On the Complexity of Dynamic Submodular Maximization

Xi Chen
xichen@cs.columbia.edu
Columbia University
New York, United States

Binghui Peng
bp2601@columbia.edu
Columbia University
New York, United States

ABSTRACT

We study dynamic algorithms for the problem of maximizing a monotone submodular function over a stream of n insertions and deletions. We show that any algorithm that maintains a $(0.5 + \epsilon)$-approximate solution under a cardinality constraint, for any constant $\epsilon > 0$, must have an amortized query complexity that is *polynomial* in n. Moreover, a linear amortized query complexity is needed in order to maintain a 0.584-approximate solution. This is in sharp contrast with recent dynamic algorithms of [LMN+20, Mon20] that achieve $(0.5 - \epsilon)$-approximation with a polylog(n) amortized query complexity.

On the positive side, when the stream is insertion-only, we present efficient algorithms for the problem under a cardinality constraint and under a matroid constraint with approximation guarantee $1 - 1/e - \epsilon$ and amortized query complexities $O(\log(k/\epsilon)/\epsilon^2)$ and $k^{\widetilde{O}(1/\epsilon^2)} \log n$, respectively, where k denotes the cardinality parameter or the rank of the matroid.

CCS CONCEPTS

• **Theory of computation → Submodular optimization and polymatroids**.

KEYWORDS

Submodular maximization, Dynamic algorithm, Query complexity

ACM Reference Format:
Xi Chen and Binghui Peng. 2022. On the Complexity of Dynamic Submodular Maximization. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing (STOC '22), June 20–24, 2022, Rome, Italy*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3519935.3519951

1 INTRODUCTION

Initiated by the classical work of [35, 36] in the 1970s, submodular maximization has developed into a central topic of discrete optimization during past decades [14, 15, 19, 25, 39] (see [12] for a survey). Capturing the natural notion of diminishing returns, submodular functions and their optimization problems have found numerous applications in areas such as machine learning [28, 40], data mining [31], algorithmic game theory [38], social networks [29], etc. The canonical form of the problem is to maximize a monotone submodular function under a cardinality constraint k, for which the greedy algorithm of [36] achieves the optimal approximation ratio of $1 - 1/e$ [24, 35]. However, despite the simplicity and optimality of this celebrated algorithm, there has been a surge of recent research effort to reexamine the problem under a variety of computational models motivated by unique challenges of working with massive datasets. These include streaming algorithms [2, 26], parallel algorithms [6–8, 17, 21–23], learning algorithms [4, 5] and distributed algorithms [10, 11, 33].

Dynamic submodular maximization. We study the power and limitations of *dynamic* algorithms for submodular maximization. In this model, the set of elements that one can choose from is subject to changes incurred by a stream of n insertions and deletions. Letting V_t denote the current set of elements after the first t operations, an algorithm needs to maintain a subset $S_t \subseteq V_t$ of size at most k that achieves a certain approximation guarantee for every round t, and its performance is measured by its amortized query complexity (i.e., the average number of queries it makes per operation on the underlying unknown monotone submodular function); see the formal definition of the model in Section 2.

The dynamic model is motivated by real-world applications of submodular maximization over massive datasets that evolve frequently. Many of these applications arise in machine learning and data mining tasks, including the data subset selection problem [28, 40], movie recommendation system [9], influence maximization in social networks [29], etc. As an example, in influence maximization, one is given a social network as well as a stochastic diffusion model, and the goal is to select a seed set of size at most k to maximize the influence spread over the network. The spread function is submodular for many well-studied models and therefore, the problem becomes submodular maximization with a cardinality constraint. Given that social networks, like Twitter or Facebook, are involving continuously over time, an old seed set could become outdated quickly. A natural rescue is to have an efficient dynamic algorithm that maintains a seed set over time.

The problem of dynamic submodular maximization under a cardinality constraint was studied in two recent papers [30, 34], giving algorithms that achieve an $(1/2 - \epsilon)$-approximation guarantee with amortized query complexities that are $O(k^2 \log^2 n \cdot \epsilon^{-3})$ and $O(\log^8 n \cdot \epsilon^{-6})$ in a stream of length n. Compared with the $1 - 1/e$ approximation guarantee of the offline greedy algorithm, however, a natural open question posted by [30] is whether the $1/2$ can be further improved under the dynamic setting, or even under the setting when the stream is insertion-only.

1.1 Our Results

We resolve the open question of [30] by showing that any algorithm with an approximation ratio of $1/2 + \epsilon$ must have amortized query complexity polynomial in the stream length n:

STOC '22, June 20–24, 2022, Rome, Italy

ACM ISBN 978-1-4503-9264-8/22/06…$15.00
https://doi.org/10.1145/3519935.3519951

Theorem 1.1. *For any constant $\epsilon > 0$, there is a constant $C_\epsilon > 0$ with the following property. When $k \geq C_\epsilon$, any randomized algorithm that achieves an approximation ratio of $1/2 + \epsilon$ for dynamic submodular maximization under cardinality constraint k requires amortized query complexity $n^{\widetilde{\Omega}(\epsilon)}/k^3$.*

Moreover, we show that any algorithm with approximation ratio 0.584 must have an amortized query complexity that is linear in n:

Theorem 1.2. *There is a constant $C > 0$ with the following property. When $k \geq C \log n$, any randomized algorithm for dynamic submodular maximization under cardinality constraint k that obtains an approximation guarantee of 0.584 must have amortized query complexity at least $\Omega(n/k^3)$.*

In the proof of both theorems we construct a family of hard functions to hide a secret matching. The main challenge is to achieve the following two properties at the same time: The first one, which we will refer to as the *large-gap* property, states that any dynamic algorithm with a certain approximation guarantee can only succeed by recovering the secret matching hidden in the function. On the other hand, the second so-called *indistinguishability* property shows that each query made by an algorithm can only reveal very little information about the secret matching. The construction for Theorem 1.2 uses a simple bipartite structure but requires a more detailed analysis to optimize for the constant 0.584. The construction for Theorem 1.1, on the other hand, is based on a more sophisticated $(1/\epsilon)$-level tree structure which, at a high level, can be viewed as a novel tree extension of the path-alike construction from [26]. We discuss in more details about our lower bound proofs in Section 1.3.

On the positive side, when the stream is insertion only, we obtain an algorithm that achieves an approximation guarantee of $1-1/e-\epsilon$ with amortized query complexity $O(\log(k/\epsilon)/\epsilon^2)$:

Theorem 1.3. *Given any $\epsilon > 0$, there is a deterministic algorithm that achieves an approximation guarantee of $1 - 1/e - \epsilon$ for dynamic submodular maximization under cardinality constraint k over insertion-only streams. The amortized query complexity of the algorithm is $O(\log(k/\epsilon)/\epsilon^2)$.*

We also obtain a randomized algorithm that achieves an approximation guarantee of $1 - 1/e - \epsilon$ for the problem under a matroid constraint, with amortized query complexity $k^{\widetilde{O}(1/\epsilon^2)} \log n$.

Theorem 1.4. *Given any $\epsilon > 0$, there is a randomized algorithm that achieves an approximation guarantee of $1 - 1/e - \epsilon$ for dynamic submodular maximization under matroid constraints over insertion-only streams. The amortized query complexity of the algorithm is $k^{\widetilde{O}(1/\epsilon^2)} \log n$, where k is the rank of the matroid.*

Our algorithms are inspired by the classic greedy solution and we turn it into a dynamic algorithm via techniques involving lazy update, multilinear extension and accelerated continuous greedy, together with the introduction of a $O(1/\epsilon)$-pass prune-greedy algorithm that could be of independent interest. We elaborate about the intuition behind our algorithms in Section 1.3.

Remark 1.5. *The best offline algorithm for maximizing monotone submodular function with cardinality constraints requires $O(n \log(1/\epsilon))$ queries [13]. Our dynamic algorithm incurs only poly-logarithmic overheads. While for matroid constraints, even the state-of-art offline algorithm requires $O(n\sqrt{k})$ value queries [13]. Hence, for matroid constraints, our focus is to design dynamic algorithms with $\mathrm{poly}(k)$ amortized query complexity.*

We review related work and then give a technical overview of our results in Section 1.3.

1.2 Related Work

Dynamic submodular maximization has only been studied recently, and the work of [30, 34] are most relavant to us. The concurrent work of [34] and [30] give $(1/2 - \epsilon)$-approximation algorithm to the dynamic submodular maximization problem under cardinality constraints, with amortized query complexity $O(k^2 \log^2 n \cdot \epsilon^{-3})$ and $O(\log^8 n \cdot \epsilon^{-6})$ respectively.

Our work is also closely related to the streaming setting. For cardinality constraints, [2] give an $(1/2 - \epsilon)$-approximation algorithm in the streaming model, The approximiation is tight, Feldman et al. [26] prove $\Omega(\epsilon n/k^3)$ space is necessary for achieving $(1/2 + \epsilon)$ approximation. For matroid contraints, [16] give an 1/4-approximation algorithm and the approximation ratio is improved to 0.3178 by [27]. We remark the algorithm of [2] and [16] can be implemented in the insertion-only dynamic setting with the same approximation guarantee, and the amortized runing query complexity are $O(\epsilon^{-1} \log(k/\epsilon))$ and $O(k)$ respectively.

On lower bound side, our work make use of the symmetric gap techniques of [32, 39] and the weight scheduling of [26].

1.3 Technical Overview

We provide a streamlined technique overview of our approach.

A linear lower bound for 0.584-approximation. Our lower bound is based on the construction of a family of monotone submodular functions with the following properties. They share the same ground set V which is partitioned into $2m$ sets $V = A_1 \cup \cdots \cup A_m \cup B_1 \cup \cdots \cup B_m$ and the algorithm knows both V and the partition. Each A_i or B_j contains $O(k)$ elements to be specified later. What is hidden inside the function is a secret bijection $\pi : [m] \to [m]$, which is unknown to the algorithm, and we write the function as $\mathcal{F}_\pi : 2^V \to [0, 1]$. The three main properties we need about $\mathcal{F}$ are:

i) For any $j \in [m]$, there is an $S \subseteq A_{\pi(j)} \cup B_j$ of size k that achieves the optimal $\mathcal{F}_\pi(S) = 1$;
ii) **Large gap:** For any $j \in [m]$, every $S \subseteq A_1 \cup \cdots \cup A_m \cup B_j$ of size at most k has $\mathcal{F}_\pi(S) \leq \kappa$ for some (as small as possible) parameter $\kappa > 0$, unless $S \cap A_{\pi(j)} \neq \emptyset$; and
iii) **Indistinguishability:** Every query made by the algorithm reveals very little information about π. (Looking ahead, we show that each query on $\mathcal{F}_\pi$ is roughly equivalent to no more than $O(k)$ queries on π, each of the limited form as "whether $\pi(a)$ is equal b or not.")

With these properties in hand, we use the following simple dynamic stream of length $\Theta(mk)$ in our lower bound proof: $A_1, \ldots, A_m$ are inserted first and then we insert B_1, delete B_1, insert B_2, delete $B_2, \ldots$, until B_m is inserted and deleted. The large-gap property (ii) ensures that any algorithm with an approximation guarantee of κ must recover each entry $\pi(j)$ of the hidden π after each set

B_j is inserted.[1] The indistinguishability property (iii), on the other hand, shows that $\Omega(m^2/k)$ queries are needed to recover the hidden bijection π (by a standard calculation) and thus, the amortized query complexity is at least $\Omega(m/k^2) = \Omega(n/k^3)$ by plugging in the stream length $n = \Theta(mk)$. The main challenge is to find a construction that satisfies (i) and (iii) and at the same time satisfies (ii) with a parameter κ that is as small as possible (optimized to be 0.584 at the end).

We focus on minimizing κ in the large-gap property (ii) in this part, given that achieving the indistinguishability (iii) will become more challenging in the polynomial lower bound for $(0.5+\epsilon)$-approximation later and will be what we focus on there. Our first attempt is the following simple coverage function $\mathcal{F}$ (though our final construction will not be a coverage function). Let U be a large universe set (for intuition consider U to be arbitrarily large so that every expectation about random sets we draw holds exactly). For each $j \in [m]$ we randomly partition U into two parts $U_{j,1}$ and $U_{j,2}$ of the same size: the union of elements of $A_{\pi(j)}$ (which correspond to subsets of U) will be $U_{j,1}$ and the union of elements of B_j will be $U_{j,2}$. Each B_j is further partitioned into w groups $B_{j,1}, \ldots, B_{j,w}$, where w is a large enough constant, such that each $B_{j,\ell}$ has size $k/2$ and corresponds to an even $(k/2)$-way random partition of $U_{i,2}$. The same happens with $A_{\pi(j)}$. Now imagine that the algorithm does not know $\pi(j)$ and returns $S \subset A_i \cup B_j$ with $i \neq \pi(j)$ by picking $k/2$ elements randomly from A_i and $k/2$ elements randomly from B_j. Simple calculation shows that elements in S would together cover $0.5(1-1/e) + 0.25(1-1/e^2) \approx 0.53$, which is very close to our goal of 0.5, and splitting elements not evenly between A_i and B_j can only cover less.

However, the algorithm can deviate by (1) not picking elements randomly from A_i and B_j and furthermore, (2) picking from not just $A_i \cup B_j$ but also the much bigger set of $A_1 \cup \cdots \cup A_m$. For (1), if the algorithm manages to find out all $k/2$ elements of group $A_{i,\ell}$ and all $k/2$ elements of group $B_{j,\ell'}$ for some ℓ, ℓ', then together they would cover 3/4. For (2), the algorithm can pick one random element each from k different A_i's to cover $1-1/e$. In our final construction, we overcame (1) by applying a symmetric function constructed from [32] on top of the coverage function above. It guarantees that no algorithm (with a small number of queries) can ever pick a set of elements from any of A_i or B_j that is non-negligibly "*unbalanced*," by having noticeably more elements from any group $A_{i,\ell}$ or $B_{j,\ell}$ than the average among A_i or B_j. Accordingly, the large-gap property (ii) needed can be updated to be the following weaker version:

ii) **Large gap, updated:** For any $j \in [m]$, every "balanced" $S \subseteq A_1 \cup \cdots \cup A_m \cup B_j$ of size at most k has $\mathcal{F}_\pi(S) \leq \kappa$ for some parameter $\kappa > 0$, unless $S \cap A_{\pi(j)} \neq \emptyset$;

To address the challenge of (2), we use two parameters to introduce a different type of *unbalancedness*: First, U is no longer partitioned 50-50; instead $U_{j,1}$ is β-fraction and $U_{j,2}$ is $(1-\beta)$-fraction of U. Second, it is no longer the case that all groups $A_{i,\ell}$ and $B_{j,\ell}$ have size $k/2$; instead every $A_{i,\ell}$ has size αk and every $B_{j,\ell}$ has size $(1-\alpha)k$. With a detailed analysis we show that κ can be set to 0.584 in the updated version of the large-gap property (ii), when $\alpha = 0.56$ and $\beta = 0.42$.

A polynomial lower bound for $(1/2+\epsilon)$-approximation. Hard instances for our linear lower bound sketched earlier reduce at the end to a simple problem of recovering a hidden matching. To obtain a polynomial lower bound for $(1/2+\epsilon)$-approximation, we need to further extend the idea to amplify the (hardness of) approximation ratio by creating a depth-L tree structure with $L = 1/\epsilon$ to hide multiple bijections at every level. Let $m_1, \ldots, m_L$ be a sequence of positive integers: $m_L = 1$ but the other integers will be specified at the end. The ground set V is then defined using a depth-L tree T, in which each internal node at depth ℓ ($\ell \in [0 : L-1]$) has $m_{\ell+1}$ children. Let U_ℓ denote the set of nodes $u = (u_1, \ldots, u_\ell) \in [m_1] \times \cdots \times [m_\ell]$ at depth ℓ; its children are given by $(u,1), \ldots, (u, m_{\ell+1})$. The ground set is defined as $V = \cup_{u \in U_1 \cup \cdots \cup U_L} A_u$, where each A_u contains ϵk elements.

In the dynamic update stream, we perform a limited DFS walk on the tree, meaning that after reaching a node u, one only explores the first $d := n^\epsilon$ children $(u,1), \ldots, (u,d)$ of u (except when u is at depth $L-1$ in which case we just explore the only child of u). Let $U_L^* := [d]^{L-1} \times \{1\}$ be the set of leaves visited by the DFS walk. We design the stream in a way such that whenever a leaf u is reached, the current set of elements is given by

$$W_u := \bigcup_{\ell \in [L]} \bigcup_{i \in [m_\ell]} A_{u_1, \ldots, u_{\ell-1}, i},$$

i.e., the union of sets of nodes that are children of any node along the path from the root to u. Let $\mathcal{A}_u = A_{u_1} \cup \cdots \cup A_{u_1, \ldots, u_{L-1}}$. The first two properties of our $\mathcal{F} : 2^V \to [0,1]$ can now be stated:

i) Let $S = \mathcal{A}_u \cup A_u$. We have $|S| = k$ and $\mathcal{F}(S)$ achieves the optimal $\mathcal{F}(S) = 1$;
ii) **Large gap:** For any $S \subseteq W_u$ of size at most k, $\mathcal{F}(S) < 1/2 + \epsilon$ unless $S \cap \mathcal{A}_u \neq \emptyset$.

At a high level, the construction of our monotone submodular function $\mathcal{F}$ can be viewed as a novel tree extension of a path-alike construction from [26] (for one-way communication complexity of submodular maximization). In particular, we adopted a weight sequence $\{w_\ell\}$ from [26] to play the role of parameter β in our linear lower bound discussed earlier, in order to achieve the large-gap property (ii) above no matter how the S is spread among different levels of the tree.

Now $\mathcal{F}$ is only the base function that we use to construct the family of hard functions. To this end, we introduce the notion of a *shuffling* π of tree T, which consists of one bijection π_u at every internal node u of T to shuffle its children. Let $\mathcal{F}_\pi$ be the function obtained after applying a hidden shuffling π on $\mathcal{F}$. Under the same stream, at the time when a leaf $u \in U_L^*$ arrives, the current set of elements remains to be W_u but the $\mathcal{A}_u$ we care about becomes

$$\mathcal{A}_u^\pi := A_{\pi_\gamma(u_1)} \cup A_{u_1, \pi_{u_1}(u_2)} \cup \cdots \cup A_{u_1, \ldots, u_{L-2}, \pi_{u_1, \ldots, u_{L-2}}(u_{L-1})},$$

where π_γ denotes the hidden bijection at the root, π_{u_1} denotes the hidden bijection at the depth-1 node u_1, so on and so forth. The two properties (i) and (ii) hold after replacing $\mathcal{A}_u$ with $\mathcal{A}_u^\pi$. So finding $S \subseteq W_u$ of size at most k with $\mathcal{F}(S) \geq 1/2 + \epsilon$ requires the algorithm to (essentially) identify one of the edges along the path after shuffling, i.e., one of the entries $\pi_\epsilon(u_1), \pi_{u_1}(u_2), \ldots, \pi_{u_1, \ldots, u_{L-2}}$

[1] Note that we are a bit sloppy here given that the dynamic algorithm only needs to output a set S_j after B_j is inserted to overlap with $A_{\pi(j)}$. But given that $|S_j| \leq k$, this can be considered almost as good as knowing $\pi(j)$.

(u_{L-1}) in the hidden bijections. The challenge left is to establish the indistinguishability property:

iii) **Indistinguishability:** Every query made by the algorithm reveals very little information about π. (More formally, we show that each query on $\mathcal{F}_\pi$ is roughly equivalent to no more than $O(k^2)$ queries on π, each of the limited form as "whether $\pi_u(a)$ for some u is equal b."

With (iii) if an algorithm tries to win by finding out $\pi_{u_1,\ldots,u_{L-2}}(u_{L-1})$ every time a leaf u is reached then each hidden entry requires $\Omega(m_{L-1}/k^2)$ queries and thus, $\Omega(m_{L-1}d^{L-1}/k^2)$ queries in total. In general if an algorithm keeps working on $\pi_{u_1,\ldots,u_\ell}(u_{\ell+1})$, the total number of queries needed is $\Omega(m_{\ell+1}d^{\ell+1}/k^2)$. Setting $m_\ell = n^{(L-\ell+1)\epsilon}/(2k)$ for each ℓ leads to a lower bound of $\Omega(n^{1+\epsilon}/k^3)$ for the total query complexity in all cases, while maintaining the stream length to be n. (It is also not difficult to show that mixing effort among different levels does not help.)

Finally let's discuss how the indistinguishability property (iii) is implemented. The first trick is to cap the function and have the final function take the form as

$$\mathcal{F}^*_\pi(S) = \min\left\{\mathcal{F}_\pi(S) + \epsilon|S|/k, 1\right\}.$$

This simple modification changes the value of solutions S we care about slightly, no more than ϵ, given that $|S| \le k$, and at the same time guarantees no algorithm makes any query of size more than k/ϵ; otherwise the value returned is trivially 1 and reveals no information. This implies that each query S made by an algorithm can only involve no more than k/ϵ nodes in the tree; let U_S denote the set of such nodes. A final key observation is that the function $\mathcal{F}_\pi$ on S is uniquely determined by the structure of the tree formed by paths of nodes in U_S to the root after applying the shuffling π. The latter is uniquely determined by the depth of the lowest common ancestor of every pair of nodes in U_S after shuffling π, which in turn can be obtained by making at most one query to π of the form described in (iii) for each pair in U_S. The bound $O(k^2)$ follows given that $|U_S| \le k/\epsilon$.

Insertion-only streams: Submodular maximization with a cardinality constraint. Our starting point is the classic (offline) greedy approach, where in each round an element with the maximum marginal contribution is added to the solution set. We adapt the greedy algorithm to the dynamic setting. For simplicity, let's assume in the overview that a value OPT is given to the algorithm at the beginning and its goal is to maintain a set that achieves a $(1-1/e-\epsilon)$-approximation when the actual optimal value reaches OPT. While it is impossible to find the element with the maximum margin before having all of them in hands, we can instead choose one whose margin gives a moderate improvement, i.e., add any element that satisfies $f_S(e) \ge (\mathrm{OPT} - f(S))/k$, where S is the current solution set. This suffices to achieve the optimal $(1-1/e)$-approximation. A naive implementation, however, requires $O(k)$ amortized query complexity since the threshold $(\mathrm{OPT}-f(S))/k$ is updated every time a new element is added to solution set, and the algorithm needs to scan over all elements in the worst case. We circumvent this with the idea of *lazy update*. We divide the mariginal contribution into $O(1/\epsilon)$ buckets, with the i-th buckets containing marignal smaller than $i \cdot (\epsilon\,\mathrm{OPT}/k)$. We don't update the margin value every time the solution set is augmented. Instead, each time a new element is inserted, the algorithm only checks the bucket whose marginal is larger than the current threshold. One can show either a new element is added to S, or it is pushed down to the next level. The latter can happen at most $O(1/\epsilon)$ times, and after that, the element has negligible marginal contribution that can be ignored safely.

Insertion-only streams: Submodular maximization with a matroid constraint. We first provide a deterministic combinatorial algorithm that achieves a $(1/2-\epsilon)$-approximation. We build upon the previous idea, but with significant adaptations. Again, our goal is to simulate an offline greedy algorithm but this time, one cannot relax the condition and hope to augment the solution set whenever the margin has a moderate improvement. The analysis of the offline greedy algorithm relies crucially on picking the maximum margin element in each step. Our first idea is to find the element with approximate maximum margin and branch over all possibilities. In particular, we divide the marginal gain into $L = O(\epsilon^{-1}\log(k/\epsilon))$ many levels, where the ℓ-th level corresponds to $[(1+\epsilon)^{-\ell+1}\,\mathrm{OPT}, (1+\epsilon)^{-\ell}\,\mathrm{OPT}]$. When working on an insertion-only stream, we proceed to the $(\ell+1)$-th level only when there is no element in the ℓ-th level anymore. Of course, we still don't know when the ℓ-th level becomes empty so that we can move to $\ell+1$, but we can enumerate over all possibilities and guarantee that there exists one branch that fits the sequence of the offline approximate greedy algorithm. The caveat is that the total number of branch is $\binom{k}{L}$, which is quasi-polynomial in k. We further reduce this number by considering a pruned version of the offline approximate greedy algorithm, where the algorithm prunes extra element in each level and only keeps the majority. We prove the algorithm still guarantees a $(1/2-\epsilon)$-approximation and the total number of branch reduces to $k^{O(1/\epsilon)}$. It requires careful analysis to make the idea work, but the very high level intuition is that one needs to be more careful in the first few buckets (since the marginal is large) and less careful at the end. To amplify the approximation ratio to $(1-1/e-\epsilon)$, we make use of the multi-linear extension and use the accelerated continuous greedy framework [3]. In each iteration, we use the above combinatorial algorithm to find the direction of improvement of the multi-linear extension. We remark a similar amplifying procedure has been used in the previous work on adaptive submodular maximization [7, 17].

Organization. We begin by presenting the linear lower bound for 0.584-approximation under fully dynamic stream in Section 3. Section 4 is devoted to prove the polynomial lower bound on $(1/2+\epsilon)$-approximation. We then turn to the insertion-only stream and provide our algorithms and analysis. The algorithm for a cardinality constraint is presented in Section 5, and we provide an efficient $(1-1/e)$-approximation algorithm for a matroid constraint in Section 6. We discuss future research directions in Section 7. All missing proofs can be found in the full version [20].

2 PRELIMINARY

Submodular functions. Let V be a finite ground set. A function $f : 2^V \to \mathbb{R}$ is *submodular* if

$$f(S \cup T) + f(S \cap T) \leq f(S) + f(T)$$

for all pairs of sets $S, T \subseteq V$. We say f is *nonnegative* if $f(S) \geq 0$ for all $S \subseteq V$ and f is *monotone* if $f(S) \leq f(T)$ whenever $S \subseteq T \subseteq V$. We study submodular functions that are both nonnegative and monotone, and assume without loss of generality that $f(\emptyset) = 0$ (see Remark 2.2). We say an algorithm has query access to f if it can adaptively pick subsets $S \subseteq V$ to reveal the value of $f(S)$.

Given $f : 2^V \to \mathbb{R}$ and $S, T \subseteq V$, the marginal gain of adding e to S is defined as

$$f_S(T) := f(S \cup T) - f(S).$$

When $T = \{e\}$ is a singleton, we write $f_S(e)$ to denote $f_S(\{e\})$ for convenience. Monotonicity and submodularity can be defined equivalently using marginal gains: f is monotone iff $f_S(e) \geq 0$ for all $e \in V$ and $S \subseteq V$; f is submodular iff $f_S(e) \geq f_T(e)$ for any $e \notin T$ and $S \subseteq T \subseteq V$.

Dynamic submodular maximization under a cardinality constraint. Given V, a positive integer k and query access to a nonnegative, monotone submodular function $f : V \to \mathbb{R}$, the goal is to find a γ-approximate solution S_i of size at most k at the end of each round i when making a pass on a dynamic stream of insertions and deletions (the stream is *insertion-only* if no deletions are allowed). More formally, starting with $V_0 = \emptyset$, an element $e_i \in V$ is either inserted or deleted at the beginning of round $i = 1, \ldots,$ so that the current ground set V_i is set to be either $V_{i-1} \cup \{e_i\}$ if e_i is inserted or $V_i = V_{i-1} \setminus \{e_i\}$ if e_i is deleted. After this, the algorithm makes queries to f to find a *γ-approximate solution S_i of f with respect to V_i*. This means that $|S_i| \leq k$ and

$$f(S_i) \geq \gamma \, \mathrm{OPT}_i, \quad \text{where} \quad \mathrm{OPT}_i := \max_{T \subseteq V_i, |T| \leq k} f(T)$$

We emphasize that an algorithm remembers every query it has made so far. Thus results of queries made in previous rounds may help finding S_i in the current round.

We will consider algorithms that are both deterministic and randomized. We say a deterministic dynamic algorithm achieves an approximation guarantee of γ if given n and any stream of length n, it returns a γ-approximate solution S_i of the i-th round for every $i \in [n]$. We say a randomized dynamic algorithm achieves an approximation guarantee of γ if given n and any stream of length n, with probability at least $2/3$ it returns a γ-approximate solution S_i for every round $i \in [n]$ at the same time. We say an (deterministic or randomized) algorithm has *amortized* query complexity Q if the total number of queries it makes is no more than $n \cdot Q$.

Remark 2.1. *We discuss some details of the model behind our upper and lower bounds:*

1. *Our lower bounds hold even if the algorithm is given V and it is allowed to query during the i-th round any set S of elements that have appeared in V_j for some $j \leq i$; our algorithms only query sets in V_i and does not need to know V initially.*
2. *Our lower bounds hold even if the algorithm is given n, the stream length; our algorithms do not need to know n and meet the stated amortized bounds at the end of every round.*

Dynamic submodular maximization under a matroid constraint A set system $\mathcal{M} \subseteq 2^V$ is a *matroid* if it satisfies (i) $\emptyset \in \mathcal{M}$, (ii) the *downward closed* and (iii) *augmentation* properties. A set system $\mathcal{M}$ is downward closed if $T \in \mathcal{M}$ implies $S \in \mathcal{M}$ for all $S \subseteq T$. The augmentation property is that if $S, T \in \mathcal{M}$ and $|S| < |T|$, then there must be an element $e \in T \setminus S$ such that $S \cup \{a\} \in \mathcal{M}$. When $S \in \mathcal{M}$, we say S is *feasible* or *independent*. The rank of the matroid $\mathcal{M}$, denoted as $\mathrm{rank}(\mathcal{M})$, is the maximum size of an independent set in $\mathcal{M}$.

The setting of dynamic submodular maximization under a matroid constraint is similar. In addition to V and query access to f, the algorithm is given query access to a matroid $\mathcal{M}$ over V: it can pick any $S \subseteq V$ to query if $S \in \mathcal{M}$ or not. To goal is to find a γ-approximate solution S_i of f with respect to V_i at the end of every round. This mdeans that $S_i \subseteq V_i$, $S_i \in \mathcal{M}$ and

$$f(S_i) \geq \gamma \cdot \mathrm{OPT}_i, \quad \text{where} \quad \mathrm{OPT}_i := \max_{T \in \mathcal{M}, T \subseteq V_i} f(T).$$

When measuring the amortized query complexity, we count queries to both f and $\mathcal{M}$.

Remark 2.2. *In both problems, one can assume without loss of generality that $f(\emptyset) = 0$ (since any γ-approximate solution to g with respect to V_i, where $g(S) := f(S) - f(\emptyset)$, must be a γ-approximate solution to f with respect to V_i as well). We will make this assumption in the rest of the paper.*

The multilinear extention The multilinear extension $F : [0,1]^{|V|} \to \mathbb{R}^+$ of a function f maps a point $\mathbf{x} \in [0,1]^{|V|}$ to the expected value of a random set $S \sim \mathbf{x}$, i.e.

$$F(\mathbf{x}) = \sum_{S \subseteq V} \prod_{e \in S} x_e \prod_{e' \in V \setminus S} (1 - x_{e'}) f(S)$$

We write $F(\mathbf{x}) = \mathbb{E}_{S \sim \mathbf{x}}[f(\mathbf{x})]$ for simplicity. For any $\mathbf{x} \in [0,1]^{|V|}$, $\lambda \in [0,1]$, $S \subseteq V$, we write $F(\mathbf{x} + \lambda S)$ to denote $F(\mathbf{x}')$, where $x'_i = x_i$ if $i \notin S$ and $x'_i = \min\{x_i + \lambda, 1\}$ if $i \in S$.

For a continuous function $F : [0,1]^{|V|} \to \mathbb{R}$, we say it is monotone if $\frac{\partial F}{\partial x_i} \geq 0$ and it is submodular if $\frac{\partial^2 F}{\partial x_i \partial x_j} \leq 0$ for every i, j. When f is monotone and submodular, the multilinear extension F is also monotone and submodular.

3 A LINEAR LOWER BOUND FOR 0.584-APPROXIMATION

We restate the main theorem of this section:

Theorem 1.2. *There is a constant $C > 0$ with the following property. When $k \geq C \log n$, any randomized algorithm for dynamic submodular maximization under cardinality constraint k that obtains an approximation guarantee of 0.584 must have amortized query complexity at least $\Omega(n/k^3)$.*

3.1 Construction of the Symmetric Function

Let w be a positive integer and let $\epsilon > 0$ be a small constant to be fixed later. Given $x \in [0,1]^w$, we let $\overline{x} = \sum_{i=1}^{w} x_i / w$. Consider the following function f over $[0,1]^w$ and its symmetric version g:

$$f(x) := 1 - \prod_{i \in [w]} (1 - x_i) \quad \text{and} \quad g(x) := 1 - (1 - \overline{x})^w.$$

The following theorem is from [32]. We need to make some minor changes on the choice of parameters and include a proof in the full version of this paper for completeness.

Theorem 3.1 ([32]). *Given any positive integer w and $\epsilon > 0$, let $\gamma = \gamma(w, \epsilon) := w^{-1}\exp(-4w^6/\epsilon)$. There is a monotone submodular function $\widehat{f} : [0,1]^w \to [0,1]$ with the following two properties:*

(1) Whenever $\max_{i,j\in[w]} |x_i - x_j| \le \gamma$, we have $\widehat{f}(x) = g(x)$;

(2) For any $x \in [0,1]^w$, we have $f(x) - \epsilon \le \widehat{f}(x) \le f(x)$.

Let m be a positive integer. We use $\widehat{f}$ from Theorem 3.1 to construct a function $\widehat{F} : [0,1]^{mw} \to \mathbb{R}$ as follows: Writing $x \in [0,1]^{mw}$ as $x = (x_1, \ldots, x_s)$ and $x_i = (x_{i,1}, \ldots, x_{i,w}) \in [0,1]^w$, we have

$$\widehat{F}(x) := 1 - \prod_{i\in[m]} (1 - \widehat{f}(x_i)). \tag{1}$$

We also define its symmetric version $G : [0,1]^{mw} \to \mathbb{R}^+$ as

$$G(x) := 1 - \prod_{i\in[m]} (1 - g(x_i)) = 1 - \prod_{i\in[m]} (1 - \overline{x_i})^w.$$

Here are some basic properties we need about $\widehat{F}$. Their proofs can be found in the full version of the paper.

Lemma 3.2. *$\widehat{F}$ satisfies the following properties:*

- *$\widehat{F}$ is monotone, submodular and satisfies $\widehat{F}(x) \in [0,1]$ for all $x \in [0,1]^{mw}$.*
- *$\widehat{F}(x) = G(x)$ when $x \in [0,1]^{mw}$ satisfies*
$$\max_{i\in[m],j,j'\in[w]} |x_{i,j} - x_{i,j'}| \le \gamma$$
and in this case, $\widehat{F}(x)$ depends on $\overline{x_1}, \ldots, \overline{x_m}$ only.
- *$\widehat{F}(x) \ge 1 - \epsilon$ when x satisfies $x_{i,j} = 1$ for some $i \in [m]$ and $j \in [w]$,*

3.2 Construction of the Hard Functions $\mathcal{F}_{c,\pi}$

We now present the construction of the family of hard functions that will be used in the proof.

Choice of parameters. Let n be the length of the dynamic stream. Let $\epsilon > 0$ be a constant. Let α and β be two constants in $(0,1)$ that we fix at the end of the proof.[2] Let

$$w = 10\left(\frac{1}{\alpha\epsilon} + \frac{1}{(1-\alpha)\epsilon}\right) \quad \text{and} \quad \gamma = w^{-1}\exp\left(-\frac{4w^6}{\epsilon}\right)$$

be two constants. Let k be the cardinality constraint parameter that is at most $n^{1/3}$ (otherwise the lower bound $\Omega(n/k^3)$ in the main theorem becomes trivial) and satisfies

$$k \ge \frac{10}{\epsilon\gamma\alpha^2(1-\alpha)^2} \cdot \log n.$$

Finally let m be such that $n = (2-\alpha)mkw$ so we have $m = \Omega(n^{2/3})$.

The ground set V. We start with the definition of V, where

$$V = A \cup B, \quad A = \bigcup_{i=1}^{m} A_i \quad \text{and} \quad B = \bigcup_{i=1}^{m} B_i,$$

where each A_i has αkw elements and each B_j has $(1-\alpha)kw$ elements and they are pairwise disjoint.

The function $\mathcal{F}_{c,\pi}$. Let $c : V \to [w]$. We say c is a *proper w-coloring* of V if each $A_{i,j}$, the set of elements in A_i with color $j \in [w]$, has size αk and each $B_{i,j}$, the set of elements in B_i with color j, has size $(1-\alpha)k$. (So $A_{i,1}, \ldots, A_{i,w}$ form an even partition of A_i and $B_{i,1}, \ldots, B_{i,w}$ form an even partition of B_i.) Let $\pi : [m] \to [m]$ be a bijection which we will view as matching $A_{\pi(i)}$ with B_i for each $i \in [m]$. Given any proper w-coloring c of V and any bijection $\pi : [m] \to [m]$, we define a function $\mathcal{F}_{c,\pi} : 2^V \to \mathbb{R}$ as follows.

For any $S \subseteq V$, let $y^S \in [0,1]^{mw}$ and $z^S \in [0,1]^{mw}$ be

$$y^S_{i,j} = \frac{|S \cap A_{i,j}|}{\alpha k} \quad \text{and} \quad z^S_{i,j} = \frac{|S \cap B_{i,j}|}{(1-\alpha)k}.$$

For any any $I \subseteq [m]$, let $x^{S,I,\pi} \in [0,1]^{mw}$ be

$$x^{S,I,\pi}_{i,j} = \begin{cases} y^S_{\pi(i),j} & \text{if } i \in I \\ z^S_{i,j} & \text{if } i \notin I \end{cases}.$$

Finally we define $\mathcal{F}_{c,\pi} : 2^V \to \mathbb{R}$: For any $S \subseteq V$,

$$\mathcal{F}_{c,\pi}(S) = \min\left\{ \sum_{I\subseteq[m]} \beta^{|I|}(1-\beta)^{m-|I|} \cdot \widehat{F}(x^{S,I,\pi}) + \frac{\epsilon}{k}|S|,\ 1 \right\}. \tag{2}$$

We state some basic properties of the function $\mathcal{F}_{c,\pi}$.

Lemma 3.3. *For any c and π, $\mathcal{F}_{c,\pi}$ satisfies the following two properties:*

(1) $\mathcal{F}_{c,\pi}$ is monotone, submodular and $\mathcal{F}_{c,\pi}(S) \in [0,1]$ for any $S \subseteq V$.

(2) When $|S| \ge k/\epsilon$, $\mathcal{F}_{c,\pi}(S) = 1$.

(3) For any $i \in [m]$ and $j \in [w]$, we have $|A_{\pi(i),j} \cup B_{i,j}| = k$ and $\mathcal{F}_{c,\pi}(A_{\pi(i),j} \cup B_{i,j}) \ge 1 - \epsilon$.

Proof. We start with the first claim. By Lemma 3.2, $\widehat{F}$ is monotone and submodular. Hence, it is easy to see that for each $I \subseteq [m]$, $\widehat{F}(x^{S,I,\pi})$ is monotone and submodular in S. As addition and the min operation keep both submodularity and monotonity, $\mathcal{F}_{c,\pi}$ is both monotone and submodular.

For the second claim, since $\widehat{F}$ is nonnegative,

$$\mathcal{F}_{c,\pi}(S) \ge \min\{\epsilon|S|/k, 1\} = 1,$$

when $|S| \ge k/\epsilon$.

For the last claim, note that $|A_{\pi(i),j}| = \alpha k$, $|B_{i,j}| = (1-\alpha)k$ and $A_{\pi(i),j} \cap B_{i,j} = \emptyset$. Therefore, $|A_{\pi(i),j} \cup B_{i,j}| = k$. Let $S = A_{\pi(i),j} \cup B_{i,j}$. We have $x^{S,I,\pi}_{i,j} = 1$ for all $I \subseteq [m]$ since $y^S_{\pi(i),j} = z^S_{i,j} = 1$. By Lemma 3.2, $\widehat{F}(x^{S,I,\pi}) \ge 1 - \epsilon$ for all I and thus,

$$\begin{aligned} \mathcal{F}(S) &= \min\left\{ \sum_{I\subseteq[m]} \beta^{|I|}(1-\beta)^{m-|I|} \cdot \widehat{F}(x^{S,I,\pi}) + \frac{\epsilon}{k}|S|, 1 \right\} \\ &\ge \min\left\{ (1-\epsilon) + \frac{\epsilon}{k}|S|, 1 \right\} \ge 1 - \epsilon. \end{aligned}$$

This finishes the proof of the lemma. □

For any subset $S \subseteq V$, we say S is *balanced* with respect to c if

$$\max_{\substack{i\in[m]\\ j,j'\in[w]}} \left|y^S_{i,j} - y^S_{i,j'}\right| \le \gamma \quad \text{and} \quad \max_{\substack{i\in[m]\\ j,j'\in[w]}} \left|z^S_{i,j} - z^S_{i,j'}\right| \le \gamma.$$

[2] Looking ahead, we will choose α and β to minimize the quantity $Q(\alpha,\beta)$ discussed in Lemma 3.7; we will set them to be $\alpha = 0.56$ and $\beta = 0.42$, respectively.

Define $\mathcal{G}_\pi : 2^V \to \mathbb{R}$ as

$$\mathcal{G}_\pi(S) = \min\left\{ \sum_{I\subseteq[m]} \beta^{|I|}(1-\beta)^{m-|I|}\cdot G(x^{S,I,\pi}) + \frac{\epsilon}{k}|S|, 1 \right\}. \tag{3}$$

Recall the function $G(x)$ only depends on $\overline{x_i}$, $i \in [m]$. As a result, $\mathcal{G}_\pi(S)$ only depends on $|S\cap A_i|$ and $|S\cap B_i|$, $i\in[m]$. The following Lemma is a direct consequence of Lemma 3.2:

Lemma 3.4. *When $S \subseteq V$ is balanced with respect to c, we have $\mathcal{F}_{c,\pi}(S) = \mathcal{G}_\pi(S)$.*

Given any $S \subseteq V$, the next lemma captures the minimal information needed to evaluate $\mathcal{G}_\pi(S)$ without full knowledge of c, π.

Lemma 3.5. *Let $S \subseteq V$ and let $\pi, \pi' : [m] \to [m]$ be two bijections. Then we have $\mathcal{G}_\pi(S) = \mathcal{G}_{\pi'}(S)$ when the following condition holds: For each $i \in [m]$, we have $\pi(i) = \pi'(i)$ if 1) $S \cap B_i \neq \emptyset$ and 2) either $S \cap A_{\pi(i)} \neq \emptyset$ or $S \cap A_{\pi'(i)} \neq \emptyset$.*

Remark 3.6. *Let $S \subseteq V$. If (1) S is known to be balanced with respect to c and (2) we are given*

$$\Big\{(i,\pi(i)) : i \in [m], S\cap B_i \neq \emptyset \text{ and } S\cap A_{\pi(i)} \neq \emptyset\Big\}, \tag{4}$$

then one can evaluate $\mathcal{F}_{c,\pi}(S)$ without more information about c and π. To see this, (1) implies that it suffices to evaluate $\mathcal{G}_\pi(S)$. Lemma 3.5 implies that $\mathcal{G}_\pi(S)$ is uniquely determined given (4).

We present proofs of Lemma 3.5 and the following lemma in the full version. Lemma 3.7 is where we need to choose the two parameters α and β carefully to minimize the constant 0.584.

Lemma 3.7. *Fix any $i^* \in [m]$. Let $S \subseteq A \cup B_{i^*}$ be a set such that S is balanced with respect to c and $S \cap A_{\pi(i^*)} = \emptyset$. Then we have $\mathcal{F}_{c,\pi}(S) \le 0.5839 + 3\epsilon$.*

3.3 Lower Bound for Dynamic Submodular Maximization

Hard streams. The ground set is V, which is known to the algorithm; the monotone submodular function $\mathcal{F}^* : 2^V \to \mathbb{R}$ is $\mathcal{F}_{c,\pi}$, where c is a proper w-coloring of V and $\pi : [m] \to [m]$ is a bijection, both of which are unknown to the algorithm. In the stream we first insert all elements of A, which takes αmkw insertions. We then divide the rest of the stream into m batches: For the t-th batch, we first insert all elements of B_t and then delete them. So the total length is $n = (2-\alpha)mkw$. This is the only stream we will use in the proof but the function is determined by the unknown c and π.

We will focus on the $0.1m$ rounds when elements of B_t, for each $t = 1, \dots, 0.1m$, has just been inserted. To ease the presentation we consider the following dynamic problem that consists of $0.1m$ stages. During the t-th stage, an algorithm can query $\mathcal{F}^* : 2^V \to \mathbb{R}$ about any $S \subseteq A \cup B_1 \cup \cdots \cup B_t$ and can choose to start the next stage at any time by outputting a set $S_t \subseteq A \cup B_t$ of size at most k. The algorithm succeeds after $0.1m$ stages if $\mathcal{F}^*(S_t) > 0.5839 + 3\epsilon$ for every $t \in [0.1m]$. We prove that any randomized algorithm that succeeds with probability at least 2/3 must have total query complexity of at least $\Omega(n^2/k^3)$. It then follows from Lemma 3.3 (part 3) that any randomized algorithm for dynamic submodular maximization with an approximation guarantee of $0.5839 + 4\epsilon$ must have amortized query complexity $\Omega(n/k^3)$.

Proof of Theorem 1.2. Let ALG be a randomized algorithm for the $0.1m$-stage problem described above that succeeds with probability at least 2/3. By Lemma 3.3, we may assume that ALG never queries a set of size more than k/ϵ. To bound the query complexity of ALG we consider the following simple $0.1m$-stage game.

In the game there is a hidden proper w-coloring c of V and a hidden bijection $\pi : [m] \to [m]$. The game similarly consists of $0.1m$ stages. During each round of stage t, the algorithm can make a query by either (1) picking a subset S of V of size at most k/ϵ or (2) picking a number $i \in [m]$. In case (1) the c-oracle returns "balanced" or "unbalanced," as whether S is balanced with respect to c or not; when receiving "unbalanced" the algorithm wins the whole game. In case (2) the π-oracle returns "matched" or "not matched," as whether $\pi(t) = i$ or not; after receiving "matched," the algorithm can choose to proceed to the next stage, and it wins the game if it passes all $0.1m$ stages.

We show that any randomized algorithm that wins the game with probability at least 2/3 must use at least $\Omega(m^2)$ queries. To this end, we consider the distribution of $(\mathbf{c}, \boldsymbol{\pi})$ where $\mathbf{c}$ and $\boldsymbol{\pi}$ are drawn uniformly and independently, and show that any deterministic algorithm that wins the game with probability at least 2/3 must use $\Omega(m^2)$ queries.

Showing this distributional $\Omega(m^2)$ lower bound for each subgame is easy. For the subgame of finding an unbalanced set with respect to $\mathbf{c}$, we have for any set S of size at most k/ϵ that

$$\begin{aligned}\Pr\left[y^S_{i,j} - y^S_{i,j'} > \gamma\right] &= \Pr\left[|S\cap A_{i,j}| - |S\cap A_{i,j'}| > \gamma\alpha k\right] \\ &\le \exp\left(-\frac{\epsilon\gamma^2\alpha^2k^2}{2k}\right) \le \frac{1}{n^5}, \quad \text{and}\end{aligned}$$

$$\begin{aligned}\Pr\left[z^S_{i,j} - z^S_{i,j'} > \gamma\right] &= \Pr\left[|S\cap B_{i,j}| - |S\cap B_{i,j'}| > \gamma(1-\alpha)k\right] \\ &\le \exp\left(-\frac{\epsilon\gamma^2(1-\alpha)^2k^2}{2k}\right) \le \frac{1}{n^5}\end{aligned}$$

for any fixed i and $j \neq j'$, given our choice of k. Therefore, any deterministic algorithm that finds an unbalanced set with probability at least 1/3 requires $\Omega(n^5)$ queries. For the subgame about $\boldsymbol{\pi}$, at the beginning of each stage t, any of the remaining $m-(t-1)$ indices (other than $\pi(1), \dots, \pi(t-1)$) is equally likely to be $\pi(t)$. Given $m-(t-1) \ge 0.9m$, it takes $\Omega(m)$ queries to pass each of the $0.1m$ stages and thus, any deterministic algorithm that wins the subgame about $\boldsymbol{\pi}$ with probability at least 1/3 requires $\Omega(m^2)$ queries.

Back to the original game, if a deterministic algorithm $\mathcal{A}$ can win with probability at least 2/3, then it can either win the first or the second subgame with probability at least 1/3. Assuming for example it is the latter case, we get a randomized algorithm for winning the second subgame over $\boldsymbol{\pi}$ with probability at least 1/3 by first drawing $\mathbf{c}$ and then simulating $\mathcal{A}$ on the original game. Given that the number of queries is at most that of $\mathcal{A}$, we have that query complexity of $\mathcal{A}$ is $\Omega(m^2)$.

To finish the proof we show how to use ALG to play the game. Let c be the hidden coloring and π be the hidden bijection. We simulate the execution of ALG on $\mathcal{F}^* = \mathcal{F}_{c,\pi}$ as follows:

(1) During the first stage (of both the dynamic problem of maximizing $\mathcal{F}^*$ over $A \cup B_1$ and the game), letting

$S \subseteq A \cup B_1$ of size at most k/ϵ be any query made by ALG, we make one query S on the c-oracle (to see if S is balanced or not) and make one query i on the π-oracle for each $i \in [m]$ with $S \cap A_i \neq \emptyset$ (to see if $\pi(1) = i$). If S is not balanced, we already won the game; otherwise, we know $\mathcal{F}^*(S) = \mathcal{G}_\pi(S)$ and the latter can be computed (see Remark 3.6) using information returned by the π-oracle, so that we can continue the simulation of ALG. When ALG decides to output $S_1 \subseteq A \cup B_1$ of size at most k, we query S_1 on the c-oracle and query each i on the π-oracle with $S_1 \cap A_i \neq \emptyset$. If S_1 is unbalanced, we won the game; if $\pi(1) = i$ for some i queried, we move to stage 2 (in both the dynamic problem and the game); if S_1 is balanced and $\pi(1) \neq i$ for any $i : S_1 \cap A_i \neq \emptyset$, it follows from Lemma 3.7 that ALG has failed so we terminate the simulation and fail the game.

(2) During the t-th stage (of both the dynamic problem and the game), we similarly simulate each query $S \subseteq A \cup B_1 \cup \cdots \cup B_t$ of ALG. The only difference is that, given that we have passed the first $(t-1)$ stages of the game, we already know $\pi(1), \ldots, \pi(t-1)$ and thus, information returned by the π-oracle would be enough for us to evaluate $\mathcal{F}^*(S)$. When ALG returns S_t, we query S_t and each i with $S_t \cap A_i \neq \emptyset$, and act according to results similarly.

To summarize, the simulation has three possible outcomes: (1) we won the game because an unbalanced set has been found; (2) we won the game because we have passed all $0.1m$ stages; or (3) ALG fails to find S_t with $\mathcal{F}_{c,\pi}(S_t) \geq 0.5839 + 3\epsilon$ for some t. Given that (3) only happens with probability at most $1/3$, we obtain an algorithm for the game that succeeds with probability at least $2/3$ and thus, must use $\Omega(m^2)$ queries. To finish the proof, we note that if ALG has total query complexity q then the algorithm we obtain for the game has total query complexity at most

$$q \cdot \left(1 + \frac{k}{\epsilon}\right) + 0.1m \cdot (1 + k),$$

which implies $q = \Omega(m^2/k)$ and thus, the amortized complexity of ALG is $\Omega(m/k^2) = \Omega(n/k^3)$. Taking $\epsilon = 3 \times 10^{-5}$, we get the lower bound on approximation ratio. □

4 A POLYNOMIAL LOWER BOUND FOR $1/2 + \epsilon$ APPROXIMATION

We restate the main theorem of this section:

Theorem 1.1. *For any constant $\epsilon > 0$, there is a constant $C_\epsilon > 0$ with the following property. When $k \geq C_\epsilon$, any randomized algorithm that achieves an approximation ratio of $1/2 + \epsilon$ for dynamic submodular maximization under cardinality constraint k requires amortized query complexity $n^{\widetilde{\Omega}(\epsilon)}/k^3$.*

4.1 The Construction

Let $\epsilon > 0$ be a positive constant. We assume that both $1/\epsilon$ and ϵk are positive integers. Our goal is to show that any randomized algorithm for dynamic submodular maximization under a cardinality constraint of k with approximation $1/2 + \widetilde{O}(\epsilon)$ must have amortized query complexity $\Omega(n^\epsilon/k^3)$.

Algorithm 1 Sample

1: **Input:** A node v of T at depth $\ell \in [0 : L]$
2: Let p_ℓ be given as below:

$$p_\ell = \frac{w_\ell}{1 - \sum_{i=0}^{\ell-1} w_i} \in [0, 1].$$

3: With probability p_ℓ, return v. ▷ Since $p_L = 1$, this always happen when $\ell = L$
4: Otherwise, run Sample(v_i) independently on each child v_i of v in the tree and return the union

We start with the construction of a monotone submodular function $\mathcal{F} : V \to \mathbb{R}$. The family of functions used in our lower bound proof will be obtained from $\mathcal{F}$ by carefully shuffling elements of V. Let $L = 1/\epsilon \in \mathbb{N}$ and let $m = (m_1, \ldots, m_L)$ be a tuple of positive integers to be fixed later, with m_L set to be 1. Let T be a tree of depth L, where the root is at depth 0 and its leaves are at depth L. Each internal node of T at depth $\ell \in [0 : L-1]$ has $m_{\ell+1}$ children; the number of nodes at depth $\ell \in [L]$ is $m_1 \cdots m_\ell$. We use γ to refer to the root of T and write $U_0 = \{\gamma\}$; for each $\ell \in [L]$, we use $V_\ell = [m_1] \times \cdots \times [m_\ell]$ to refer to nodes of T at depth ℓ. So the set of nodes is $U_0 \cup U_1 \cup \cdots \cup U_L$ and whenever we refer to a node u of T at depth ℓ, it should be considered as a tuple $(u_1, \ldots, u_\ell) \in U_\ell$. Children of $u \in U_\ell$, $\ell \leq L-1$, are given by $(u_1, \ldots, u_\ell, 1), \ldots, (u_1, \ldots, u_\ell, m_{\ell+1})$.

The ground set V of $\mathcal{F}$ is defined as follows. For each node $u \in U := U_1 \cup \cdots \cup U_L$, we introduce a set $A_u = \{a_{u,1}, \ldots, a_{u,w}\}$ of $w := \epsilon k$ new elements. We define V as the union of A_u for all $u \in U$. To construct $\mathcal{F} : 2^V \to \mathbb{R}$, we will utilize a weight sequence $\{w_\ell\}_{\ell \in [0:L]}$ from [26] to define a probability distribution $\mathcal{D}$ over subsets of nodes of T. We specify the sequence later in Lemma 4.5; for now it suffices to know that $w_0 = 0$, w_ℓ's are nonnegative, and they sum to 1.

We define the distribution of $\mathcal{D}$ over 2^U as follows. Drawing a sample from $\mathbf{R} \sim \mathcal{D}$ can be done by calling the recursive procedure Sample in Algorithm 1 on the root of T. Informally, starting with $\mathbf{R} = \emptyset$, Sample performs a DFS walk on T. Whenever reaching a node v at depth ℓ, it adds v to $\mathbf{R}$ and does not explore any of its children with probability p_ℓ (see Algorithm 1; note that $p_0 = 0$ and $p_L = 1$); otherwise, it continues the DFS walk to visit each of its children. Because $p_0 = 0$, the root is never included in the set and thus, $\mathbf{R}$ is always a subset of U.

We need the following properties about $\mathbf{R}$; the proof can be found in the full version of the paper.

Lemma 4.1. *Let $\mathbf{R} \sim \mathcal{D}$, we need have the following properties*

- *For any node u of T at depth ℓ, the probability of $u \in \mathbf{R}$ with $\mathbf{R} \sim \mathcal{D}$ is w_ℓ.*
- *For any root-to-leaf path of T, every R in the support of $\mathcal{D}$ has exactly one node in the path.*

Let $P_1 \subseteq U$ and $P_2 \subseteq U$ be two subsets of nodes of T and let τ be a bijection from P_1 to P_2 such that (1) u and $\tau(u)$ are at the same depth for every $u \in P_1$; and (2) for any $u, v \in P_1$, the depth of the lowest common ancestor (LCA) of u and v is the same as that of $\tau(u)$ and $\tau(v)$. The following lemma follows from how the procedure Sample works; the proof can be found in the full version:

Lemma 4.2. *The distribution of $P_2 \cap \mathbf{R}$, $\mathbf{R} \sim \mathcal{D}$, is distributed the same as first drawing $\mathbf{R} \sim \mathcal{D}$, taking $P_1 \cap \mathbf{R}$ and applying τ on $P_1 \cap \mathbf{R}$.*

We use $\mathcal{D}$ to define $\mathcal{F}$. Given $S \subseteq V$, we define $x^S \in [0,1]^U$ as the vector indexed by $u \in U$ with

$$x^S_u = \frac{|S \cap A_u|}{\epsilon k} \in [0,1].$$

For any R in the support of $\mathcal{D}$ we define $g_R : [0,1]^U \to [0,1]$ and $\mathcal{G} : [0,1]^U \to [0,1]$ as

$$g_R(x) := 1 - \prod_{u \in R} (1 - x_u) \quad \text{and} \quad \mathcal{G}(x) = \mathop{\mathbb{E}}_{\mathbf{R} \sim \mathcal{D}} \left[g_{\mathbf{R}}(x) \right]. \tag{5}$$

We are now ready to define $\mathcal{F}$ over 2^V as

$$\mathcal{F}(S) = \min \left\{ \mathcal{G}(x^S) + \frac{\epsilon}{k}|S|, 1 \right\}, \quad \text{for any set } S \subseteq V. \tag{6}$$

We state some basic properties of $\mathcal{F}$.

Lemma 4.3. *$\mathcal{F}$ is monotone submodular and $\mathcal{F}(S) \in [0,1]$ for any $S \subseteq V$. Moreover, $\mathcal{F}(S) = 1$ whenever $|S| \geq k/\epsilon$.*

For each leaf $u \in U_L$, we define two important subsets of V. The first set, $\mathcal{A}_u$, is the union of A's along the path from u's parent to the root (not including the root):

$$\mathcal{A}_u := A_{u_1} \cup A_{u_1,u_2} \cup \cdots \cup A_{u_1,\ldots,u_{L-1}}.$$

The other set W_u is the union of A's of nodes that are children of nodes along the root-to-u path:

$$W_u := \bigcup_{\ell \in [L]} \bigcup_{j \in [m_\ell]} A_{u_1,\ldots,u_{\ell-1},j}.$$

The following lemma shows that $\mathcal{F}(\mathcal{A}_u \cup A_u)$ is large:

Lemma 4.4. *For any leaf $u \in U_L$, we have $|\mathcal{A}_u \cup A_u| = k$ and $\mathcal{F}(\mathcal{A}_u \cup A_u) = 1$.*

We would like to show in the next lemma that any set $S \subseteq W_u$ that has size at most k and does not overlap with $\mathcal{A}_u$ must have a small $\mathcal{F}(S)$. For this we need to specify the weight sequence $\{w_\ell\}$ that we will use from [26] (recall that $w_0 = 0$).

Definition 4.5 (Weight sequence)**.** *Define $\{\delta_\ell\}_{\ell \in [L]}, \{a_\ell\}_{\ell \in [L]}$ and the weight sequence $\{w_\ell\}_{\ell \in [L]}$ inductively as follow. We set $\delta_L = 1$ and for $\ell = L-1, L-2, \ldots, 1$, let*

$$\delta_\ell = 1 + \left(\frac{1 + \sqrt{1 + 4/\delta_{\ell+1}}}{2} \right) \cdot \delta_{\ell+1}$$

and

$$a_\ell = \prod_{i=1}^{\ell-1} \left(\frac{\delta_i - 1}{\delta_{i+1}} \right) = \prod_{i=1}^{\ell-1} \frac{1}{1 - 1/\delta_i},$$

the weight sequence $\{w_\ell\}_{\ell \in [L]}$ is defined as $w_\ell = a_\ell / \sum_{\ell \in [L]} a_\ell$.

The proof of the following lemma is adapted from Lemma 5.3 in [26] with some generalizations, and the proof can be found in the full version of the paper.

Lemma 4.6. *For any leaf $u \in U_L$ and $S \subseteq W_u$ with $|S| \leq k$ and $S \cap \mathcal{A}_u = \emptyset$, we have*

$$\mathcal{F}(S) \leq 0.5 + O\left(\epsilon \log^2(1/\epsilon)\right).$$

As it will become clear soon at the beginning of the next subsection, our goal behind the family of hard streams is to have the dynamic algorithm solve repeatedly the question of finding an $S \subseteq W_u$ with $|S| \leq k$ and $S \cap \mathcal{A}_u \neq \emptyset$, for a large number of leafs u of T. These questions are, however, only interesting after we shuffle nodes of T in the fashion to be described next.

A *shuffling* π of T consists of a bijection $\pi_u : [m_{\ell+1}] \to [m_{\ell+1}]$ for every $u \in U_\ell$, $\ell \in [0 : L-1]$. We use π to shuffle each node $u \in U$ to $\pi(u)$ as follows: $\pi(\gamma) = \gamma$; for each u of depth $\ell \in [L]$, set

$$\pi(u) = \left(u_1, \ldots, u_{\ell-1}, \pi_{u_1,\ldots,u_{\ell-1}}(u_\ell)\right).$$

A shuffling π induces a bijection $\rho_\pi : V \to V$: For each element $a_{u,i}$ for some $u \in U$ and $i \in [w]$, we set $\rho_\pi(a_{u,i}) = a_{\pi^{-1}(u),i}$. Finally we define for each shuffling π of T,

$$\mathcal{F}_\pi(S) := \mathcal{F}\left(\rho_\pi(S)\right).$$

It is clear that Lemma 4.3 holds for $\mathcal{F}_\pi$ for any shuffling π.

Given a leaf $u \in U_L$ of T and a shuffling π, let

$$\mathcal{A}^\pi_u := A_{u_1,\ldots,u_{L-2},\pi_{u_1,\ldots,u_{L-2}}(u_{L-1})} \cup \cdots A_{u_1,\pi_{u_1}(u_2)} \cup A_{\pi_\epsilon(u_1)}.$$

We get the following corollary of Lemma 4.4 and Lemma 4.6:

Corollary 4.7. *For any shuffling π of T, we have*

- *For any leaf $u \in U_L$, we have $|\mathcal{A}^\pi_u \cup A_u| = k$ and $\mathcal{F}_\pi(\mathcal{A}^\pi_u \cup A_u) = 1$.*
- *For any leaf u and $S \subseteq W_u$ with $|S| \leq k$ and $S \cap \mathcal{A}^\pi_u = \emptyset$,*

$$\mathcal{F}_\pi(S) \leq 0.5 + O\left(\epsilon \log^2(1/\epsilon)\right). \tag{7}$$

We need a corollary of Lemma 4.2. Similar to Lemma 3.5, it captures the minimal information needed about π to evaluate $\mathcal{F}_\pi$ at a given set $S \subseteq V$:

Corollary 4.8. *Let $S \subseteq V$ and let π, π' be two shufflings of T. We have $\mathcal{F}_\pi(S) = \mathcal{F}_{\pi'}(S)$ when the following condition holds: For every two nodes u, v of T such that $S \cap A_u$ and $S \cap A_v$ are nonempty, the LCA of $\pi^{-1}(u)$ and $\pi^{-1}(v)$ is at the same depth as the LCA of $\pi'^{-1}(u)$ and $\pi'^{-1}(v)$, both in T.*

Remark 4.9. *To evaluate $\mathcal{F}_\pi(S)$, it suffices to know the LCA of $\pi^{-1}(u)$ and $\pi^{-1}(v)$ for every u, v with $S \cap A_u \neq \emptyset$ and $S \cap A_v \neq \emptyset$. The LCA of $\pi^{-1}(u)$ and $\pi^{-1}(v)$ can be determined as follows.*

(1) First consider the case when u is a prefix of v or v is a prefix of u. Let $u = (u_1, \ldots, u_\ell)$ and $v = (u_1, \ldots, u_\ell, v_{\ell+1}, \ldots)$. (The case when v is a prefix is similar.) Then the depth of LCA of $\pi(u)$ and $\pi^{-1}(v)$ is either ℓ if $\pi_{u_1,\ldots,u_{\ell-1}}(u_\ell) = u_\ell$, or $\ell - 1$ otherwise.

(2) Assume that $u = (u_1, \ldots, u_\ell, u_{\ell+1}, \ldots)$ and $v = (v_1, \ldots, v_\ell, v_{\ell+1}, \ldots)$ with $u_1 = v_1, \ldots, u_\ell = v_\ell$ but $u_{\ell+1} \neq v_{\ell+1}$. We have three subcases. If both u and v have length strictly longer than $\ell + 1$, then the depth of LCA of of $\pi^{-1}(u)$ and $\pi^{-1}(v)$ is ℓ. If both u and v have length $\ell + 1$, then the depth of LCA of $\pi^{-1}(u)$ and $\pi^{-1}(v)$ is also ℓ. (So in these two subcases we we do not need to know anything about π.) Finally, if u has length $\ell + 1$ and v has length longer than $\ell + 1$, then the depth of LCA of $\pi^{-1}(u)$ and $\pi^{-1}(v)$ is $\ell + 1$ if $\pi_{v_1,\ldots,v_\ell}(v_{\ell+1}) = u_{\ell+1}$, and is ℓ otherwise. The case when u has length longer than $\ell + 1$ and v has length $\ell + 1$ is similar.

To summarize, to determine the depth we only need to know whether a particular entry of π is equal to a certain value or not. Moreover, the entry is either $\pi_{u_1,\ldots,u_{\ell-1}}(u_\ell)$ or $\pi_{v_1,\ldots,v_{\ell-1}}(v_\ell)$ for some ℓ.

4.2 Lower Bound for Dynamic Submodular Maximization

Choices of parameters. Let $L = 1/\epsilon$ be a positive integer. Let k be a positive integer such that $k^2 \le n^\epsilon$ (as otherwise the lower bound we aim for becomes trivial) and $w = \epsilon k$ is a positive integer. Let n be the length of dynamic streams and $d = n^\epsilon$. Set $m_L = 1$ and

$$m_\ell = \frac{n^{(L-\ell+1)\epsilon}}{2k}, \quad \text{for each } \ell \in [L-1].$$

Hard streams. The ground set is V, which is known to the algorithm; the monotone submodular function $\mathcal{F}^* : 2^V \to \mathbb{R}$ is $\mathcal{F}_\pi$, where π is a shuffling of T, which is unknown to the algorithm. The stream is constructed by running Tranverse in Algorithm 2, and is independent of π. It can be viewed as a DFS over the tree T, starting at the root, except that every time it reaches a node v, it inserts all children of v but only explores the first d children, and deletes all children of v at the end of the exploration.

We first bound the total number of operations by n. For each $\ell \in [0 : L-2]$, Tranverse visits d^ℓ nodes and creates $2m_{\ell+1} \cdot w$ operations for each of them. Tranverse visits d^{L-1} nodes at depth $L-1$ and creates $2w$ operations for each of them. Hence, the total number of operations is

$$\sum_{\ell=0}^{L-2} d^\ell \cdot m_{\ell+1} \cdot 2w + d^{L-1} \cdot 2w \le \sum_{\ell=0}^{L-2} \epsilon n + \epsilon n \le n.$$

To gain some intuition behind the stream, we note that leafs of T that appear in the stream are exactly those in $U_L^* := [d]^{L-1} \times \{1\}$, and they appear in the stream under the lexicographical order (which we will denote by $\prec$). For each such leaf $u \in U_L^*$, at the time when the set A_u was inserted, the current set of elements is W_u. Inspired by Corollary 4.7 we will consider the following simplified d^{L-1}-stage dynamic problem, where stages are indexed using leaves in U_L^* under the lexicographical order. During the u-th stage, an algorithm can query $\mathcal{F}^*$ about any subset $S \subseteq \cup_{u' \in U_L^* : u' \preceq u} W_{u'}$ and can choose to start the next stage at any time by returning an $S_u \subseteq W_u$ of size at most k. We say an algorithm succeeds if $\mathcal{F}^*(S_u) \ge 0.5 + \widetilde{O}(\epsilon)$ as on the RHS of (7) for every stage. We show below that any randomized algorithm that succeeds with probability at least $2/3$ must have total query complexity $\Omega(n^{1+\epsilon}/k^3)$. It follows from Corollary 4.7 that any randomized algorithm for dynamic submodular maximization with an approximation guarantee of $0.5 + \widetilde{\Omega}(\epsilon)$ must have amortized query complexity $\Omega(n^\epsilon/k^3)$.

Proof of Theorem 1.1. Let $\boldsymbol{\pi}$ be a shuffling of T drawn uniformly at random (i.e., every bijection in $\boldsymbol{\pi}$ is drawn independently and uniformly). Consider any deterministic algorithm ALG that succeeds with probability $2/3$ on the d^{L-1}-stage dynamic problem described above with $\mathcal{F}^* = \mathcal{F}_{\boldsymbol{\pi}}$. By Lemma 4.3 we assume without loss of generality that ALG only queries sets of size at most k/ϵ. When ALG succeeds on $\mathcal{F}^* = \mathcal{F}_\pi$ for some shuffling π, we have from Corollary 4.7 that the S_u it outputs during the u-th stage must

Algorithm 2 Tranverse

```
Input: A node u of T at depth ℓ ∈ [0 : L − 1]
if ℓ = L − 1 then
    Insert all elements of A_{u_1,...,u_{L−1},1} and then delete them
else
    Insert all elements in A_{u_1,···,u_ℓ,i} for each i ∈ [m_{ℓ+1}]
    for i from 1 to d do
        Call Tranverse on (u_1, ..., u_ℓ, i)
    end for
    Delete all elements in A_{u_1,···,u_ℓ,i} for each i ∈ [m_{ℓ+1}]
end if
```

satisfy $S_u \cap \mathcal{A}_u^\pi \ne \emptyset$, which implies (using the definition of $\mathcal{A}_u^\pi$)

$$S_u \cap A_{u_1,\ldots,u_\ell,\pi_{u_1,\ldots,u_\ell}(u_{\ell+1})} \ne \emptyset \tag{8}$$

for some $\ell \in [0 : L-2]$. By an averaging argument, we have that there exists an $\ell \in [0 : L-2]$ such that with probability at least $2/(3L)$ over $\boldsymbol{\pi}$, at least $(1/L)$-fraction of S_u returned by ALG satisfy (8) for this ℓ. Fix such an ℓ and this inspires us to introduce the following simple game.

In the game there are d^ℓ hidden bijections $\pi_{v_1,\ldots,v_\ell} : [m_{\ell+1}] \to [m_{\ell+1}]$, for each $v_1, \ldots, v_\ell \in [d]$. The game consists of $d^{\ell+1}$ stages; each stage is indexed by a $v = (v_1, \ldots, v_{\ell+1}) \in [d]^{\ell+1}$ and ordered by the lexicographical order. During the v-th stage of the game, an algorithm can send a number $i \in [m_{\ell+1}]$ to the oracle and the latter reveals if $\pi_{v_1,\ldots,v_\ell}(v_{\ell+1}) = i$. We say the algorithm wins the v-th stage if it queries an i that matches $\pi_{v_1,\ldots,v_\ell}(v_{\ell+1})$ during the v-th stage. At any time it can choose to give up and move forward to the next stage, in which case $\pi_{v_1,\ldots,v_\ell}(v_{\ell+1})$ is revealed to the algorithm. We say an algorithm succeeds if it wins at least $(1/L)$-fraction of the $d^{\ell+1}$ stages.

We prove the following lower bound for this game in the full version of the paper.

Claim 4.10. *When the hidden bijections are drawn independently and uniformly, any deterministic algorithm that succeeds with probability at least $2/(3L)$ has total query complexity $\Omega(m_{\ell+1} d^{\ell+1})$.*

To finish the proof, we show that ALG can be used to play the game as follows:

(1) We start by drawing a random bijection for every node in the tree T except for those at $(v_1, \ldots, v_\ell) \in [d]^\ell$. Let π be the shuffling when they are combined with hidden bijections $\pi_{v_1,\ldots,v_\ell}$ in the game. We simulate ALG on $\mathcal{F}_\pi$ over the stream Tranverse and maintain the following invariant. During the v-th stage of the game, with $v = (v_1, \ldots, v_{\ell+1}) \in [d]^{\ell+1}$, we simulate ALG through its stages for leaves $u \in U_L^*$ that have v as a prefix and assume that we already know $\pi_{w_1,\ldots,w_\ell}(w_{\ell+1})$ for all $w = (w_1, \ldots, w_{\ell+1}) \prec v$ and $w \in [d]^{\ell+1}$.
(2) During the u-th stage of the simulation of ALG for some leaf $u \in U_L^*$, we are in the v-th stage of the game with $v = (u_1, \ldots, u_{\ell+1}) \in [d]^{\ell+1}$. For each query $S \subseteq \cup_{u' \in U_L^* : u' \preceq u} W_{u'}$ (of size at most k/ϵ) made by ALG, it follows from Remark 4.9 that, to evaluate S at $\mathcal{F}_\pi$, we only need to know the depth of LCA of $\pi^{-1}(u'')$ and $\pi^{-1}(v'')$ in

T for no more than $(k/\epsilon)^2$ many pairs of u'', v'' with $S \cap A_{u''} \neq \emptyset$ and $S \cap A_{v''} \neq \emptyset$. For each such pair, it follows from Remark 4.9 again that either we know the answer already or we need to compare $\pi_{u''_1,\ldots,u''_\ell}(u''_{\ell+1})$ or $\pi_{v''_1,\ldots,v''_\ell}(v''_{\ell+1})$ with a certain value. Given that $u'' \preceq v$ and $v'' \preceq v$, either the answer is already known or we just need to make a query to the game oracle. So we only need to make at most $(k/\epsilon)^2$ queries to continue the simulation of ALG. At the end of the u-th stage of ALG, let S_u be the set that ALG returns. For every i such that $S_u \cap A_{u_1,\ldots,u_\ell,i} \neq \emptyset$ (note that there are at most k such i given that $|S| \leq k$), we query i on the game oracle. Note that we would have won the v-th stage of the game by now if (8) holds for S_u. We then continue to simulate ALG on the next stage of the dynamic problem. If the next stage of ALG is about a new leaf u with $v \prec (u_1, \ldots, u_{\ell+1})$, then we also move to the next stage in the game.

It is clear from the simulation that for any π, if ALG running on $\mathcal{F}_\pi$ satisfies (8) for at least $(1/L)$-fraction of leaves in U^*_L, then we win the game when π is the shuffling we get by combining our own random samples at the beginning with hidden bijections in the game. Using the promise about ALG at the beginning, we get a randomized algorithm that succeeds in the game with probability at least $2/(3L)$ when the hidden bijections are drawn independently and uniformly at random. On the other hand, if the query complexity of ALG is q, then our simulation uses

$$q \cdot \left(\frac{k}{\epsilon}\right)^2 + d^{L-1} \cdot k$$

Combining with Claim 4.10 we have $q = \Omega(n^{1+\epsilon}/k^3)$. □

5 INSERTION-ONLY STREAMS UNDER A CARDINALITY CONSTRAINT

We consider insertion-only streams and give a deterministic $(1 - 1/e - \epsilon)$-approximation algorithm with $O(\log(k/\epsilon)/\epsilon^2)$ amortized query complexity. As discussed in Remark 2.1, our algorithm does not need to know the ground set V or the number of rounds n at the beginning.

Theorem 1.3. *Given any $\epsilon > 0$, there is a deterministic algorithm that achieves an approximation guarantee of $1 - 1/e - \epsilon$ for dynamic submodular maximization under cardinality constraint k over insertion-only streams. The amortized query complexity of the algorithm is $O(\log(k/\epsilon)/\epsilon^2)$.*

To prove Theorem 1.3, we give a deterministic algorithm (pseudocode in Algo 3) with the following performance guarantees. We follow standard arguments to finish the proof of Theorem 1.3 and the proof can be found in the full version of the paper.

Lemma 5.1. *There is a deterministic algorithm that satisfies the following performance guarantees. Given a positive integer k, $\epsilon > 0$ and $\mathrm{OPT} > 0$, the algorithm runs on an insertion-only stream and outputs a set $S_t \subseteq V_t$ of size at most k at the end of each round t such that (1) $S_1 \subseteq S_2 \subseteq \cdots$ and (2) when $\mathrm{OPT}_t \geq \mathrm{OPT}$ for the first time, S_t must satisfy $f(S_t) \geq (1 - 1/e - \epsilon)\,\mathrm{OPT}$. The amortized query complexity of the algorithm is $O(1/\epsilon)$.*

Algorithm 3 Dynamic submodular maximization with a cardinality constraint.

```
1:  procedure Initialize(k, ε, OPT)
2:      Set S = ∅, Δ = ε OPT /k and B_ℓ = ∅ for ℓ = 0, 1, ..., ⌊1/ε⌋
3:  end procedure
4:
5:  procedure Insert(e)
6:      if f_S(e) ≥ (OPT − f(S))/k − Δ and |S| < k then
7:          Update S ← S ∪ {e} and call Revoke
8:      else
9:          Update B_ℓ ← B_ℓ ∪ {e} with ℓ = ⌊f_S(e)/Δ⌋
10:         (Note that ℓ ≤ ⌊1/ε⌋ given that f_S(e) < OPT /k)
11:     end if
12: end procedure
13:
14: procedure Revoke
15:     Let r = ⌊(OPT − f(S))/(kΔ)⌋.
16:     if |S| < k and there exists an index ℓ ≥ r with B_ℓ ≠ ∅ then
17:         Let ℓ be any such index and let e' be any element in B_ℓ.
18:         if f_S(e') ≥ (OPT − f(S))/k − Δ then
19:             Update S ← S ∪ {e'} and B_ℓ ← B_ℓ \ {e'}
20:         else
21:             Update B_ℓ ← B_ℓ \ {e'} and B_ℓ' ← B_ℓ' ∪ {e'} with
                    ℓ' = ⌊f_S(e')/Δ⌋
22:             (Note that 0 ≤ ℓ' < r ≤ ℓ)
23:         end if
24:         Go to Line 15.
25:     end if
26: end procedure
```

6 INSERTION-ONLY STREAM: EFFICIENT ALGORITHM FOR MATROID CONSTAINTS

We present an efficient $(1 - 1/e - \epsilon)$-approximation algorithm under the general matroid constraint. We first give a $(1/2 - \epsilon)$-approximate deterministic combinatorial algorithm (Section 6.1), we then embed it into the accelerated continuous greedy framework of [3] to achieve $(1 - 1/e - \epsilon)$ approximation (Section 6.2).

6.1 The Combinatorial Algorithm

Given k as the rank of the matroid $\mathcal{M}$, we define L, R and $\mathcal{A}$ as

$$L = \left\lceil \frac{\log(k/\epsilon)}{\epsilon} \right\rceil, \quad R = \left\lceil \frac{2\log(k/\epsilon)}{\epsilon^2} \right\rceil$$

and

$$\mathcal{A} = \left\{ (a_1, \cdots, a_L) \in \mathbb{Z}^L_{\geq 0} : \sum_{\ell \in [L]} a_\ell \leq R \right\}.$$

We note the size of $\mathcal{A}$ can be upper bounded by

$$|\mathcal{A}| = \sum_{d=0}^{R} \binom{d+L-1}{L-1} \leq (R+1) \cdot \binom{R+L-1}{L-1} \leq (R+1)\left(\frac{2eR}{L}\right)^L = k^{\widetilde{O}(1/\epsilon)}. \tag{9}$$

Both of our deterministic algorithm and randomized algorithm use a deterministic subroutine called Prune-greedy described in

Algorithm 4 Prune-greedy

procedure Initialize(k, OPT, a, h, $\mathcal{M}$) ▷ $a \in \mathcal{A}$
  Initialize $S \leftarrow \emptyset$ and $c_\ell \leftarrow a_\ell \Delta$ for each $\ell \in [L]$, where
$$\Delta := \frac{\epsilon^2 \,\mathrm{OPT}}{\log(k/\epsilon)}.$$
  Let ℓ^* be the smallest $\ell \in [L]$ with $c_\ell > 0$
end procedure

procedure Insert(e)
  if $S \cup e \in \mathcal{M}$ and $h_S(e) \geq (1+\epsilon)^{-\ell^*}$ OPT **then**
    Update $c_{\ell^*} \leftarrow c_{\ell^*} - h_S(e)$ and $S \leftarrow S \cup \{e\}$
    if $c_{\ell^*} \leq 0$ **then**
      Call Revoke
    end if
  end if
end procedure

procedure Revoke
  Update ℓ^* to be the smallest ℓ with $c_\ell > 0$
  Terminate and return S **if** no such ℓ exists
  for each e_i inserted so far (in the order of insertions) **do**
    if $S \cup e_i \in \mathcal{M}$ and $h_S(e_i) \geq (1+\epsilon)^{-\ell^*}$ OPT **then**
      Update $c_{\ell^*} \leftarrow c_{\ell^*} - h_S(e_i)$ and $S \leftarrow S \cup \{e_i\}$,
    end if
  end for
  Go to line 16 **if** $c^*_\ell \leq 0$
end procedure

Algorithm 4. The inputs of Prune-greedy include k as the rank of the underlying matroid $\mathcal{M}$, a positive number OPT, a tuple $a \in \mathcal{A}$, as well as query access to both f and $\mathcal{M}$. When running on an insertion-only stream, Prune-greedy may decide to terminate at the end of a round and output a set S. A complication due to the application of this subroutine in the randomized algorithm is that we will give it query access to a perturbed version of f: We say $h : V \to \mathbb{R}$ is a κ-*close* of f if $h(S) = f(S) \pm \kappa$ for every $S \subseteq V$.

We state its performance guarantees in the following lemma:

Lemma 6.1. *There is a deterministic algorithm that, given a positive integer k, $\mathrm{OPT} > 0$, $a \in \mathcal{A}$ and query access to a matroid $\mathcal{M}$ over V of rank k and a function $h : V \to \mathbb{R}$ that is κ-close to a nonnegative and monotone submodular function $f : V \to \mathbb{R}$, where $\kappa = \epsilon^3 \,\mathrm{OPT}/k$. The algorithm runs on insertion-only streams with amortized query complexity $O(L)$ and has the following performance guarantee. Given any insertion-only stream $e_1, \ldots, e_t$ such that $\mathrm{OPT} \leq \mathrm{OPT}_t \leq (1+\epsilon)\,\mathrm{OPT}$, there exists an $a^* \in \mathcal{A}$ such that when given OPT and a^* as input, the algorithm terminates before the end of round t and outputs a feasible set $S \subseteq V_t$ that satisfies*

$$f(S) \geq (1 - O(\epsilon)) f_S(O), \quad \text{for any set } O \text{ such that } O \in \mathcal{M} \text{ and } O \subseteq V_t.$$

Proof. The algorithm is described in Algorithm 4.

Let $e_1, \ldots, e_t$ be the stream with $\mathrm{OPT} \leq \mathrm{OPT}_t \leq (1+\epsilon)\,\mathrm{OPT}$. To specify the $a^* \in \mathcal{A}$ in the statement of the lemma, we consider the following L-pass greedy algorithm. The algorithm maintains a set $T \in \mathcal{M}$. It starts with $T = \emptyset$ and updates $T \to T \cup T_\ell$ at the end of the ℓ-th pass (so we have $T = T_1 \cup \cdots \cup T_\ell$ at the end of the ℓ-th pass). During the ℓ-th pass, we set $S_\ell = \emptyset$ and go through $e_1, \ldots, e_t$. For each e_i, the algorithm checks if $T \cup S_\ell \cup \{e_i\} \in \mathcal{M}$ and $h_{T \cup S_\ell}(e_i) \geq (1+\epsilon)^{-(\ell-1)}$ OPT. If so, e_i is added to S_ℓ. At the end of the ℓ-th pass, we do *not* add all elements S_ℓ to T. Instead, we further prune S_ℓ to get T_ℓ: Let $e_{i_1}, e_{i_2}, \ldots$ be elements added to S_ℓ during the ℓ-th pass. T_ℓ is set to be $\{e_{i_1}, \ldots, e_{i_j}\}$ such that j is the smallest integer such that

$$h_{T_1 \cup \cdots \cup T_{\ell-1}}(T_\ell) \geq \Delta \left\lfloor \frac{h_{T_1 \cup \cdots \cup T_{\ell-1}}(S_\ell)}{\Delta} \right\rfloor,$$

where $T_\ell = \emptyset$ when the RHS above is 0. This finishes the ℓ-th pass and the algorithm updates T with $T \cup T_\ell$. Let $(S_1, \ldots, S_L)$ and $(T_1, \ldots, T_L)$ be the two sequences of sets obtained from this L-pass algorithm. Let $a^* \in \mathbb{Z}^L_{\geq 0}$ be defined as $a^*_\ell = \lfloor h_{T_1 \cup \cdots \cup T_{\ell-1}}(S_\ell)/\Delta \rfloor$ for each $\ell \in [L]$. It is easy to see that

$$\begin{aligned}\Delta \sum_{\ell \in [L]} a^*_\ell &= \Delta \sum_{\ell \in [L]} \left\lfloor \frac{h_{T_1 \cup \cdots \cup T_{\ell-1}}(S_\ell)}{\Delta} \right\rfloor \\ &\leq \sum_{\ell=1}^{L} h_{T_1 \cup \cdots \cup T_{\ell-1}}(T_\ell) \leq f(T) + 2\kappa < 2\,\mathrm{OPT}.\end{aligned}$$

Hence $\sum_{\ell \in [L]} a^*_\ell \leq R$ and thus, $a^* \in \mathcal{A}$. The following lemma connects Prune-Greedy with this L-pass greedy algorithm:

Lemma 6.2. *Suppose that Prune-Greedy is given a^* at the beginning, then it terminates before the end of the t-th round and outputs exactly $T = T_1 \cup \cdots \cup T_L$.*

Given Lemma 6.2, it suffices to prove that

$$f(T) \geq (1 - O(\epsilon)) f_T(O), \quad \text{for every feasible set } O \subseteq V_t.$$

Fix an $O \in \mathcal{M}$ and $O \subseteq V_t$. The following Lemma is a folklore.

Lemma 6.3. *Let $\mathcal{M}$ be a matroid and $T \in \mathcal{M}$ with $T = T_1 \cup \cdots \cup T_L$ such that $T_1, \ldots, T_L$ are pairwise disjoint. Then any $O \in \mathcal{M}$ can be partitioned into pairwise disjoint $O_1, \ldots, O_L$ such that*

(1) If $|O| \geq |T|$ then $|O_i| = |T_i|$ for all $i < L$; If $|O| < |T|$, letting ℓ be the smallest integer such that $|O| \leq \sum_{i \leq \ell} |T_\ell|$, then $|O_i| = |T_i|$ for all $i < \ell$ and $|O_\ell| = |O| - \sum_{i<\ell} |O_i|$. (Note that we always have $|O_i| \leq |T_i|$ except for $i = L$.)
(2) For all $i < j$, $T_i \cap O_j = \emptyset$ and for every $i < L$,
$$T_1 \cup \cdots \cup T_i \cup O_{i+1} \cup \cdots \cup O_L \in \mathcal{M}.$$

Recall $S_1, \ldots, S_L$ from the L-pass greedy algorithm. We have for every $\ell \in [L]$,
$$T_1 \cup \cdots \cup T_{\ell-1} \cup S_\ell \in \mathcal{M}.$$
For the analysis we partition O into pairwise disjoint sets $E_0, E_1 \ldots, E_{L-2}, P_1, \ldots, P_L$ as follows.

(1) $P_1 = O_1$;
(2) For each $\ell \geq 2$ and each $o \in O_\ell$, we consider two cases. If $T_1 \cup \cdots \cup T_{\ell-2} \cup S_{\ell-1} \cup \{o\} \in \mathcal{M}$ then we have $o \in P_\ell$; otherwise, we have $o \in E_r$ where $r \geq 0$ is the largest integer such that $T_1 \cup \cdots \cup T_{r-1} \cup S_r \cup \{o\} \in \mathcal{M}$ and $T_1 \cup \cdots \cup T_r \cup S_{r+1} \cup \{o\} \notin \mathcal{M}$. Note that $r \leq \ell - 2$ and such an $r \geq 0$ always exists given that the condition when $r = 0$ is just that $\{o\} \in \mathcal{M}$.

The following claim about the size of E_ℓ follows from the definition and its proof can be found in the full version.

Claim 6.4. $|E_\ell| \le |S_{\ell+1} \setminus T_{\ell+1}|$.

Now we have

$$\begin{aligned} f_T(O) &\le \sum_{\ell\in[L]} f_{T_1\cup\cdots\cup T_{\ell-1}}(O_\ell) \\ &\le f(O_1) + \sum_{\ell\in[2:L]} \sum_{r\in[0:\ell-2]} f_{T_1\cup\cdots\cup T_{\ell-1}}(O_\ell \cap E_r) \\ &\quad + \sum_{\ell\in[2:L]} f_{T_1\cup\cdots\cup T_{\ell-1}}(P_\ell), \end{aligned} \tag{10}$$

where both steps follow from the submodularity of f.

We bound the first and last terms as follow.

Lemma 6.5. *We have*

$$f(O_1) + \sum_{\ell\in[2:L]} f_{T_1\cup\cdots\cup T_{\ell-1}}(P_\ell) \le (1+\epsilon) f(T) + 6\epsilon \,\mathrm{OPT}\,.$$

We bound the second term of Eq. (10) as follow.

Lemma 6.6. *We have*

$$\sum_{\ell\in[2:L]} \sum_{r\in[0:\ell-2]} f_{T_1\cup\cdots\cup T_{\ell-1}}(O_\ell \cap E_r) \le 8\epsilon \,\mathrm{OPT}\,.$$

Proofs of the two lemmas above can be found in the full version.

Combining Lemma 6.5, Lemma 6.6 and Eq. (10), we have $f_T(O) \le (1+\epsilon) f(T) + 11\epsilon \,\mathrm{OPT}$. Since this holds for all feasible sets, it holds for the set O_t. Taking a linear combination, we have

$$\begin{aligned} f(T) &\ge \frac{1}{1+\epsilon}\Big((1-11\epsilon) f_T(O) + 11\epsilon f_T(O_t) - 11\epsilon \,\mathrm{OPT}\Big) \\ &\ge \frac{1}{1+\epsilon}\Big((1-11\epsilon) f_T(O) - 11\epsilon f(T)\Big). \end{aligned}$$

Rearranging the term, we get the desired.

Finally, we bound the amortized query complexity of the algorithm. We charge the two queries made in the evaluation of $h_S(e)$ (line 7 or 19) to e and show that the number of queries charged to e is at most $O(L)$. To see this, we note that e is charged twice when it is just inserted. Every time e is charged during Revoke, either it is added to S so that it is never charged again, or its marginal contribution is small and won't be queried in the ℓ-th level later. Hence, e has been queried for at most $O(L)$ times. We conclude the proof here. □

By standard argument, we conclude

Theorem 6.7. *Given any matroid $\mathcal{M}$, for any $\epsilon > 0$, there is a combinatorial algorithm that maintains a feasible set S with $(1/2-\epsilon)$-approximation at each iteration. Moreover, the amortized number of queries per update is $k^{\widetilde{O}(1/\epsilon)}$.*

6.2 Amplification via Accelerated Continuous Greedy

We amplify the approximation ratio of the combinatorial algorithm via the accelerated continuous greedy framework [3]. Let $m = O(1/\epsilon)$ and

$$D = \{\mathrm{OPT}, (1+\epsilon)^{-1}\,\mathrm{OPT}, \cdots, (1+\epsilon)^{-\lceil 4\log(1/\epsilon)/\epsilon\rceil}\,\mathrm{OPT}\} \cup \{0\}.$$

Algorithm 5 Amplification via accelerated continuous greedy

1: **Input:** $\mathbf{d} \in D^m, \mathbf{a} \in \mathcal{A}^m$
2: Initialize $\mathbf{x} \leftarrow \mathbf{0}$
3: **for** $\tau = 1, 2, \cdots, m$ **do**
4: Define $g(S) = F(\mathbf{x} + \frac{1}{m}S) - F(\mathbf{x})$ for all $S \subseteq V$
5: Invoke Prune-greedy$(d_\tau, a_\tau, \mathrm{g})$, and wait until it returns a solution S_τ
6: $\mathbf{x} \leftarrow \mathbf{x} + \frac{1}{m}S_\tau$
7: **end for**

We run a separate branch for each $\mathbf{d} \in D^m$ and $\mathbf{a} \in \mathcal{A}^m$. Intuitively, $d_\tau \in D$ should be seen as an estimate on the progress of optimal solution in the τ-th iteration, and $a_\tau \in \mathcal{A}$ is a guess on the greedy sequence. Our algorithm segragates answer from all branches and outputs the one with the maximum value. In order to return an integral solution, the algorithm rounds the fractional solution via the swap rounding approach [18]. The algorithm description is presented in Algorithm 5.

We need the following lemma in our analysis. The proof idea follows from [3] and it appears in [7]. It has some minor difference with previous work and we provide a proof in the full version for completeness.

Lemma 6.8. *Let $\mathrm{OPT} \le \mathrm{OPT}_t \le (1+\epsilon)\,\mathrm{OPT}$. Suppose in each iteration of Algorithm 5, Prune-greedy returns a set S that satisfies*

$$g(S) \ge (1 - O(\epsilon)) \sum_{i\in[L]} g_{S\setminus(O_i\cup\cdots\cup O_L)}(O_i) - \epsilon^2\,\mathrm{OPT}, \tag{11}$$

for some partition $O = O_1 \cup \cdots \cup O_L$ of O and some partition $S = S_1 \cup \cdots \cup S_L$ of S such that $\forall i \in [L], S_1 \cup \cdots S_i \cup O_{i+1} \cup \cdots \cup O_L \in \mathcal{M}$ and $S_i \cap (O_{i+1} \cup \cdots \cup O_L) = \emptyset$. Then the final solution $\mathbf{x}$ satisfies

$$F(\mathbf{x}) \ge (1 - 1/e - O(\epsilon))\,\mathrm{OPT}\,.$$

We use Lemma 6.8 to finish the proof of Theorem 1.4 in the full version of the paper.

7 CONCLUSIONS

We study the power and limitations of dynamic algorithms for submodular maximization. On the lower bound side, we prove a polynomial lower bound on the amortized query complexity for achieving a $(1/2+\epsilon)$-approximation, together with a linear lower bound for 0.584-approximation, under fully dynamic streams with insertions and deletions. On the algorithmic side, we develop efficient $(1-1/e)$-approximation algorithms for insertion-only streams under both cardinality and matroid constraints. There are many interesting directions for further investigations:

- Many submodular functions important in practice can be accessed in white box models instead of query models, e.g., the MAX-k coverage problem, influence maxmization (see [37] for an example). Can ideas in this paper be extended to obtain upper/lower bounds on amortized time complexity for these problems?
- Can we extend results (algorithm or hardness) to non-monotone submodular maximization? As far as we know, there is no known constant-factor approximation algorithm with poly(k) amortized query complexity for the

non-monotone setting under fully dynamic streams. How does the dynamic model compare to the streaming model [1] under this setting?
- For matroid constraints, can one improve the query complexity to $O(\sqrt{k})$ over insertion-only streams? Also, for fully dynamic streams, there is no known constant-factor approximation algorithm with poly(k) amortized queries for matroid constraints.

ACKNOWLEDGEMENT

Research of X.C. and B.P. were supported in part by NSF grants CCF-1703925, IIS-1838154, CCF-2106429, CCF-2107187, CCF-1763970, CCF-1910700 and DMS-2134059. We would like to thank Paul Liu for pointing out a mistake in an earlier version of the paper.

REFERENCES

[1] Naor Alaluf, Alina Ene, Moran Feldman, Huy L. Nguyen, and Andrew Suh. 2020. Optimal Streaming Algorithms for Submodular Maximization with Cardinality Constraints. In *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*. 6:1–6:19.
[2] Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. 2014. Streaming submodular maximization: Massive data summarization on the fly. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 671–680.
[3] Ashwinkumar Badanidiyuru and Jan Vondrák. 2014. Fast algorithms for maximizing submodular functions. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*. SIAM, 1497–1514.
[4] Maria-Florina Balcan and Nicholas JA Harvey. 2011. Learning submodular functions. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*. 793–802.
[5] Eric Balkanski, Aviad Rubinstein, and Yaron Singer. 2017. The limitations of optimization from samples. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*. 1016–1027.
[6] Eric Balkanski, Aviad Rubinstein, and Yaron Singer. 2019. An exponential speedup in parallel running time for submodular maximization without loss in approximation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 283–302.
[7] Eric Balkanski, Aviad Rubinstein, and Yaron Singer. 2019. An optimal approximation for submodular maximization under a matroid constraint in the adaptive complexity model. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*. 66–77.
[8] Eric Balkanski and Yaron Singer. 2018. The adaptive complexity of maximizing a submodular function. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*. 1138–1151.
[9] Eric Balkanski and Yaron Singer. 2018. Approximation guarantees for adaptive sampling. In *International Conference on Machine Learning*. PMLR, 384–393.
[10] Rafael Barbosa, Alina Ene, Huy Nguyen, and Justin Ward. 2015. The power of randomization: Distributed submodular maximization on massive datasets. In *International Conference on Machine Learning*. PMLR, 1236–1244.
[11] Rafael da Ponte Barbosa, Alina Ene, Huy L Nguyen, and Justin Ward. 2016. A new framework for distributed submodular maximization. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*. Ieee, 645–654.
[12] Niv Buchbinder and Moran Feldman. 2018. Submodular Functions Maximization Problems.
[13] Niv Buchbinder, Moran Feldman, and Roy Schwartz. 2017. Comparing apples and oranges: Query trade-off in submodular maximization. *Mathematics of Operations Research* 42, 2 (2017), 308–329.
[14] Niv Buchbinder, Moran Feldman, Joseph Seffi, and Roy Schwartz. 2015. A tight linear time (1/2)-approximation for unconstrained submodular maximization. *SIAM J. Comput.* 44, 5 (2015), 1384–1402.
[15] Gruia Calinescu, Chandra Chekuri, Martin Pal, and Jan Vondrák. 2011. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM J. Comput.* 40, 6 (2011), 1740–1766.
[16] Chandra Chekuri, Shalmoli Gupta, and Kent Quanrud. 2015. Streaming algorithms for submodular function maximization. In *International Colloquium on Automata, Languages, and Programming*. Springer, 318–330.
[17] Chandra Chekuri and Kent Quanrud. 2019. Parallelizing greedy for submodular set function maximization in matroids and beyond. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*. 78–89.
[18] Chandra Chekuri, Jan Vondrák, and Rico Zenklusen. 2010. Dependent randomized rounding via exchange properties of combinatorial structures. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*. IEEE, 575–584.
[19] Chandra Chekuri, Jan Vondrák, and Rico Zenklusen. 2014. Submodular function maximization via the multilinear relaxation and contention resolution schemes. *SIAM J. Comput.* 43, 6 (2014), 1831–1879.
[20] Xi Chen and Binghui Peng. 2021. On the Complexity of Dynamic Submodular Maximization. *arXiv: 2111.03198* (2021).
[21] Alina Ene and Huy L Nguyen. 2019. Submodular maximization with nearly-optimal approximation and adaptivity in nearly-linear time. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 274–282.
[22] Alina Ene, Huy L Nguyen, and Adrian Vladu. 2019. Submodular maximization with matroid and packing constraints in parallel. In *Proceedings of the 51st annual ACM SIGACT symposium on theory of computing*. 90–101.
[23] Matthew Fahrbach, Vahab Mirrokni, and Morteza Zadimoghaddam. 2019. Submodular maximization with nearly optimal approximation, adaptivity and query complexity. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 255–273.
[24] Uriel Feige. 1998. A threshold of $\ln n$ for approximating set cover. *J. ACM* 45, 4 (1998), 634–652.
[25] Moran Feldman, Joseph Naor, and Roy Schwartz. 2011. A unified continuous greedy algorithm for submodular maximization. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*. IEEE, 570–579.
[26] Moran Feldman, Ashkan Norouzi-Fard, Ola Svensson, and Rico Zenklusen. 2020. The one-way communication complexity of submodular maximization with applications to streaming and robustness. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*. 1363–1374.
[27] Moran Feldman, Ashkan Norouzi-Fard, Ola Svensson, and Rico Zenklusen. 2021. Streaming Submodular Maximization with Matroid and Matching Constraints. *arXiv preprint arXiv:2107.07183* (2021).
[28] Daniel Golovin and Andreas Krause. 2011. Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *Journal of Artificial Intelligence Research* 42 (2011), 427–486.
[29] David Kempe, Jon Kleinberg, and Éva Tardos. 2003. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. 137–146.
[30] Silvio Lattanzi, Slobodan Mitrović, Ashkan Norouzi-Fard, Jakub Tarnawski, and Morteza Zadimoghaddam. 2020. Fully dynamic algorithm for constrained submodular optimization. *Advances in Neural Information Processing Systems* (2020).
[31] Hui Lin and Jeff Bilmes. 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.
[32] Vahab Mirrokni, Michael Schapira, and Jan Vondrák. 2008. Tight information-theoretic lower bounds for welfare maximization in combinatorial auctions. In *Proceedings of the 9th ACM conference on Electronic commerce*. 70–77.
[33] Vahab Mirrokni and Morteza Zadimoghaddam. 2015. Randomized composable core-sets for distributed submodular maximization. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*. 153–162.
[34] Morteza Monemizadeh. 2020. Dynamic Submodular Maximization. *Advances in Neural Information Processing Systems* 33 (2020).
[35] George L Nemhauser and Laurence A Wolsey. 1978. Best algorithms for approximating the maximum of a submodular set function. *Mathematics of operations research* 3, 3 (1978), 177–188.
[36] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. 1978. An analysis of approximations for maximizing submodular set functions. *Mathematical programming* 14, 1 (1978), 265–294.
[37] Binghui Peng. 2021. Dynamic influence maximization. *Advances in Neural Information Processing Systems* 34 (2021).
[38] Tim Roughgarden. 2010. Algorithmic game theory. *Commun. ACM* 53, 7 (2010), 78–86.
[39] Jan Vondrák. 2013. Symmetry and approximability of submodular maximization problems. *SIAM J. Comput.* 42, 1 (2013), 265–304.
[40] Kai Wei, Rishabh Iyer, and Jeff Bilmes. 2015. Submodularity in data subset selection and active learning. In *International Conference on Machine Learning*. PMLR, 1954–1963.