Dynamic Non-monotone Submodular Maximization

Kiarash Banihashem[*†]
kiarash@umd.edu
University of Maryland

Leyla Biabani[*‡]
l.biabani@tue.nl
TU Eindhoven

Samira Goudarzi[*†]
samirag@umd.edu
University of Maryland

MohammadTaghi Hajiaghayi[*†]
hajiagha@cs.umd.edu
University of Maryland

Peyman Jabbarzade[*†]
peymanj@umd.edu
University of Maryland

Morteza Monemizadeh[*‡]
m.monemizadeh@tue.nl
TU Eindhoven

Abstract

Maximizing submodular functions has been increasingly used in many applications of machine learning, such as data summarization, recommendation systems, and feature selection. Moreover, there has been a growing interest in both submodular maximization and dynamic algorithms. In 2020, Monemizadeh [46] and Lattanzi, Mitrovic, Norouzi-Fard, Tarnawski, and Zadimoghaddam [39] initiated developing dynamic algorithms for the monotone submodular maximization problem under the cardinality constraint k. In 2022, Chen and Peng [14] studied the complexity of this problem and raised an important open question: "*Can we extend [fully dynamic] results (algorithm or hardness) to non-monotone submodular maximization?*". We affirmatively answer their question by demonstrating a reduction from maximizing a non-monotone submodular function under the cardinality constraint k to maximizing a monotone submodular function under the same constraint. Through this reduction, we obtain the first dynamic algorithms solving the non-monotone submodular maximization problem under the cardinality constraint k. We've derived two algorithms, both maintaining an $(8+\epsilon)$-approximate of the solution. The first algorithm requires $\mathcal{O}(\epsilon^{-3}k^3\log^3(n)\log(k))$ oracle queries per update, while the second one requires $\mathcal{O}(\epsilon^{-1}k^2\log^3(k))$. Furthermore, we showcase the benefits of our dynamic algorithm for video summarization and max-cut problems on several real-world data sets.

1 Introduction

Submodular functions are powerful tools for solving real-world problems as they provide a theoretical framework for modeling the famous "*diminishing returns*" [29] phenomenon arising in a variety of practical settings. Many theoretical problems such as those involving graph cuts, entropy-based clustering, coverage functions, and mutual information can be cast in the submodular maximization framework. As a result, submodular functions have been increasingly used in many applications of machine learning such as data summarization [52, 51, 50], feature selection [16, 18, 17, 37], and recommendation systems [23]. These applications include both the monotone and non-monotone versions of the maximization of submodular functions.

Applications of non-monotone submodular maximization. The general problem of non-monotone submodular maximization has been studied extensively in [26, 11, 10, 43, 4, 47]. This

[*] equal contribution

[†] Department of Computer Science, University of Maryland, College Park, MD, USA.

[‡] Department of Mathematics and Computer Science, Eindhoven University of Technology, the Netherlands.

Preprint.

problem has numerous applications in video summarization, movie recommendation [43], and revenue maximization in viral marketing [34][4]. An important application of this problem appears in maximizing the difference between a monotone submodular function and a linear function that penalizes the addition of more elements to the set (e.g., the coverage and diversity trade-off). An illustrative example of this application is the maximum facility location in which we want to open a subset of facilities and maximize the total profit from served clients plus the cost of facilities we did not open [20]. Another important application occurs when expressing learning problems such as feature selection using weakly submodular functions [16, 37, 24, 49].

Our contribution. In this paper, we consider the non-monotone submodular maximization problem under cardinality constraint k in the *fully dynamic setting*. In this model, we have a *universal ground set* V. At any time t, ground set $V_t \subseteq V$ is the set of elements that are inserted but not deleted after their last insertion till time t. More formally, we assume that there is a sequence of "updates" such that each update either inserts an element to V_{t-1} or deletes an element from V_{t-1} to form V_t.

We assume that there is a (non-monotone) submodular function f that is defined over the universal ground set V. Our goal is to maintain, at each point in time, a set of size at most k whose submodular value is maximum among any subset of V_t of size at most k.

Since calculating such a set is known to be NP-hard [26] even in the offline setting (where you get all the items at the same time), we focus on providing algorithms with provable approximation guarantees, while maintaining fast update time. This is challenging as elements may be inserted or deleted, possibly in an adversarial order. While several dynamic algorithms exist for monotone submodular maximization, non-monotone submodular maximization is a considerably more challenging problem as adding elements to a set may decrease its value.

In STOC 2022, Chen and Peng [14] raised the following open question:

Open problem: "Can we extend [fully dynamic] results (algorithm or hardness) to non-monotone submodular maximization?"

In this paper, we answer their question affirmatively by providing the first dynamic algorithms for non-monotone submodular maximization.

To emphasize the significance of our result, it should be considered that although monotone submodular maximization under cardinality constraint has a tight $\frac{e}{e-1}$ approximation algorithm in the offline mode and nearly tight $(2+e)$ approximation algorithms for both streaming and dynamic settings, there is a hardness result for the non-monotone version stating that it is impossible to obtain a 2.04 (i.e., 0.491) approximation algorithm for this problem even in the offline setting[30], and to the best of our knowledge, the current state of the art algorithms for this problem have 2.6 and 3.6 (i.e., 0.385 and 0.2779) approximation guarantees for offline [9] and streaming settings [1], respectively.

We obtain our result, by proposing a general reduction from the problem of dynamically maintaining a non-monotone submodular function under cardinality constraint k to developing a dynamic thresholding algorithm for maximizing monotone submodular functions under the same constraint. We first define τ-thresholding dynamic algorithms that we use in our reduction.

Definition 1.1 (τ-Thresholding Dynamic Algorithm)**.** Let $\tau > 0$ be a parameter. We say a dynamic algorithm is τ-thresholding if at any time t of sequence Ξ, it reports a set $S_t \subseteq V_t$ of size at most k such that

- **Property 1:** either **a)** S_t has k elements and $f(S_t) \geq k\tau$, or **b)** S_t has less than k elements and for any $v \in V_t \setminus S_t$, the marginal gain $\Delta(v|S_t) < \tau$.
- **Property 2:** The number of elements changed in any update, i.e, $|S_{t+1} \backslash S_t| + |S_t \backslash S_{t+1}|$, is not more than the number of queries made by the algorithm during the update.

In the above definition, the first property reflects the main intuition of threshold-based algorithms, while the last property is a technical condition required in our analysis. It's worth noting that the

[4]The problem of selecting a subset of people in a social network to maximize their influence in a viral marketing campaign can be modeled as a constrained submodular maximization problem. When we introduce a cost, then the influence minus the cost is modeled as non-monotone submodular maximization problems [8, 7, 33].

thresholding technique has been used widely for optimizing submodular functions [46, 41, 25, 39, 15]. We next state our main result, which is a general reduction.

Theorem 1.2 (Reduction Metatheorem). *Suppose that $f : 2^V \to \mathbb{R}_{\geq 0}$ is a (possibly non-monotone) submodular function defined on subsets of a ground set V and let $k \in \mathbb{N}$ be a parameter.*

Assume that for any given value of τ, there exists a τ-thresholding dynamic algorithm with an expected (amortized) $O(g(n,k))$ oracle queries per update. Then, there exist the following dynamic algorithms:

- *A dynamic algorithm with an approximation guarantee of $(8+\varepsilon)$ using an expected (amortized) $O(k + \min(k, g(n,k)) \cdot g(n,k) \cdot \varepsilon^{-1} \log(k))$ oracle queries per update.*
- *A dynamic algorithm maintaining a $(10+\varepsilon)$-approximate solution of the optimal value of f using an expected (amortized) $O(\min(k, g(n,k)) \cdot g(n,k) \cdot \varepsilon^{-1} \log(k))$ oracle calls per update.*

In [46], Monemizadeh developed a dynamic algorithm for monotone submodular maximization under cardinality constraint k, which requires an amortized $O(\varepsilon^{-2} k^2 \log^3(n))$ number of oracle queries per update. Interestingly, in the appendix, we show that this algorithm is indeed τ-thresholding (for any given τ). Now, if we use this τ-thresholding dynamic algorithm inside our reduction Metatheorem 1.2, we obtain a dynamic algorithm that maintains a $(8+\varepsilon)$-approximate solution using an expected amortized $O(\varepsilon^{-3} k^3 \log^3(n) \log(k))$ oracle queries per update.

The recent paper [6] of Banihashem, Biabani, Goudarzi, Hajiaghayi, Jabbarzade, and Monemizadeh develops a new dynamic algorithm for monotone submodular maximization under cardinality constraint k, which uses an expected $O(\varepsilon^{-1} k \log^2(k))$ number of oracle queries per update. A similar proof shows that this new algorithm is τ-thresholding as well. We have provided its pseudocode and a detailed explanation on why this algorithm is indeed τ-thresholding in the appendix. By exploiting this algorithm in our Reduction Metatheorem 1.2, we can reduce the number of oracle queries mentioned to an expected number of $O(\varepsilon^{-2} k^2 \log^3(k))$ per update.

The second result in Theorem 1.2 is also of interest as it can be used to devise a dynamic algorithm for non-monotone submodular maximization with polylogarithmic query complexity if one can provide a τ-thresholding dynamic algorithm for maximizing monotone submodular functions (under the cardinality constraint k) with polylogarithmic query complexity.

1.1 Preliminaries

Submodular maximization. Let V be a ground set of elements. We say a function $f : 2^V \to \mathbb{R}_{\geq 0}$ is a *submodular* function if for any $A, B \subseteq V$, $f(A)+f(B) \geq f(A \cup B)+f(A \cap B)$. Equivalently, f is a submodular function if for any subsets $A \subseteq B \subseteq V$ and for any element $e \in V \setminus B$, it holds that $f(A \cup \{e\}) - f(A) \geq f(B \cup \{e\}) - f(B)$. We define $\Delta(e|A) := f(A \cup \{e\}) - f(A)$ the *marginal gain* of adding the element e to set A. Similarly, we define $\Delta(B|A) := f(A \cup B) - f(A)$ for any sets $A, B \subseteq V$. Function f is *monotone* if $f(A) \leq f(B)$ holds for any $A \subseteq B \subseteq V$, and it is *non-monotone* if it is not necessarily the case. In the submodular maximization problem under cardinality constraint k, we seek to compute a set S^* such that $|S^*| \leq k$ and $f(S^*) = \max_{|S| \leq k} f(S)$, where f is a submodular function and $k \in \mathbb{N}$ is a given parameter.

Query access model. Similar to recent dynamic works [39, 14], we assume the access to a submodular function f is given by an *oracle*. The oracle allows *set queries* such that for every subset $A \subseteq V$, one can query the value $f(A)$. In this query access model, the marginal gain $\Delta_f(e|A) \doteq f(A \cup \{e\}) - f(A)$ for every subset $A \subseteq V$ and an element $e \in V \setminus A$, can be computed using two set queries. To do so, we first query $f(A \cup \{e\})$ and then $f(A)$.

Dynamic model. Let Ξ be a sequence of inserts and deletes of an underlying universe V. We assume that $f : 2^V \to \mathbb{R}_{\geq 0}$ is a (possibly non-monotone) submodular function defined on subsets of the universe V. We define time t to be the t^{th} update (i.e., insertion or deletion) of sequence Ξ. We let Ξ_t be the sub-sequence of updates from the beginning of sequence Ξ till time t and denote by $V_t \subseteq V$ the set of elements that are inserted but not deleted from the beginning of the sequence Ξ till any time t. That is, V_t is the current ground set of elements. We let $OPT_t = \max_{S \subseteq V_t : |S| \leq k} f(S)$.

Query complexity. The *query complexity* of a dynamic α-approximate algorithm is the number of oracle queries that the algorithm must make to compute a solution S_t with respect to ground set V_t whose submodular value is an α-approximation of OPT_t. That is, $|S_t| \leq k$ and $f(S_t) \geq \alpha \cdot OPT_t$. Observe that the dynamic algorithm remembers every query it has made so far. Thus results of queries made in previous times may help find S_t in current time t.

Oblivious adversarial model. The dynamic algorithms that we develop in this paper are in the *oblivious adversarial model* as is common for analysis of randomized data structures such as universal hashing [12]. The model allows the adversary, who is aware of the submodular function f and the algorithm that is going to be used, to determine all the arrivals and departures of the elements in the ground set V. However, the adversary is unaware of the random bits used in the algorithm and so cannot choose updates adaptively in response to the randomly guided choices of the algorithm. Equivalently, we can suppose that the adversary prepares the full input (insertions and deletions) before the algorithm runs.

1.2 Related Work

Offline algorithms. The offline version of non-monotone submodular maximization was first studied by Feige, Mirrokni, and Vondrák in [26]. They studied *unconstrained non-monotone submodular maximization* and developed constant-factor approximation algorithms for this problem. In the offline query access model, they showed that a subset S chosen uniformly at random has a submodular value which is a 4-approximation of the optimal value for this problem. In addition, they also described two local search algorithms. The first uses f as the objective function, and provides 3-approximation and the second uses a noisy version of f as the objective function and achieves an improved approximation guarantee 2.5 for maximizing unconstrained non-monotone non-negative submodular functions. Interestingly, they showed $(2-\varepsilon)$-approximation for symmetric submodular functions would require an exponential number of queries for any fixed $\varepsilon > 0$.

Oveis Gharan and Vondrák [31] showed that an extension of the 2.5-approximation algorithm can be seen as *simulated annealing* method which provides an improved approximation of roughly 2.4. Later, Buchbinder, Feldman, Naor, and Schwartz [10] at FOCS'12, presented a randomized linear time algorithm achieving a tight approximation guarantee of 2 that matches the known hardness result of [26]. Bateni, Hajiaghayi, and Zadimoghaddam [8, 7] and Gupta, Roth, Schoenebeck, and Talwar [33] independently studied non-monotone submodular maximization subject to cardinality constraint k in the offline and secretary settings. In particular, Gupta *et al.* [33] obtained an offline 6.5-approximation for this problem.

All of the aforementioned approximation algorithms are offline, where the whole input is given in the beginning, whereas the need for real-time analysis of rapidly changing data streams motivates the study of this problem in settings such as the dynamic model that we study in this paper.

Streaming algorithms. The dynamic model that we study in this paper is closely related to the streaming model [2, 35]. However, the difference between these two models is that in the streaming model, we maintain a data structure using which we compute a solution at the end of the stream and so, the time to extract the solution is not important as we do it once. However, in the dynamic model, we need to maintain a solution after every update, thus, the update time of a dynamic algorithm should be as fast as possible.

The known streaming algorithms [44, 27, 28] work in the insertion-only streaming model and they do not support deletions as well as insertions. Indeed, there are streaming algorithms [36, 45] for the monotone submodular maximization problem that support deletions, but the space and the update time of these algorithms depend on the number of deletions which could be $\Omega(n)$, where $n = |V|$ is the size of ground set V.

For monotone submodular maximization, Badanidiyuru, Mirzasoleiman, Karbasi, and Krause [3] proposed an insertion-only streaming algorithm with a $(2+\varepsilon)$-approximation guarantee under a cardinality constraint k. Chekuri, Gupta, and Quanrud [13] presented (insertion-only) streaming algorithms for maximizing monotone and non-monotone submodular functions subject to p-

matchoid constraint[5]. Later, Mirzasoleiman, Jegelka, and Krause [44] and Feldman, Karbasi, and Kazemi [27] developed streaming algorithms with better approximation guarantees for maximizing a non-monotone function under a p-matchoid constraint. Currently, the best streaming algorithm for maximizing a non-monotone submodular function subject to a cardinality constraint is due to Alaluf, Ene, Feldman, Nguyen, and Suh [1] whose approximation guarantee is $3.6 + \varepsilon$, improving the 5.8-approximation guarantee that was proposed by Feldman *et al.* [27].

Dynamic algorithms. At NeurIPS 2020, Lattanzi, Mitrovic, Norouzi-Fard, Tarnawski, and Zadimoghaddam [39]

and Monemizadeh [46] initiated the study of submodular maximization in the dynamic model. They presented dynamic algorithms that maintain $(2 + \varepsilon)$-approximate solutions for maximizing a monotone submodular function subject to cardinality constraint k. Later, at STOC 2022, Chen and Peng [14] studied the complexity of this problem and they proved that developing a c-approximation dynamic algorithm for $c < 2$ is not possible unless we use a number of oracle queries polynomial in the size of ground set V. In 2023, Banihashem, Biabani, Goudarzi, Hajiaghayi, Jabbarzade, and Monemizadeh[5] developed an algorithm for monotone submodular maximization problem under cardinality constraint k using a polylogarithmic amortized update time. Concurrent works of Banihashem, Biabani, Goudarzi, Hajiaghayi, Jabbarzade, and Monemizadeh[6] and Duetting, Fusco, Lattanzi, Norouzi-Fard, and Zadimoghaddam[21] developed the first dynamic algorithms for monotone submodular maximization under a matroid constraint. Authors of [6] also improve the algorithm of [46] for monotone submodular maximization subject to cardinality constraint k. There are also studies on the dynamic model of influence maximization, which shares similarities with submodular maximization [48].

In this paper, for the first time, we study the generalized version of their problem by presenting an algorithm for maximizing the non-monotone submodular functions in the dynamic setting.

2 Dynamic algorithm

In this section, we explain the algorithm that we use in the reduction that we stated in Metatheorm 1.2. The pseudocode of our algorithm is given in Algorithm 1, Algorithm 2, and Algorithm 3.

Such reductions were previously proposed in the offline model by [33], and later works extended this idea to the streaming model [13, 44]. We develop a reduction in the dynamic model inspired by these works, though in our proof, we require a tighter analysis to obtain the approximation guarantee in our setting.

We consider an arbitrary time t of sequence Ξ where V_t is the set of elements inserted before time t, but not deleted after their last insertion. Let $OPT_t^* = \max_{S \subseteq V_t : |S| \leq k} f(S)$. For simplicity, we drop t from V_t and OPT_t^*, when it is clear from the context.

In the following, we assume that the value of OPT is known. Although the exact value of OPT^* is unknown, we can maintain parallel runs of our dynamic algorithm for different guesses of the optimal value. By using $(1 + \varepsilon')^i$, where $i \in \mathbb{Z}$ as our guesses for the optimal value, one of our guesses $(1+\varepsilon')$-approximates the value of OPT^*. We show that the output of our algorithm satisfies the approximation guarantee in the run whose OPT $(1+\varepsilon')$-approximates the value of OPT^*. Later, in the appendix, we show that it is enough to consider each element e only in runs i for which we have $\frac{\varepsilon'}{k} \cdot (1 + \varepsilon')^i \leq f(e) \leq (1 + \varepsilon')^i$. This method increases the query complexity of our dynamic algorithm by only a factor of $O(\varepsilon^{-1} \log k)$.

Our approach for solving the non-monotone submodular maximization is to first run the thresholding algorithm with input set V to find a set S_1 of at most k elements. Since f is non-monotone, subsets of S_1 might have a higher submodular value than $f(S_1)$. Then, we use an α-approximation algorithm (for $0 < \alpha \leq 1$) to choose a set $S_1' \subseteq S_1$ with guarantee $\mathbf{E}[f(S_1')] \geq \alpha \cdot \max_{C \subseteq S_1} f(C)$. Next, we run the thresholding algorithm with the input set $V \backslash S_1$ and compute a set S_2. At the end, we return set $S = \arg\max_{C \in \{S_1, S_1', S_2\}} f(C)$. Intuitively, for an optimal solution S^*, if $f(S_1 \cap S^*)$ is

[5]For non-monotone submodular maximization subject to cardinality constraint k, Chekuri, Gupta, and Quanrud [13] claimed that they obtained 4.7-approximation algorithm. However, Alaluf, Ene, Feldman, Nguyen, and Suh [1] found an error in the proof of this approximation guarantee.

a good approximation of OPT, then $f(S'_1)$ is a good approximation of OPT. On the other hand, if both $f(S_1)$ and $f(S_1 \cap S^*)$ are small with respect to OPT, then we can ignore the elements of S_1 and show that we can find a set $S_2 \subseteq V \setminus S_1$ of size at most k whose submodular value is a good approximation of OPT. The following lemma proves that the submodular value of S is a reliable approximation of the optimal solution. The formal proof of this lemma can be found in Section 2.

Lemma 2.1 (Approximation Guarantee)**.** *Assuming that $OPT^* \in [\frac{OPT}{1+\varepsilon'}, OPT]$, the expected submodular value of set S is $\mathbf{E}[f(S)] \geq (1 - O(\varepsilon'))\frac{OPT^*}{6+\frac{1}{\alpha}}$.*

Next, we explain the steps of our reduction in detail.

Let us first fix the threshold $\tau = \frac{OPT}{k(3+1/(2\alpha))}$. Then, we fix a τ-thresholding dynamic algorithm (for example, [46] or [6]) and suppose we denote it by DYNAMICTHRESHOLDING. Before sequence Ξ of updates starts, we create two independent instances $\mathcal{I}_1$ and $\mathcal{I}_2$ of DYNAMICTHRESHOLDING. The first instance will maintain set S_1 and the second instance will maintain set S_2. For instance $\mathcal{I}_i$ where $i \in \{1, 2\}$, we consider the following subroutines:

- INSERT$_{\mathcal{I}_i}(v)$: This subroutine inserts an element v to instance $\mathcal{I}_i$.
- DELETE$_{\mathcal{I}_i}(v)$: Invoking this subroutine will delete the element v from instance $\mathcal{I}_i$.
- EXTRACT$_{\mathcal{I}_i}$: This subroutine returns the maintained set (of size at most k) of $\mathcal{I}_i$.

Extracting S_1. After the update at time t, first, we would like to set $Z = S_1^- \cup \{v\}$ or $Z = S_1^- \setminus \{v\}$, if the update is the insertion of an element v or the deletion of an element v, respectively, where S_1^- is the set S_1 that instance $\mathcal{I}_1$ maintains just before this update. To find set S_1^-, we just need to invoke subroutine EXTRACT$_{\mathcal{I}_1}$. If the update is an insertion, we insert it into instance $\mathcal{I}_1$ using INSERT$_{\mathcal{I}_1}(v, \tau)$, and if the update is a deletion, we delete v from both $\mathcal{I}_1$ and $\mathcal{I}_2$ using DELETE$_{\mathcal{I}_1}(v)$ and DELETE$_{\mathcal{I}_2}(v)$. We then invoke EXTRACT$_{\mathcal{I}_1}$ once again to return set S_1.

Extracting S'_1. Buchbinder *et al.* [10] developed a method to extract a subset $S'_1 \subseteq S_1$ whose submodular value is a good approximation of $\max_{C \subseteq S_1} f(C)$. In this algorithm, we start with two solutions $\emptyset$ and S_1. The algorithm considers the elements (in arbitrary order) one at a time. For each element, it determines whether it should be added to the first solution or removed from the second solution. Thus, after a single pass over set S_1, both solutions completely coincide, which is the solution that the algorithm outputs. They show that a (deterministic) greedy choice in each step obtains 3-approximation of the best solution in S_1. However, if we combine this greedy choice with randomization, we can obtain a 2-approximate solution. Since we do a single pass over set S_1, the number of oracle queries is $O(|S_1|)$.

The second algorithm that we can use to extract S'_1 is a random sampling algorithm proposed by Feige *et al.* [26], which choose every element in S_1 with probability $1/2$. They show that this random sampling returns a set S'_1 whose approximation factor is $1/4$ of $\max_{C \subseteq S_1} f(C)$, and its number of oracle calls is $O(1)$. We denote either of these two methods by SUBSETSELECTION.

Extracting S_2. Next, we would like to update the set S_2 that is maintained by instance $\mathcal{I}_2$. To do this, for every element $u \in Z \setminus S_1$, we add it to $\mathcal{I}_2$ using INSERT$_{\mathcal{I}_2}(u, \tau)$, and for every element $u \in S_1 \setminus Z$, we delete it from $\mathcal{I}_2$ using DELETE$_{\mathcal{I}_2}(u, \tau)$. Finally, when $\mathcal{I}_2$ exactly includes all the current elements other than the ones in S_1, we call subroutine EXTRACT$_{\mathcal{I}_2}$ to return set S_2.

Corollary 2.2. *We obtain the $(8 + \varepsilon)$ approximation guaranty stated in the Metatheorem 1.2 by using the local search method for our* SUBSETSELECTION*, and we get the $(10 + \varepsilon)$ approximation guaranty by utilizing the random sampling method for our* SUBSETSELECTION *subroutine.*

Proof. These are immediate results of Lemma 2.1, and α being $\frac{1}{2}$ and $\frac{1}{4}$ in the local search method and random sampling method, respectively. □

Analysis. In this section, we prove the correctness of our algorithms and analyze the number of oracle queries of our algorithms, which finishes the proof of Theorems 1.2.

Consider an arbitrary time t. Let S_t be the reported set of DYNAMICTHRESHOLDING at time t. Recall that V_t is the ground set at time t, and we drop the t for simplicity, so we use V and S to

Algorithm 1 Initialization(k, OPT)

$\tau \leftarrow \frac{OPT}{k(3+1/(2\alpha))}$, where α is $\frac{1}{2}$ or $\frac{1}{4}$ based on the selection of algorithm for SUBSETSELECTION.
Instantiate two independent instances $\mathcal{I}_1$ and $\mathcal{I}_2$ of DYNAMICTHRESHOLDING for monotone submodular maximization under cardinality constraint k using τ

Algorithm 2 UPDATE(v)

$Z \leftarrow \text{EXTRACT}_{\mathcal{I}_1}$
if UPDATE(v) is an insertion **then**
    Invoke $\text{INSERT}_{\mathcal{I}_1}(v)$, $Z \leftarrow Z \cup \{v\}$
else
    Invoke $\text{DELETE}_{\mathcal{I}_1}(v)$, $\text{DELETE}_{\mathcal{I}_2}(v)$, $Z \leftarrow Z \setminus \{v\}$
$S_1 \leftarrow \text{EXTRACT}_{\mathcal{I}_1}$
$S_1' \leftarrow \text{SUBSETSELECTION}(S_1)$
for $u \in S_1 \backslash Z$ **do**
    $\text{DELETE}_{\mathcal{I}_2}(u)$
for $u \in Z \backslash S_1$ **do**
    $\text{INSERT}_{\mathcal{I}_2}(u)$
$S_2 \leftarrow \text{EXTRACT}_{\mathcal{I}_2}$
Return $\arg\max_{C \in \{S_1, S_1', S_2\}} f(C)$

denote V_t and S_t. We first present Lemma 2.3 whose proof is given in the appendix. Then we proceed to prove Lemma 2.1 and Theorem 1.2

Lemma 2.3. *Suppose that set S satisfies Property 1.b of Definition 1.1. It means that S has less than k elements and for any $v \in V \setminus S$, the marginal gain $\Delta(v|S) < \tau$. Then, for any arbitrary subset $C \subseteq V$, we have $f(S) \geq f(S \cup C) - |C| \cdot \tau$.*

Proof of Lemma 2.1: Assume that at a fixed time t, OPT^* and S^* are the optimal value and an optimal solution for the submodular maximization of function f under cardinality constraint k. This means that $|S^*| \leq k$ and $f(S^*) = OPT^*$. Recall that $\tau = \frac{OPT}{k(3+\frac{1}{2\alpha})}$, where OPT is our guess for the optimal value. Also, by assumption we have $OPT^* \in [\frac{OPT}{1+\epsilon'}, OPT]$, or equivalently $OPT \in [OPT^*, (1+\epsilon')OPT^*]$.

To prove the lemma, we claim that $\max(\mathbf{E}[f(S_1)], \mathbf{E}[f(S_1')], \mathbf{E}[f(S_2)]) \geq (1 - O(\varepsilon'))\frac{OPT^*}{6+\frac{1}{\alpha}}$.

Suppose this claim is true. Using Jensen's inequality [22], we have $\mathbf{E}[\max(f(S_1), f(S_1'), f(S_2))] \geq \max(\mathbf{E}[f(S_1)], \mathbf{E}[f(S_1')], \mathbf{E}[f(S_2)])$, which yields $\mathbf{E}[f(S)] \geq (1 - O(\varepsilon'))\frac{OPT^*}{6+1/\alpha}$.

To prove the claim, we consider two cases. The first case is when $f(S_1 \cap S^*) \geq \frac{\tau k}{2\alpha}$ and the second case is if $f(S_1 \cap S^*) < \frac{\tau k}{2\alpha}$.

Suppose the first case is true. Then, the subset selection algorithm (either random sampling method or local search) returns S_1' for which $\mathbf{E}[f(S_1')] \geq \alpha \cdot \max_{S \subseteq S_1} f(S)$. Since $S_1 \cap S^* \subseteq S_1$, we have

For the latter case, we show that $\mathbf{E}[f(S_1)] + \mathbf{E}[f(S_2)] \geq (1 - O(\varepsilon'))\frac{OPT^*}{3+1/2\alpha}$, inferring $\max(\mathbf{E}[f(S_1)], \mathbf{E}[f(S_2)]) \geq (1 - O(\varepsilon'))\frac{OPT^*}{6+1/\alpha}$. Indeed, since S_1 and S_2 are reported by an τ-thresholding algorithm, if $|S_1| = k$ or $|S_2| = k$, then $\max(\mathbf{E}[f(S_1)], \mathbf{E}[f(S_2)])$ is at least $\tau k = \frac{OPT}{3+1/2\alpha}$ by the first property of τ-thresholding algorithms.

Now suppose that $|S_1|, |S_2| < k$, which means that Property 1.b of Definition 1.1 holds for S_1 and S_2. Therefore, we have $f(S_1) \geq f(S_1 \cup S^*) - \tau|S^*|$ and $f(S_2) \geq f(S_2 \cup (S^* \setminus S_1)) - \tau|S^* \setminus S_1|$ by Lemma 2.3. Besides, we have $f(S_1 \cap S^*) - \frac{\tau k}{2\alpha} < 0$. Therefore,

$$f(S_1) + f(S_2) \geq f(S_1 \cup S^*) - \tau|S^*| + f(S_2 \cup (S^* \setminus S_1)) - \tau|S^* \setminus S_1| + f(S_1 \cap S^*) - \frac{\tau k}{2\alpha} \ .$$

Algorithm 3 SUBSETSELECTION(S)

function UNIFORMSUBSET(S)
  $T \leftarrow \emptyset$
  for $s \in S$ **do**
    if $Coin(\frac{1}{2})$ **then** ▷ With probability $\frac{1}{2}$
      $T \leftarrow T \cup \{s\}$
  return T
function LOCALSEARCHSUBSET(S)
  $X_0 \leftarrow \emptyset,\ Y_0 \leftarrow S.$
  for $i = 1$ to $|S|$ **do**
    $a_i \leftarrow f(X_{i-1} \cup \{s_i\}) - f(X_{i-1}),\ b_i \leftarrow f(Y_{i-1} \setminus \{s_i\}) - f(Y_{i-1})$
    $a'_i \leftarrow \max(a_i, 0),\ b'_i \leftarrow \max(b_i, 0)$
    if $a'_i = b'_i = 0$ **then** $a'_i/(a'_i + b'_i) = 0$
    with probability $a'_i/(a'_i + b'_i)$ **do:**
    $X_i \leftarrow X_{i-1} \cup \{s_i\},\ Y_i \leftarrow Y_{i-1}$
    else do: $X_i \leftarrow X_{i-1},\ Y_i \leftarrow Y_{i-1} \setminus \{s_i\}$
  return $X_{|S|}$ (or equivalently $Y_{|S|}$)

Since $|S^* \setminus S_1| \leq |S^*| \leq k$ we have

$$f(S_1) + f(S_2) \geq f(S_1 \cup S^*) + f(S_2 \cup (S^* \setminus S_1)) + f(S_1 \cap S^*) - (2 + 1/(2\alpha))\tau k \ .$$

Since $S_1 \cap S_2 = \emptyset$ and f is submodular, we have $f(S_1 \cup S^*) + f(S_2 \cup (S^* \setminus S_1)) \geq f(S_1 \cup S_2 \cup S^*) + f(S^* \setminus S_1)$. Additionally, by the submodularity and non-negativity of f, we have $f(S_1 \cap S^*) \geq f(S^*) - f(S^* \setminus S_1)$, because $f(S^* \setminus S_1) + f(S_1 \cap S^*) \geq f(S^*) + f(\emptyset)$. By adding the last two inequalities and using the non-negativity of f once again, we get $f(S_1 \cup S^*) + f(S_2 \cup (S^* \setminus S_1)) + f(S_1 \cap S^*) \geq f(S_1 \cup S_2 \cup S^*) + f(S^*) \geq f(S^*) = OPT^*$. By putting everything together we have,

$$f(S_1) + f(S_2) \geq OPT^* - (2 + 1/(2\alpha))\tau k = OPT^* - (\frac{4\alpha + 1}{2\alpha})(\frac{OPT(2\alpha)}{6\alpha + 1}).$$

By using the assumption that $OPT \leq (1 + \epsilon')OPT^*$, we have,

$$f(S_1)+f(S_2) \geq OPT^*(1-(\frac{(4\alpha + 1)(1 + \epsilon')}{6\alpha + 1})) \geq OPT^*(\frac{2\alpha - \epsilon'(4\alpha + 1)}{6\alpha + 1}) = (1-O(\varepsilon'))\frac{OPT^*}{3 + 1/(2\alpha)}.$$

□

Proof of Theorem 1.2:

As previously discussed, we've established in Lemma 2.1 and Corollary 2.2 that utilizing the local search method for the SUBSETSELECTION subroutine results in an approximation ratio of $(8 + \varepsilon)$, whereas the random sampling method achieves an approximation ratio of $(10 + \varepsilon)$. Thus, the only remaining aspect to address in proving this theorem is proving the query complexity of our proposed algorithm. In Lemma 2.4, we bound the number of queries made in each run of our algorithm per update, proving the bounds given in Theorem 1.2 by considering the extra $O(\varepsilon^{-1} \log k)$-factor caused by our parallel runs. □

Lemma 2.4. *Let the random variable Q_t denote the number of oracle calls that our algorithm in Theorem 1.2 makes at time t in each of the parallel runs. Depending on whether the expected or expected amortized number of oracle calls made by the thresholding algorithm* DYNAMICTHRESHOLD *per each update is $O(g(n, k))$, if we choose the local search method as our* SUBSETSELECTION *subroutine, we have*

$$\mathbf{E}[Q_t] \in O(\min(k \cdot g(n, k), g(n, k)^2)) + O(k) \ ,$$

or

$$\mathbf{E}[\sum_{t=1}^{T} Q_t] \in O(T \cdot \min(k \cdot g(n, k), g(n, k)^2)) + O(k) \ ,$$

and if we choose the random sampling method as our SUBSETSELECTION *subroutine, we have*

$$\mathbf{E}[Q_t] \in O(\min(k \cdot g(n,k), g(n,k)^2)) \ ,$$

or

$$\mathbf{E}[\sum_{t=1}^{T} Q_t] \in O(T \cdot \min(k \cdot g(n,k), g(n,k)^2))$$

Proof. Consider the case where the expected number of oracle calls made by the thresholding algorithm DYNAMICTHRESHOLD per each update is $O(g(n,k))$. Per each update, our algorithm makes an update in instance $\mathcal{I}_1$ causing $O(g(n,k))$ oracle queries. Next, we make either $O(k)$ or 0 oracle queries for the SUBSETSELECTION subroutine, depending on the used method. We also make a series of updates in instance $\mathcal{I}_2$, each causing $O(g(n,k))$ oracle queries. The number of such updates is bounded by the number of changes in the output of instance $\mathcal{I}_1$, which is bounded by both k and $O(g(n,k))$ (according to the second property of Definition 1.1). These comprise all the oracle queries made by our algorithm at time t. Therefore, the given bounds for this case hold. A detailed proof for the remaining bounds is provided in the appendix. □

3 Empirical results

In this section, we empirically study our $(8+\varepsilon)$-approximation dynamic algorithm. We implement our codes in C++ and run them on a MacBook laptop with 8 GB RAM and $M1$ processor. We empirically study the performance of our algorithm for video summarization and the Max-Cut problem.

Video summarization. Here, we use the Determinantal Point Process (DPP) which is introduced by [42], and combine it with our algorithm to capture a video summarization. We run our experiments on YouTube and Open Video Project (OVP) datasets from [19].

For each video, we use the linear method of [32] to extract a subset of frames and find a positive semi-definite kernel L with size $n \times n$ where n is the number of extracted frames. Then, we try to find a subset S of frames such that it maximizes $\frac{det(L_S)}{det(I+L)}$ where L_S is the sub-matrix of L restricted to indices corresponding to frames S. Since L is a positive semi-definite matrix, we have $det(L_S) \geq 0$. Interestingly, [38] showed that $\log(det(L_S))$ is a non-monotone function. We use these properties and set $f(S) := \log(det(L_S) + 1)$ to make f a non-monotone non-negative submodular function. Then we run our $(8+\varepsilon)$-approximate dynamic algorithm to find the best S to maximize $f(S)$ such that $|S| \leq k$ for $k \in [10]$.

Figure 1: Video summarization of Susan Boyle's performance on Britain's Got Talent show (video 106) from YouTube.

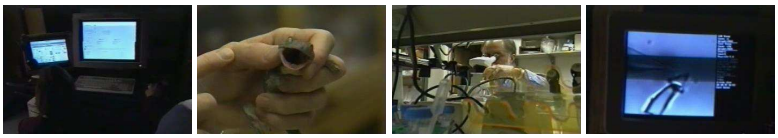


Figure 2: Video summarization for "Senses And Sensitivity, Introduct. to Lecture 1 presenter" (video 36) from OVP.

First, we insert all frames to observe the quality of our algorithm. Figure 1 and 2 are the selected frames by our algorithm for Video 106 from YouTube and Video 36 from OVP, respectively, when we limit the number of selected frames to 4. Then, we create a sequence Ξ of updates of frames

of each video. Similar to [39], we define the sequence as a sliding window model. That is, given a window of size W for a parameter $W \in \mathbb{N}$, a frame is inserted at a time t and will be alive for a window of size W and then we delete that frame.

To evaluate the performance of our algorithm, we benchmark (See Figure 3) the total number of query calls and the submodular value of set S of our algorithm and the streaming algorithm proposed for non-monotone submodular maximization so-called SAMPLE-STREAMING proposed in [27]. This algorithm works as follows: Upon arrival of an element u, with probability $(1-q)$, for a parameter $0 < q < 1$, we ignore u, otherwise (i.e., with probability q), we do the following. If the size of set S that we maintain is less than k, i.e, $|S| < k$ and $\Delta(u|S) > 0$, we add u to S. However, if $|S| = k$, we select an element $v \in S$ for which $\Delta(v : S)$ is minimum possible, where $\Delta(u : S)$ equals to $\Delta(u|S_u)$ where S_u are elements that arrived before u in sequence Ξ. If $\Delta(u|S) \geq (1+c)\Delta(v : S)$ for a constant c, we replace v by u; otherwise, we do nothing. Now we convert this streaming algorithm into a dynamic algorithm. To accomplish this, we restart SAMPLE-STREAMING after every deletion that deletes an element of solution set S that is reported by SAMPLE-STREAMING's outputs. That is, if a deletion does not touch any element in set S, we do nothing; otherwise we restart the streaming algorithm.

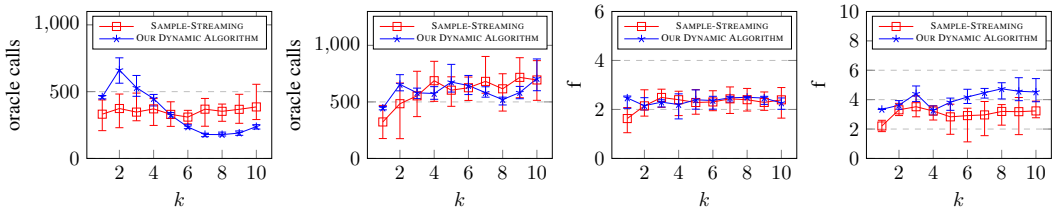


Figure 3: We plot the total number of query calls and the average output of our dynamic algorithm and SAMPLE-STREAMING on video 106 from YouTube and video 36 from OVP. In this figure, from left to right, Sub-figures (a) and (b) are the total oracle calls for video 106 and 36, respectively. Similarly, Sub-figures (c) and (d) are average submodular value for video 106 and 36, respectively.

We run our algorithm for $\varepsilon = k/2$ and compare the total oracle calls and average output of our algorithm and SAMPLE-STREAMING in Figure 3. To prove the approximation guarantee of our dynamic algorithm, we assumed $\varepsilon \leq 1$. However, in practice, it is possible to increase ε up to a certain level without affecting the output of the algorithm significantly. On the other hand, increasing ε reduces the total oracle calls and makes the algorithm faster. As you can see in Figure 3 plots (b) and (d), the submodular value of our algorithm is not worse than the SAMPLE-STREAMING algorithm whose approximation factor is $3 + 2\sqrt{2} \approx 5.828$ which is better than our approximation factor. Thus, our algorithm has an outcome better than our expectation, while its total oracle calls are better than SAMPLE-STREAMING algorithm (look at Figure 3 plots (a) and (c)).

We also empirically study the celebrated Max-Cut problem which is a non-monotone submodular maximization function (See [26]). These experiments are given in the appendix.

4 Conclusion

In this paper, we studied non-monotone submodular maximization subject to cardinality constraint k in the dynamic setting by providing a reduction from this problem to maximizing monotone submodular functions under the cardinality constraint k with a certain kind of algorithms(τ-thresholding algorithms). Moreover, we used our reduction to develop the first dynamic algorithms for this problem. In particular, both our algorithms maintain a solution set whose submodular value is a $(8+\varepsilon)$-approximation of the optimal value and require $O(\varepsilon^{-3}k^3\log^3(n)\log(k))$ and $O(\varepsilon^{-1}k^2\log^3(k))$ oracle queries per update, respectively.

5 Acknowledgements

This work is partially supported by DARPA QuICC NSF AF:Small #2218678, and NSF AF:Small #2114269.

References

[1] Naor Alaluf, Alina Ene, Moran Feldman, Huy L. Nguyen, and Andrew Suh. Optimal streaming algorithms for submodular maximization with cardinality constraints. In *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 6:1–6:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[2] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.

[3] Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. Streaming submodular maximization: massive data summarization on the fly. *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014.

[4] Eric Balkanski, Adam Breuer, and Yaron Singer. Non-monotone submodular maximization in exponentially fewer iterations. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 2359–2370, Red Hook, NY, USA, 2018. Curran Associates Inc.

[5] Kiarash Banihashem, Leyla Biabani, Samira Goudarzi, Mohammadtaghi Hajiaghayi, Peyman Jabbarzade, and Morteza Monemizadeh. Dynamic constrained submodular optimization with polylogarithmic update time. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 1660–1691. PMLR, 23–29 Jul 2023.

[6] Kiarash Banihashem, Leyla Biabani, Samira Goudarzi, MohammadTaghi Hajiaghayi, Peyman Jabbarzade, and Morteza Monemizadeh. Dynamic algorithms for matroid submodular maximization. In *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, arXiv:2306.00959*, 2024.

[7] MohammadHossein Bateni, Mohammad Taghi Hajiaghayi, and Morteza Zadimoghaddam. Submodular secretary problem and extensions. *ACM Trans. Algorithms*, 9(4):32:1–32:23, 2013.

[8] MohammadHossein Bateni, MohammadTaghi Hajiaghayi, and Morteza Zadimoghaddam. Submodular secretary problem and extensions. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 13th International Workshop, APPROX 2010, and 14th International Workshop, RANDOM 2010, Barcelona, Spain, September 1-3, 2010. Proceedings*, volume 6302 of *Lecture Notes in Computer Science*, pages 39–52. Springer, 2010.

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

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

[11] Niv Buchbinder, Moran Feldman, Joseph (Seffi) Naor, and Roy Schwartz. Submodular maximization with cardinality constraints. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, page 1433–1452, USA, 2014. Society for Industrial and Applied Mathematics.

[12] Larry Carter and Mark N. Wegman. Universal classes of hash functions (extended abstract). In *Proceedings of the 9th Annual ACM Symposium on Theory of Computing, May 4-6, 1977, Boulder, Colorado, USA*, pages 106–112, 1977.

[13] Chandra Chekuri, Shalmoli Gupta, and Kent Quanrud. Streaming algorithms for submodular function maximization. In *Automata, Languages, and Programming*, pages 318–330, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[14] Xi Chen and Binghui Peng. On the complexity of dynamic submodular maximization. In *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 1685–1698. ACM, 2022.

[15] Yixin Chen and Alan Kuhnle. Practical and parallelizable algorithms for non-monotone submodular maximization with size constraint, 2022.

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

[17] Abhimanyu Das and David Kempe. Submodular meets spectral: Greedy algorithms for subset selection, sparse approximation and dictionary selection. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 1057–1064. Omnipress, 2011.

[18] Abhimanyu Das and David Kempe. Approximate submodularity and its applications: Subset selection, sparse approximation and dictionary selection. *J. Mach. Learn. Res.*, 19:3:1–3:34, 2018.

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

[20] Delbert Dueck and Brendan J. Frey. 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*, pages 1–8. IEEE Computer Society, 2007.

[21] Paul Duetting, Federico Fusco, Silvio Lattanzi, Ashkan Norouzi-Fard, and Morteza Zadimoghaddam. Fully dynamic submodular maximization over matroids. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 8821–8835. PMLR, 23–29 Jul 2023.

[22] Rick Durrett. *Probability: Theory and Examples, 4th Edition*. Cambridge University Press, 2010.

[23] Khalid El-Arini and Carlos Guestrin. Beyond keyword search: discovering relevant scientific literature. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 21-24, 2011*, pages 439–447. ACM, 2011.

[24] Ethan R. Elenberg, Rajiv Khanna, Alexandros G. Dimakis, and Sahand N. Negahban. Restricted strong convexity implies weak submodularity. *CoRR*, abs/1612.00804, 2016.

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

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

[27] Moran Feldman, Amin Karbasi, and Ehsan Kazemi. Do less, get more: Streaming submodular maximization with subsampling. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 730–740, 2018.

[28] Moran Feldman, Joseph Naor, and Roy Schwartz. Nonmonotone submodular maximization via a structural continuous greedy algorithm - (extended abstract). In *Automata, Languages and Programming - 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part I*, volume 6755 of *Lecture Notes in Computer Science*, pages 342–353. Springer, 2011.

[29] Satoru Fujishige. Theory of submodular programs: A fenchel-type min-max theorem and subgradients of submodular functions. *Math. Program.*, 29(2):142–155, 1984.

[30] Shayan Oveis Gharan and Jan Vondrák. Submodular maximization by simulated annealing. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '11, page 1098–1116, USA, 2011. Society for Industrial and Applied Mathematics.

[31] Shayan Oveis Gharan and Jan Vondrák. Submodular maximization by simulated annealing. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 1098–1116. SIAM, 2011.

[32] Boqing Gong, Wei-Lun Chao, Kristen Grauman, and Fei Sha. Diverse sequential subset selection for supervised video summarization. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2069–2077, 2014.

[33] Anupam Gupta, Aaron Roth, Grant Schoenebeck, and Kunal Talwar. Constrained non-monotone submodular maximization: Offline and secretary algorithms. In *Internet and Network Economics - 6th International Workshop, WINE 2010, Stanford, CA, USA, December 13-17, 2010. Proceedings*, volume 6484 of *Lecture Notes in Computer Science*, pages 246–257. Springer, 2010.

[34] Jason Hartline, Vahab Mirrokni, and Mukund Sundararajan. Optimal marketing strategies over social networks. In *Proceedings of the 17th international conference on World Wide Web*, pages 189–198, 2008.

[35] Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *J. ACM*, 53(3):307–323, 2006.

[36] Ehsan Kazemi, Morteza Zadimoghaddam, and Amin Karbasi. 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*, pages 2549–2558. PMLR, 2018.

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

[38] Alex Kulesza and Ben Taskar. Determinantal point processes for machine learning. *Found. Trends Mach. Learn.*, 5(2-3):123–286, 2012.

[39] Silvio Lattanzi, Slobodan Mitrovic, Ashkan Norouzi-Fard, Jakub Tarnawski, and Morteza Zadimoghaddam. Fully dynamic algorithm for constrained submodular optimization. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[40] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[41] Paul Liu and Jan Vondrák. Submodular optimization in the mapreduce model. In *2nd Symposium on Simplicity in Algorithms, SOSA@SODA 2019, January 8-9, 2019 - San Diego, CA, USA*, volume 69 of *OASICS*, pages 18:1–18:10. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[42] Odile Macchi. The coincidence approach to stochastic point processes. *Advances in Applied Probability*, 7(1):83–122, 1975.

[43] Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, and Amin Karbasi. Fast constrained submodular maximization: Personalized data summarization. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1358–1367, New York, New York, USA, 20–22 Jun 2016. PMLR.

[44] Baharan Mirzasoleiman, Stefanie Jegelka, and Andreas Krause. Streaming non-monotone submodular maximization: Personalized video summarization on the fly. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 1379–1386. AAAI Press, 2018.

[45] Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. Deletion-robust submodular maximization: Data summarization with "the right to be forgotten". In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 2449–2458. PMLR, 2017.

[46] Morteza Monemizadeh. Dynamic submodular maximization. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[47] Ashkan Norouzi-Fard, Jakub Tarnawski, Slobodan Mitrovic, Amir Zandieh, Aidasadat Mousavifar, and Ola Svensson. Beyond 1/2-approximation for submodular maximization on massive data streams. 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*, pages 3826–3835. PMLR, 2018.

[48] Binghui Peng. Dynamic influence maximization. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 10718–10731, 2021.

[49] Sharon Qian and Yaron Singer. Fast parallel algorithms for statistical subset selection problems. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2019. Curran Associates Inc.

[50] Ian Simon, Noah Snavely, and Steven M. Seitz. Scene summarization for online image collections. In *IEEE 11th International Conference on Computer Vision, ICCV 2007, Rio de Janeiro, Brazil, October 14-20, 2007*, pages 1–8. IEEE Computer Society, 2007.

[51] Ruben Sipos, Adith Swaminathan, Pannaga Shivaswamy, and Thorsten Joachims. Temporal corpus summarization using submodular word coverage. In *21st ACM International Conference on Information and Knowledge Management, CIKM'12, Maui, HI, USA, October 29 - November 02, 2012*, pages 754–763. ACM, 2012.

[52] Sebastian Tschiatschek, Rishabh K Iyer, Haochen Wei, and Jeff A Bilmes. Learning mixtures of submodular functions for image collection summarization. In *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

A Guessing optimal value

The dynamic algorithm that we proposed in the main part of the paper, assumes the value of OPT is given as a parameter. However, we know that the value of OPT^* is not known; it changes after every insertion or deletion, and our final goal in this algorithm is to find a correct approximation of it. As discussed earlier, to achieve this goal, we run parallel instances of our dynamic algorithm for different guesses of the optimal value. We use $(1+\varepsilon')^i$, where $i \in \mathbb{Z}$ as our guesses for the optimal value. We showed that at any fixed time, the run in our algorithm whose corresponding OPT $(1+\varepsilon')$-approximates OPT^* also finds a solution with a guaranteed approximation factor of $(8+\varepsilon)$ or $(10+\varepsilon)$, depending on the value of α. However, If ρ is the ratio between the maximum and minimum non-zero possible value of the optimal solution at any time t, then the number of parallel instances of our algorithm will be $O(\log_{(1+\varepsilon')} \rho) = O(\varepsilon^{-1} \cdot \log \rho)$. This incurs an extra $O(\varepsilon^{-1} \cdot \log \rho)$-factor in the query complexity of our dynamic algorithm.

Next, we show how to replace this extra factor with an extra factor of $O(\varepsilon^{-1} \cdot \log k)$, which is independent of ρ. We use the well-known technique that has been also used in [39]. In particular, for every element v, we only add it to those instances i for which we have $\frac{\varepsilon'}{k} \cdot (1+\varepsilon')^i \leq f(v) \leq (1+\varepsilon')^i = OPT$. Note that for run i corresponding to the guess of optimal value $OPT = (1+\varepsilon')^i$, $f(v) > (1+\varepsilon')^i$ means that $f(v)$ is greater than OPT. Therefore, we can safely ignore this element for this run. In fact, this run might be useful for finding the solution only if OPT is a $(1+\varepsilon')$ approximation of OPT^*, which can only happen after the deletion of v, so ignoring v has no effect on the solution and saves query complexity.

Moreover, ignoring all elements v whose $f(v) < \frac{\varepsilon'}{k} \cdot (1+\varepsilon')^i$ in the run i corresponding to the guess of optimal value $OPT = (1+\varepsilon')^i$, decreases the submodular value of its solution by at most $OPT \cdot \varepsilon'$. This changes the approximation guarantee by $O(\varepsilon')$, which can be ignored.

In this way, every element v is added to at most $O(\varepsilon^{-1} \log k)$ parallel instances. Thus, after every insertion or deletion, we need to update only $O(\varepsilon^{-1} \log k)$ instances of our dynamic algorithm.

B Proof of Lemma 2.3

Lemma 2.3. Suppose set S satisfies Property 2 of Definition 1.1. In other words, S has less than k elements and for any $v \in V \setminus S$, the marginal gain $\Delta(v|S) < \tau$. Then, for any arbitrary subset $C \subseteq V$, we have $f(S) \geq f(S \cup C) - |C| \cdot \tau$.

Proof. For the sake of contradiction, assume that $f(S) < f(S \cup C) - |C| \cdot \tau$. Rewriting this bound in terms of the marginal gain of set C with respect to set S gives: $|C| \cdot \tau < f(S \cup C) - f(S) = \Delta(C|S) \leq \sum_{v \in C} \Delta(v|S)$, where the second inequality holds because f is a submodular function. This implies that $\tau < \dfrac{\sum_{v \in C} \Delta(v|S)}{|C|}$, which means that there must be an element $v^* \in C$ whose marginal gain is more than τ. Observe that $\Delta(v^*|S) > \tau > 0$ and then $v^* \notin S$. That is, $\Delta(v^*|S) > \tau$ for an element $v^* \in V \setminus S$, which contradicts the assumption that we made. Thus, the claim must be true. □

C Proof of Lemma 2.4

Lemma 2.4. Let random variable Q_t denote the number of oracle calls that the algorithm in Theorem 1.2 makes after the t-th update. Then if we choose local search method as SUBSETSELECTION subroutine we have

$$\mathbf{E}[\sum_{t=1}^{T} Q_t] \leq T(\min(k \cdot g(n,k), g(n,k)^2) + O(k)) \ , \tag{1}$$

and if we choose random sampling method as SUBSETSELECTION subroutine we have

$$\mathbf{E}[\sum_{t=1}^{T} Q_t] \leq T \cdot \min(k \cdot g(n,k), g(n,k)^2) \ , \tag{2}$$

where $g(n,k)$ is the number of oracle calls that thresholding algorithm DYNAMICTHRESHOLD makes after each update.

Proof. We first distinguish the queries depending on the operation that makes the queries. We denote the queries made by an update in instance $\mathcal{I}_1$ as *type-1 queries*, queries made by an update in instance $\mathcal{I}_2$ as *type-2 queries*, and the queries made by SUBSETSELECTION as *type-3 queries*. We will use $Q_t^{(1)}, Q_t^{(2)}$ and $Q_t^{(3)}$ to denote the number of type-1, type-2 and type-3 queries, the algorithms make for the t-th operation. As all of the queries made by the algorithm fall into one of the type-1, type-2, and type-3 categories, $Q_t = Q_t^{(1)} + Q_t^{(2)} + Q_t^{(3)}$. Thus, it suffices to bound $\mathbf{E}[\sum_{t=1}^{T} Q_t^{(\ell)}]$ seperately for each $\ell \in \{1,2,3\}$.

Type-1 queries. We start with $\mathbf{E}[\sum_{t=1}^{T} Q_t^{(1)}]$. By assumption, the expected amortized number of query calls made by $\mathcal{I}_1$ is at most $g(n,k)$. Thus, $\mathbf{E}[\sum_{t=1}^{T} Q_t^{(1)}] \leq T \cdot g(n,k)$.

Type-2 queries. We now consider $\sum_{t=1}^{T} Q_t^{(2)}$. For each t, let random variable X_t denote $|S_1 \backslash Z| + |Z \backslash S_1|$ during the t-th call to UPDATE and let $v_t^{(2),1}, \ldots, v_t^{(2),X_t}$ denote the elements inserted to or deleted from $\mathcal{I}_2$ during this call. In other words, $\{v_t^{(2),1}, \ldots, v_t^{(2),X_t}\} = (S_1 \backslash Z) \cup (Z \backslash S_1)$.

For each $1 \leq i \leq X_t$, let $Q_t^{(2),i}$ denote the number of oracle calls $\mathcal{I}_2$ makes because of the insertion or deletion of $v_t^{(2),i}$. It is clear that $Q_t^{(2)} = \sum_{i=1}^{X_t} Q_t^{(2),i}$. By assumption, the amortized expected query complexity of $\mathcal{I}_2$ is at most $g(n,k)$. Conditioning on $X_1, \ldots, X_T$, we obtain

$$\begin{aligned}
&\mathbf{E}[\sum_{t=1}^{T} Q_t^{(2)} | X_1, \ldots X_T] = \mathbf{E}[(\sum_{t=1}^{T} \sum_{i=1}^{X_t} Q_t^{(2),i}) | X_1, \ldots X_T] \\
&= \sum_{t=1}^{T} \sum_{i=1}^{X_t} \mathbf{E}[Q_t^{(2),i} | X_1, \ldots X_T] \overset{(a)}{\leq} \sum_{t=1}^{T} \sum_{i=1}^{X_t} g(n,k) = (\sum_{t=1}^{T} X_t) \cdot g(n,k) \ ,
\end{aligned}$$

where for (a), we have used the guarantee on the amortized query complexity of $\mathcal{I}_2$. Note that as the guarantee holds in the oblivious adversarial model, it holds even after conditioning on $X_1, \ldots, X_T$. Taking expectation over $X_1, \ldots, X_T$, we obtain $\mathbf{E}[\sum_{t=1}^{T} Q_t^{(2)}] = g(n,k) \cdot \mathbf{E}[\sum_{t=1}^{T} X_t]$. It remains to bound $\mathbf{E}[\sum_{t=1}^{T} X_t]$. As $X_t = |S_1 \backslash Z| + |Z \backslash S_1|$, we can always guarantee $X_t \leq 2k$, leading to the bound $\mathbf{E}[\sum_{t=1}^{T} X_t] \leq 2k \cdot T$. In addition, by Property 2 in Definition 1.1, we can guarantee $X_t \leq Q_t$, which leads to the bound $\mathbf{E}[\sum_{t=1}^{T} X_t] \leq \mathbf{E}[\sum_{t=1}^{T} Q_t^{(1)}] \leq T \cdot g(n,k)$. Thus, we obtain $\mathbf{E}[\sum_{t=1}^{T} Q_t^{(2)}] \leq T \cdot \min(k \cdot g(n,k), g(n,k)^2)$.

Type-3 queries. Finally, we bound $\mathbf{E}[\sum_{t=1}^{T} Q_t^{(3)}]$. For the $(8+\varepsilon)$-approximation algorithm, the number of type-3 queries is the number of queries that SUBSETSELECTION(S_1) makes, which is $O(|S_1|)$. Since, the size of S_1 is always bounded by k, we then have $\mathbf{E}[Q_t^{(3)}] \leq O(|S_1|) = O(k)$, which further implies $\mathbf{E}[\sum_{t=1}^{T} Q_t^{(3)}] \leq T \cdot O(k)$. For the $(10+\varepsilon)$-approximation algorithm, since UNIFORMSUBSET make no queries, the number of type-3 queries is always zero, leading to the bound $\mathbf{E}[\sum_{t=1}^{T} Q_t^{(3)}] = 0$.

Combining the above the bounds for type-1, type-2, and type-3 queries, we obtain the bound (1) for the $(8+\varepsilon)$-approximation algorithm and the bound (2) for the $(10+\varepsilon)$-approximation algorithm. □

D τ-thresholding algorithm

In this section, we show that the algorithms provided in [46] and [6] are indeed τ-thresholding for any given τ, and can be used in our Metatheorem 1.2.

The following is the modified version of the monotone submodular maximization algorithm provided by [6], ready to be used in our reduction.

Algorithm 4 MONOTONECARDINALITYCONSTRAINTLEVELING(k, τ)

function INIT(V)
    $I_0 \leftarrow \emptyset, \quad R_0 \leftarrow \emptyset, \quad T \leftarrow 0$

function CONSTRUCTLEVEL (i)
    Let P be a random permutation of elements of R_i and $\ell \leftarrow i$
    for e in P **do**
        if PROMOTE$(I_{\ell-1}, e) = True$ **then**
            $e_\ell \leftarrow e, \quad I_\ell \leftarrow I_{\ell-1} + e, \quad z \leftarrow \ell, \quad R_{\ell+1} \leftarrow \emptyset, \quad \ell \leftarrow \ell + 1$
        else
            Run binary search to find the lowest $z \in [i, \ell - 1]$ such that PROMOTE$(I_z, e) = False$
        for $r \leftarrow i + 1$ **to** z **do**
            $R_r \leftarrow R_r + e$.
    $T \leftarrow \ell - 1$, which is the final value of ℓ in the for-loop above subtracted by one

function PROMOTE(I, e)
    if $f(I + e) - f(I) \geq \tau$ and $|I| < k$ **then**
        return True
    return False

Algorithm 5 MONOTONECARDINALITYCONSTRAINTSUBROUTINES(k, τ)

function INSERT(v)
    $R_0 \leftarrow R_0 + v$.
    for $i \leftarrow 1$ **to** $T + 1$ **do**
        if PROMOTE(I_{i-1}, v) = False **then**
            break
        $R_i \leftarrow R_i + v$.
        Let $p = 1$ with probability $\frac{1}{|R_i|}$, and otherwise $p = 0$.
        if $p = 1$ **then**
            $e_i \leftarrow v, \quad I_i \leftarrow I_{i-1} + v$
            $R_{i+1} = \{e' \in R_i : \text{PROMOTE}(I_i, e') = True\}$
            CONSTRUCTLEVEL $(i + 1)$
            break

function DELETE(v)
    $R_0 \leftarrow R_0 - v$
    for $i \leftarrow 1$ **to** T **do**
        if $v \notin R_i$ **then**
            break
        $R_i \leftarrow R_i - v$
        if $e_i = v$ **then**
            CONSTRUCTLEVEL (i).
            break

function EXTRACT
    return I_T

This algorithm maintains a leveled data structure consisting of elements $e_1, \cdots, e_T$, sets $R_0 \supseteq R_1 \supset \cdots \supset R_T \supset R_{T+1} = \emptyset$, and $I_0, I_1, \cdots, I_T$, $I_i = I_{i-1} + e_i$ for any $i \in [1, T]$, and $T \leq k$. For any $i \in [1, T]$, we have $f(I_{i-1} + e_i) - f(I_{i-1}) \geq \tau$. Additionally, if $e \notin I_T$, you can infer that either $T = k$ or $f(I_T + e) - f(I_T) < \tau$, considering the submodularity of the function f and the fact that e has been filtered out of R_j for some $j \in [1, T]$ due to its low marginal gain with respect to I_j or the size of I_j. Combining the last two facts establishes Property 1 of Definition 1.1. Regarding Property 2, when the update is an insertion, the number of changes in the output is either 0 or at most $|R_i|$ if CONSTRUCTLEVEL $(i + 1)$ gets invoked. Meanwhile, the number of oracle queries is at least 1 or

$i + |R_i|$ just by considering the oracle queries made before the invocation of CONSTRUCTLEVEL . Also, when the update is a deletion, the number of changes in the output is either 0 or at most $|R_i|$ if CONSTRUCTLEVEL (i) gets invoked. However, the number of oracle queries is either 0 or at least R_i even with discarding any oracle call made during the binary searches. Therefore, this algorithm also satisfies Property 2 of Definition 1.1. The query complexity of this algorithm is $O(klogk)$ per update. [See Lemma 91 in [6] where their parallel runs, which we do not use, are ignored.]

The algorithm of [46] also has a similar construction. It constructs its final output G in multiple *levels*. Initializing $G_0 := \emptyset, R_0 = V$ and $i = 1$, it repeatedly performs the following actions.

1. Filtering Step: The algorithm shaves off, i.e., filters, those elements $e \in V$ whose marginal gain with respect to G_i is less than τ and collects the remaining elements in R_{i+1}. In particular, $R_{i+1} = \{e \in R_i : \Delta(e|G_i) \geq \tau\}$. If there are no remaining elements, the algorithm stops and G_i is returned as the final output. Otherwise, i is increased by one and the algorithm proceeds to the next step

2. Greedy Step: The algorithm chooses a set S_i taken from R_i uniformly at random. It then creates G_i by adding elements of S_i to G_{i-1} one by one, as long as the marginal gain of the element with respect to G_i is at least τ. In particular, the algorithm initializes $G_i := G_{i-1}$ and then iterates through S, adding element e to G_i if $\Delta(e|G_i) \geq \tau$.
   If at any point the size of G_i reaches k, the algorithm stops, and $G := G_i$ is returned as the final output.

This approach forms a leveled construction characterized by the sets $\emptyset = G_0 \subset G_1 \subset \cdots \subset G_{i^*} = G$ where i^* denotes the final value of i (which is $O(k \log(n))$, See Corollary 5 in their paper). In order to handle updates, whenever an element in G is deleted, the construction is restarted from the level where the deletion occurred. As for insertions, whenever an element e is inserted, the algorithm adds it to each level i as long as it is not filtered beforehand, i.e., $\Delta(e|G_{i-1}) \geq \tau$. Once the element is added, the construction may be restarted with some probability. We refer to [46] for a more detailed analysis along with the pseudocode.
Given this description, if $|G| = k$, then $f(G) \geq k \cdot \tau$ as all elements added to G increased $f(G)$ by at least τ, which proves Property 1.a. As for the case of $|G| < k$, for each $e \in V$, since e was filtered out at some point, it satisfies $\Delta(e|G_i) \leq \tau$ for some $i \leq i^*$. Since f is submodular and $G_i \subseteq G$, this proves Property 1.b. Indeed, these properties are formally proved in Lemma 3 of [46].
As for Property 2, whenever a level is restarted, the number of changed elements is at most the size of the level (i.e., $|R_i|$) since in the worst case, all elements in R_i are added to G_i. The number of queries the algorithm makes however is at least the size of the level as the filtering step performs $|R_i|$ queries. The number of changed elements is therefore bounded by the number of queries as desired. Finally, the guarantee on the expected amortized query complexity is provided in [46] (Lemma 7 and Lemma 8).

E Empirical results

E.1 Max-Cut.

In this section, we study the celebrated Max-Cut problem which is a non-monotone submodular maximization function (See [26]). In this problem, given a graph $G(V, E)$, we would like to compute a set of vertices $S \subseteq V$ that maximizes the size of cut $C(S, V \setminus S)$.

We denote the number of edges in cut $C(S, V \setminus S)$ by $f(S) = |C(S, V \setminus S)|$. For the input of our algorithm, we use real-world data sets that we collect from SNAP Data Collection [40]: loc-Gowalla and web-Stanford. Similar to the video summarization experiment, we create a sequence of updates of vertices of graph $G(V, E)$ in a sliding window model. For comparing the performance of our algorithm, in addition to SAMPLE-STREAMING, we run RANDOM ALGORITHM that selects k random elements from the remaining elements after each update.

Finally, we run our algorithm and SAMPLE-STREAMING algorithm ten times on different values of k. In plots (a) and (c), it is shown that our algorithm has fewer calls than SAMPLE-STREAMING when k is small. In plots (b) and (d), we show that in practice, our algorithm has a better approximation guarantee than what we proved theoretically.

As for the non-monotone submodular function, we study the celebrated **Max-Cut** problem. In this problem, given a graph $G(V, E)$, we would like to compute a set of vertices $S \subseteq V$ that maximizes the size of cut $C(S, V \setminus S)$ which is the number of edges between S and $V \setminus S$. We denote the number of edges in cut $C(S, V \setminus S)$ by $f(S) = |C(S, V \setminus S)|$. This function is non-monotone submodular (see [26]). Our goal is, given a graph $G(V, E)$ and a parameter $k \in \mathbb{N}$, to compute a Max-Cut $C(S, V \setminus S)$ of G for which the number of vertices in S is at most $|S| \leq k$.

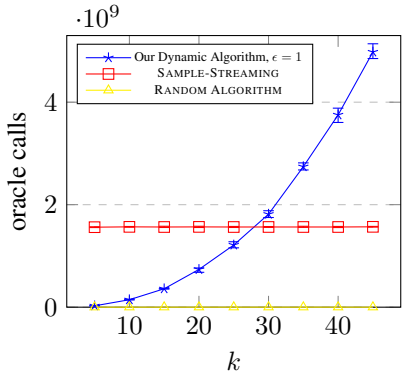
For the input of our algorithm, we use real-world data sets that we choose from SNAP Data Collection [40]: (1) loc-Gowalla (Gowalla location based online social network) that has $196,591$ vertices and $950,327$ edges, and (2) web-Stanford (Web graph of Stanford.edu) that has $281,903$ vertices and $2,312,497$ edges.

We create a sequence Ξ of updates of vertices of graph $G(V, E)$. Similar to [39], we define the sequence as a sliding window model. That is, given a window size W, a vertex is inserted at a time t and will be alive for a window of size W and then we delete that vertex. To make the sequence an adversarial sequence, we insert vertices in descending order of their degrees, and each vertex remains alive for a window of W vertex insertions and deletions, and later the vertex will be deleted from the graph.

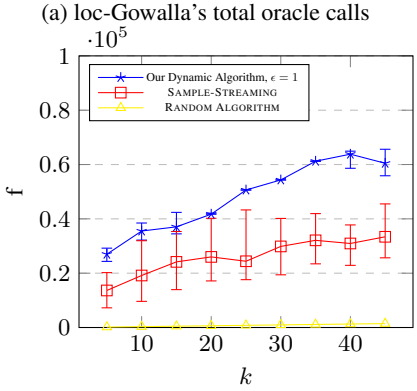
To evaluate the performance of our algorithm, we benchmark the total number of query calls and the submodular value of set S of our algorithm and the streaming algorithm proposed for non-monotone submodular maximization so-called SAMPLE-STREAMING proposed in [27]. This algorithm works as follows: Upon arrival of an element u, with probability $(1 - q)$, for a parameter $0 < q < 1$, we ignore u, otherwise (i.e., with probability q), we do the following. If the size of set S that we maintain is less than k, i.e, $|S| < k$ and $\Delta(u|S) > 0$, we add u to S. However, if $|S| = k$, we select an element $v \in S$ for which $\Delta(v : S)$ is minimum possible, where $\Delta(u : S)$ equals to $\Delta(u|S_u)$ where S_u are elements that arrived before u in the stream. If $\Delta(u|S) \geq (1 + c)\Delta(v : S)$ for a constant c, we replace v by u; otherwise, we do nothing. Now we convert this streaming algorithm into a dynamic algorithm. For that, we restart Sample-Streaming after every deletion that deletes a vertex of reported set S by SAMPLE-STREAMING's outputs. That is, if a deletion does not touch any vertex in set S, we do nothing; otherwise we restart the streaming algorithm.

Finally, we run our algorithm for $\varepsilon = 1$ and the SAMPLE-STREAMING algorithm ten times on different values of k and benchmark them in Figure 4. In plots (a) and (c), it is shown that our algorithm has fewer oracle calls than SAMPLE-STREAMING when k is small. The number of oracle calls of the SAMPLE-STREAMING algorithm is $O(n^2)$ in the worst case which is independent of k. Therefore, by increasing k, the number of oracle calls of our algorithm becomes worse than the number of oracle calls of the Sample-Streaming algorithm. On the other hand, our algorithm works significantly better when the size of the input graph increases. As is shown in Figure 4, in plots (a) and (c), running on a bigger graph increases the number of oracle calls of SAMPLE-STREAMING more than our algorithm.

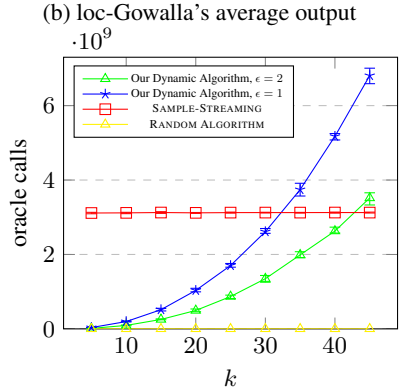
In addition, in plots (b) and (d), we show that in practice, our algorithm has a better approximation guarantee than what we proved theoretically. Interestingly, the approximation factor of SAMPLE-STREAMING is $3 + 2\sqrt{2} \approx 5.828$ which is better than the approximation factor that we proved for our dynamic algorithm, but the plots (b) and (d) show that in reality, our algorithm performs better. Moreover, we run RANDOM ALGORITHM that selects k random elements from remaining elements after each update. As we see in Figure 4 plots (b) and (d), RANDOM ALGORITHM performs poorly. Also, we run our algorithm for $\varepsilon = 2$. As is shown in plot (d), the output has not changed significantly by increasing ε, and it remains more than the average output of SAMPLE-STREAMING. On the other hand, the number of oracle calls decreased significantly (plot (c)). Therefore, for bigger k, increasing the ε is a good option to get an appropriate output in a short time.



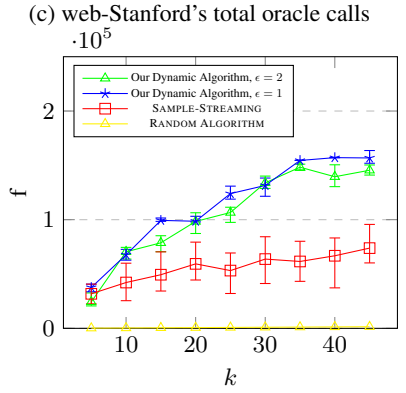
(a) loc-Gowalla's total oracle calls



(b) loc-Gowalla's average output



(c) web-Stanford's total oracle calls

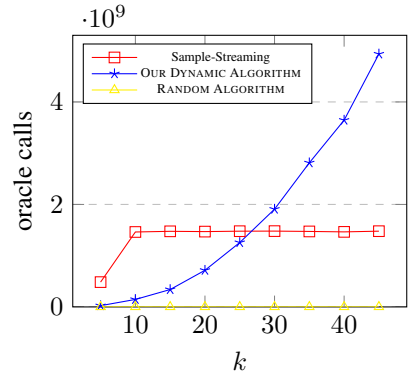


(d) web-Stanford's average output

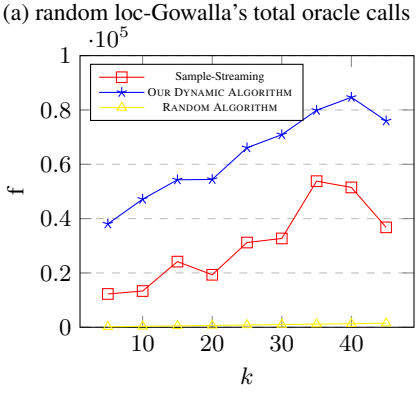
Figure 4: We plots the total number of query calls and the average output of our dynamic algorithm, SAMPLE-STREAMING, and RANDOM ALGORITHM on two data sets.

E.2 Additional experiments

In this section, in addition to our previous experiments, we run another experiment. In Figure 5, we use a slightly different type of input where we first insert all vertices in descending order of their degrees. Then, we delete them in a semi-randomized order. For deletions, we first sort vertices in descending order of their degrees (ties are broken arbitrarily) and with half probability we swap a vertex with its left or right neighbors. This will be the order that we follow to delete vertices. We use this type of input to make our input more general, such that no sliding windows algorithm works on it. Also, the noise that we added to the deletion makes our input more real.



(a) random loc-Gowalla's total oracle calls

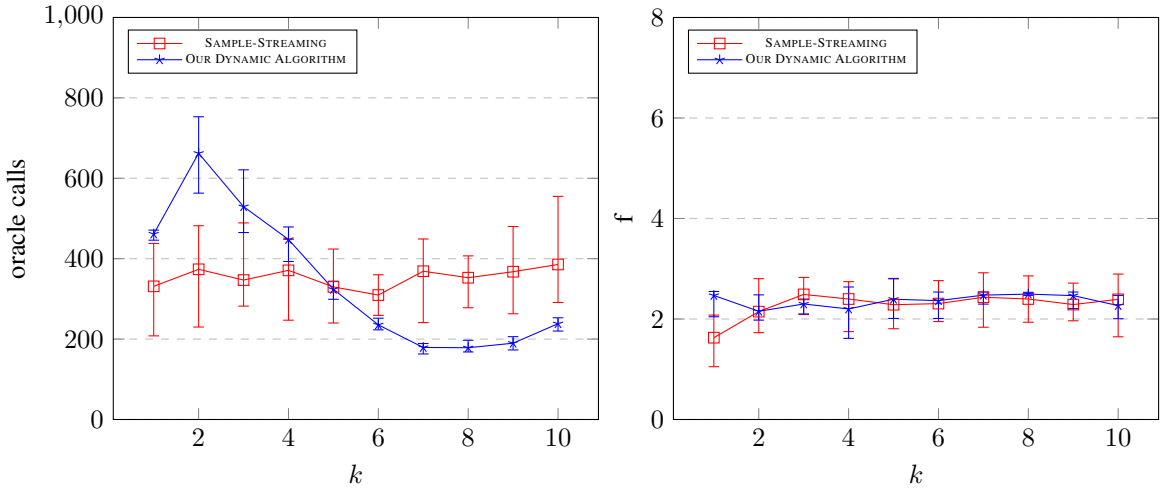


(b) random loc-Gowalla's average output

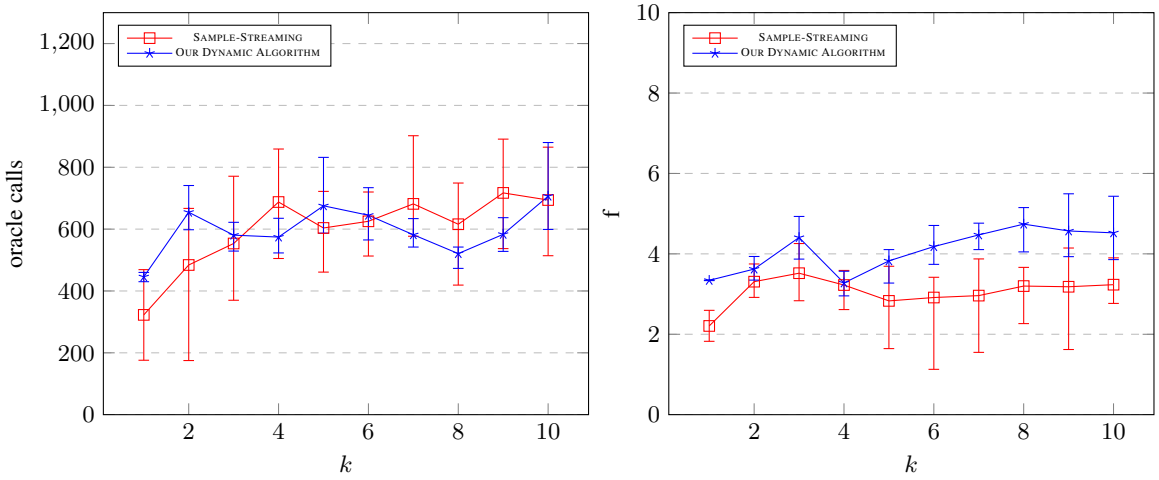
Figure 5: We plot the total number of query calls and the average output of our dynamic algorithm, SAMPLE-STREAMING, and RANDOM ALGORITHM on loc-Gowalla with a noisy order for deletion.

E.3 Larger version of video summarization plots

Here in Figure 6 we provided a larger version of Figure 3 to make it easier to read.



(a) Video 106 total oracle calls and average output



(b) Video 36 total oracle calls and average output

Figure 6: We plot the total number of query calls and the average output of our dynamic algorithm and SAMPLE-STREAMING on video 106 from YouTube and video 36 from OVP.