sequential Submodular Maximization. *arXiv preprint arXiv:2308.08641*.

Tschiatschek, S.; Singla, A.; and Krause, A. 2017. Selecting sequences of items via submodular maximization. In *Thirty-First AAAI Conference on Artificial Intelligence*.

Zhang, G.; Tatti, N.; and Gionis, A. 2022. Ranking with submodular functions on a budget. *Data mining and knowledge discovery*, 36(3): 1197–1218.