Dynamic Algorithms for Matroid Submodular Maximization

Kiarash Banihashem* Leyla Biabani† Samira Goudarzi ‡ MohammadTaghi Hajiaghayi§

Peyman Jabbarzade¶ Morteza Monemizadeh‖

Abstract

Submodular maximization under matroid and cardinality constraints are classical problems with a wide range of applications in machine learning, auction theory, and combinatorial optimization. In this paper, we consider these problems in the dynamic setting where (1) we have oracle access to a monotone submodular function $f : 2^{\mathcal{V}} \to \mathbb{R}^+$ and (2) we are given a sequence $\mathcal{S}$ of insertions and deletions of elements of an underlying ground set $\mathcal{V}$.

We develop the first fully dynamic algorithm for the submodular maximization problem under the matroid constraint that maintains a $(4+\epsilon)$-approximation solution $(0 < \epsilon \leq 1)$ using an expected query complexity of $O(k \log(k) \log^3(k/\epsilon))$, which is indeed parameterized by the rank k of the matroid $\mathcal{M}(\mathcal{V}, \mathcal{I})$ as well.

Chen and Peng [52] at STOC'22 studied the complexity of this problem in the insertion-only dynamic model (a restricted version of the fully dynamic model where deletion is not allowed), and they raised the following important open question: *"for fully dynamic streams [sequences of insertions and deletions of elements], there is no known constant-factor approximation algorithm with poly(k) amortized queries for matroid constraints."* Our dynamic algorithm answers this question as well as an open problem of Lattanzi et al. [109] (NeurIPS'20) affirmatively.

As a byproduct, for the submodular maximization under the cardinality constraint k, we propose a parameterized (by the cardinality constraint k) dynamic algorithm that maintains a $(2+\epsilon)$-approximate solution of the sequence $\mathcal{S}$ at any time t using an expected query complexity of $O(k\epsilon^{-1}\log^2(k))$, which is an improvement upon the dynamic algorithm that Monemizadeh [125] (NeurIPS'20) developed for this problem using an expected query complexity $O(k^2\epsilon^{-3}\log^5(n))$. In particular, this dynamic algorithm is the first one for this problem whose query complexity is independent of the size of ground set $\mathcal{V}$ (i.e., $n = |\mathcal{V}|$).

We develop our dynamic algorithm for the submodular maximization problem under the matroid or cardinality constraint by designing a randomized leveled data structure that supports insertion and deletion operations, maintaining an approximate solution for the given problem. In addition, we develop a fast construction algorithm for our data structure that uses a one-pass over a random permutation of the elements and utilizes monotonicity property of our problems which has a subtle proof in the matroid case. We believe these techniques could also be useful for other optimization problems in the area of dynamic algorithms.

1 Introduction

Submodularity is a fundamental notion that arises in many applications such as image segmentation, data summarization [106, 137], RNA and protein sequencing [146, 113] hypothesis identification [19, 53], information gathering [134], and social networks [99]. A function $f : 2^{\mathcal{V}} \to \mathbb{R}^+$ is called *submodular* if for all $A \subseteq B \subseteq \mathcal{V}$ and $e \notin B$, it satisfies $f(A \cup \{e\}) - f(A) \geq f(B \cup \{e\}) - f(B)$, and it is called *monotone* if for every $A \subseteq B$, it satisfies $f(A) \leq f(B)$.

Given a monotone submodular function $f : 2^{\mathcal{V}} \to \mathbb{R}^+$ that is defined over a ground set $\mathcal{V}$ and a parameter $k \in \mathbb{N}$, in the *submodular maximization problem under the cardinality constraint* k, we would like to report a set $I^* \subseteq \mathcal{V}$ of size at most k whose submodular value is maximum among all subsets of $\mathcal{V}$ of size at most k.

Matroid [131] is a basic branch of mathematics that generalizes the notion of linear independence in vector spaces and has basic links to linear algebra [121], graphs [66], lattices [117], codes [95], transversals [67], and projective geometries [116]. A matroid $\mathcal{M}(\mathcal{V}, \mathcal{I})$ consists of a *ground set* $\mathcal{V}$ and a nonempty downward-closed set system $\mathcal{I} \subseteq 2^V$ (known as the independent sets) that satisfies the *exchange axiom*: for every pair of independent sets $A, B \in \mathcal{I}$ such that $|A| < |B|$, there exists an element $x \in B \backslash A$ such that $A \cup \{x\} \in \mathcal{I}$.

A growing interest in machine learning [142, 88, 70, 104, 22, 114, 138, 69, 4, 145, 62], online auction theory [23, 87, 9, 101, 83, 102, 9, 101, 68], and combinatorial optimization [112, 50, 111, 100, 108, 132, 118] is to study the problem of maximizing a monotone submodular function $f : 2^V \to \mathbb{R}^+$ under a matroid $\mathcal{M}(\mathcal{V}, \mathcal{I})$ constraint. In particular, the goal in

*Computer Science Department, University of Maryland, College Park, MD, USA. kiarash@umd.edu.

†Department of Mathematics and Computer Science, TU Eindhoven, the Netherlands. l.biabani@tue.nl.

‡Computer Science Department, University of Maryland, College Park, MD, USA. samirag@umd.edu.

§Computer Science Department, University of Maryland, College Park, MD, USA. hajiagha@cs.umd.edu.

¶Computer Science Department, University of Maryland, College Park, MD, USA. peymanj@umd.edu.

‖Department of Mathematics and Computer Science, TU Eindhoven, the Netherlands. m.monemizadeh@tue.nl.

the *submodular maximization problem under the matroid constraint* is to return an independent set $I^* \in \mathcal{I}$ of the maximum submodular value $f(I^*)$ among all independent sets in $\mathcal{I}$.

The seminal work of Fisher, Nemhauser and Wolsey [129] in the 1970s, was the first that considered the submodular maximization problem under the matroid constraint problem in the offline model. Indeed, they developed a simple, elegant leveling algorithm for this problem that in time $O(nk)$ (where k is the rank of the matroid $\mathcal{M}(\mathcal{V}, \mathcal{I})$), returns an independent set whose submodular value is a 2-approximation of the optimal value $OPT = \max_{I^* \in \mathcal{I}} f(I^*)$.

However, despite the simplicity and optimality of this celebrated algorithm, there has been a surge of recent research efforts to reexamine these problems under a variety of massive data models motivated by the unique challenges of working with massive datasets. These include streaming algorithms [10, 79, 6, 97], dynamic algorithms [124, 98, 109, 125, 52], sublinear time algorithms [141], parallel algorithms [107, 15, 16, 72, 71, 74, 49], online algorithms [89], private algorithms [45], learning algorithms [13, 12, 14] and distributed algorithms [122, 73, 60, 59].

Among these big data models, the *(fully) dynamic model* [135, 93] has been of particular interest recently. In this model, we see a sequence $\mathcal{S}$ of updates (i.e., inserts and deletes) of elements of an underlying structure (such as a graph, matrix, and so on), and the goal is to maintain an approximate or exact solution of a problem that is defined for that structure using a fast update time. For example, the influential work of Onak and Rubinfeld [130](STOC'10) studied dynamic versions of the matching and the vertex cover problems. Some other new advances in the dynamic model have been seen by developing dynamic algorithms for matching and vertex cover [130, 34, 31, 139, 128, 44, 29, 30, 38], graph connectivity [94, 5], graph sparsifiers [32, 3, 64, 57, 51, 82, 143], set cover [37, 86, 35, 84, 86, 1], approximate shortest paths [26, 90, 27, 28, 91, 144, 92, 2], minimum spanning forests [127, 126, 47], densest subgraphs [36, 136], maximal independent sets [8, 25, 46], spanners [30, 20, 40, 21], and graph coloring [140, 33], to name a few[1].

However, for the very basic problem of submodular maximization under the matroid constraint, there is no (fully) dynamic algorithm known. This problem was repeatedly posed as an open problem at STOC'22 [52] as well as NeurIPS'20 [109]. Indeed, Chen and Peng [52](STOC'22) raised the following open question:

> **Open question [52, 109]:** "For fully dynamic streams [sequences of insertions and deletions of elements], there is no known constant-factor approximation algorithm with poly(k) amortized queries for matroid constraints."

In this paper, we answer this question as well as the open problem of Lattanzi et al. [109] (NeurIPS'20) affirmatively. As a byproduct, we also develop a dynamic algorithm for the submodular maximization problem under the cardinality constraint. We next state our main result.

> Theorem 1.1. (Main Theorem) *Suppose we are provided with oracle access to a monotone submodular function $f : 2^{\mathcal{V}} \to \mathbb{R}^+$ that is defined over a ground set $\mathcal{V}$. Let $\mathcal{S}$ be a sequence of insertions and deletions of elements of the underlying ground set $\mathcal{V}$. Let $0 < \epsilon \leq 1$ be an error parameter.*
>
> - *We develop the first parameterized (by the rank k of a matroid $\mathcal{M}(\mathcal{V}, \mathcal{I})$) dynamic $(4 + \epsilon)$-approximation algorithm for the submodular maximization problem under the matroid constraint using a worst-case expected $O(k \log(k) \log^3 (k/\epsilon))$ query complexity.*
> - *We also present a parameterized (by the cardinality constraint k) dynamic algorithm for the submodular maximization under the cardinality constraint k, that maintains a $(2 + \epsilon)$-approximate solution of the sequence $\mathcal{S}$ at any time t using a worst-case expected complexity $O(k\epsilon^{-1} \log^2(k))$.*

The seminal work of Fisher, Nemhauser and Wolsey [129], which we mentioned above, developed a simple and elegant leveling algorithm for the submodular maximization problem under the cardinality constraint that achieves the optimal approximation ratio of $\frac{e}{e-1} \approx 1.58$ in time $O(nk)$ [129, 76].

The study of the submodular maximization in the dynamic model was initiated at NeurIPS 2020 based on two independent works due to Lattanzi, Mitrovic, Norouzi-Fard, Tarnawski, and Zadimoghaddam [109] and Monemizadeh [125]. Both works present dynamic algorithms that maintain $(2 + \epsilon)$-approximate solutions for the submodular maximization under the cardinality constraint k in the dynamic model. The amortized expected query complexity of these two algorithms are $O(\epsilon^{-11} \log^6(k) \log^2(n))$ and $O(k^2\epsilon^{-3} \log^5(n))$, respectively.

Our dynamic algorithm for the cardinality constraint improves upon the dynamic algorithm that Monemizadeh [125] (NeurIPS'20) developed for this problem using an expected query complexity $O(k^2\epsilon^{-3} \log^5(n))$. In particular, our dynamic

[1] Interestingly, the best paper awards at SODA'23 were awarded to two dynamic algorithms [24, 39] for the matching size problem in the dynamic model.

algorithm is the first one for this problem whose query complexity is independent of the size of the ground set $\mathcal{V}$.

We develop our dynamic algorithm for the submodular maximization problem under the matroid or cardinality constraint by designing a randomized leveled data structure that supports insertion and deletion operations, maintaining an approximate solution for the given problem. In addition, we develop a fast construction algorithm for our data structure that uses a one-pass over a random permutation of the elements and utilizes a monotonicity property of our problems which has a subtle proof in the matroid case. We believe these techniques could also be useful for other optimization problems in the area of dynamic algorithms.

Interestingly, our results can be seen from the lens of parameterized complexity [119, 81, 103, 96, 75, 54, 56, 55, 85]. In particular, the query complexity of our dynamic algorithms for the submodular maximization problems under the matroid and cardinality constraints (1) are independent of the size of the ground set $\mathcal{V}$ (i.e., $|\mathcal{V}| = n$) , and (2) are parameterized by the rank k of the matroid $\mathcal{M}(\mathcal{V}, \mathcal{I})$ and the cardinality k, respectively. We hope that our work sheds light on the connection between dynamic algorithms and the Fixed-Parameter Tractability (FPT) [61, 80, 58] world. We should mention that streaming algorithms [75, 54, 56] through the lens of the parameterized complexity have been considered before where vertex cover and matching parameterized by their size were designed in these works.

Finally, one may ask whether we can obtain a dynamic c-approximate algorithm for the cardinality constraint for $c < 2$ with a query complexity that is polynomial in k. Let $g : \mathbb{N} \to \mathbb{R}^+$ be an arbitrary function. Building on a hardness result recently obtained by Chen and Peng [52], we show in Appendix C that there is no randomized $(2 - \epsilon)$-approximate algorithm for the dynamic submodular maximization under cardinality constraint k with amortized expected query time of $g(k)$ (e.g., not even doubly exponentially in k), even if the optimal value is known after every insertion/deletion.

Concurrent work. In a concurrent work, Dütting, Fusco, Lattanzi, Norouzi-Fard, and Zadimoghaddam [65] also provide an algorithm for dynamic submodular maximization under a matroid constraint. Their algorithm obtains a $4 + \epsilon$ approximation with $\frac{k^2}{\epsilon}\log(k)\log^2(n)\log^3(\frac{k}{\epsilon})$ amortized expected query comlpexity. [2]

Our query complexity of $k\log(k)\log^3(\frac{k}{\epsilon})$ is strictly better as **(a)** it does not depend on n and **(b)** its dependence on k is nearly linear rather than nearly quadratic and the dependence on ϵ^{-1} is polylogarithmic. Additionally, our guarantees are worst-case expected, rather than amortized expected.

1.1 Preliminaries

Notations. For two natural numbers $x < y$, we use $[x, y]$ to denote the set $\{x, x+1, \cdots, y\}$, and $[x]$ to denote the set $\{1, 2, \cdots, x\}$. For a set A and an element e, we denote by $A + e$, the set that is the union of two sets A and $\{e\}$. Similarly, for a set A and an element $e \in A$, we denote by $A - e$ or $A \backslash e$, the set A from which the element e is removed. For a level L_i, we represent by $L_{1\le j\le i}$ the levels $L_1, L_2, \cdots, L_i$, and we simplify $L_{1\le j\le i}$ and show it by $L_{\le i}$. The levels $L_{i\le j\le T}$ and its simplification $L_{i\le}$ are defined similarly. For a function x and a set A, we denote by $x[A]$ the function x that is restricted to domain A. For an event E, we use $\mathbb{1}[E]$ as the *indicator function* of E. That is, $\mathbb{1}[E]$ is set to one if E holds and is set to zero otherwise. For random variables and their values, we use bold and non-bold letters, respectively. For example, we denote a random variable by $\mathbf{X}$ and its value by X. We will use the notations $\mathbb{P}[\mathbf{X}]$ and $\mathbb{E}[\mathbf{X}]$ to denote the probability and the expectation of a random variable $\mathbf{X}$. For two events A and B, we will use the notation $\mathbb{P}[A|B]$ to denote "the conditional probability of A given B" or "the probability of A under the condition B". For an event A with nonzero probability and a discrete random variable $\mathbf{X}$, we denote by $\mathbb{E}[\mathbf{X}|A]$ conditional expectation of X given A, which is $\mathbb{E}[\mathbf{X}|A] = \sum_x x \cdot \mathbb{P}[\mathbf{X} = x|A]$. Similarly, if $\mathbf{X}$ and $\mathbf{Y}$ are discrete random variables, the conditional expectation of $\mathbf{X}$ given $\mathbf{Y}$ is denoted by $\mathbb{E}[\mathbf{X}|\mathbf{Y} = y]$.

Submodular function. Given a ground set $\mathcal{V}$, a function $f : 2^{\mathcal{V}} \to \mathbb{R}^+$ is called *submodular* if it satisfies $f(A \cup \{e\}) - f(A) \ge f(B \cup \{e\}) - f(B)$, for all $A \subseteq B \subseteq \mathcal{V}$ and $e \notin B$. In this paper, we assume that f is *normalized*, i.e., $f(\emptyset) = 0$. When f satisfies the additional property that $f(A \cup \{e\}) - f(A) \ge 0$ for all A and $e \notin A$, we say f is *monotone*. For a subset $A \subseteq \mathcal{V}$ and an element $e \in \mathcal{V}\backslash A$, the function $f(A \cup \{e\}) - f(A)$ is often called the *marginal gain* [10, 97] of adding e to A.

Let $f : 2^{\mathcal{V}} \to \mathbb{R}^+$ be a monotone submodular function defined on the ground set $\mathcal{V}$. The *monotone submodular maximization problem under the cardinality constraint* k is defined as finding $OPT = \max_{I\subseteq\mathcal{V}:|I|\le k} f(I)$. We denote by I^* an optimal subset of size at most k that achieves the optimal value $OPT = f(I^*)$. Note that we can have more than one optimal set.

The leveling algorithm of the seminal work of Nemhauser, Wolsey, and Fisher [129] that can approximate OPT to a factor of $(1 - 1/e)$, is as follows. In the beginning, we let $S = \emptyset$. We then take k passes over the set V, and in each pass, we find an element $e \in V$ that maximizes the marginal gain $f(S \cup \{e\}) - f(S)$, add it to S and delete it from V.

[2] The two works appeared on arxiv at the same time; We had submitted an earlier version of our work to SODA'23, in July 2022.

Access Model. We assume the access to a monotone submodular function $f : 2^{\mathcal{V}} \to \mathbb{R}^+$ is given by an *oracle*. That is, we consider the oracle that allows *set queries* such that for every subset $A \subseteq \mathcal{V}$, one can query the value $f(A)$. The marginal gain $f(A \cup \{e\}) - f(A)$ for every subset $A \subseteq \mathcal{V}$ and an element $e \in \mathcal{V}$ in this query access model can be computed using two queries $f(A \cup \{e\})$ and $f(A)$.

Matroid. A matroid $\mathcal{M}(\mathcal{V}, \mathcal{I})$ consists of a *ground set* $\mathcal{V}$ and a nonempty downward-closed set system $\mathcal{I} \subseteq 2^{\mathcal{V}}$ (known as the independent sets) that satisfies the *exchange axiom*: for every pair of independent sets $A, B \in \mathcal{I}$ such that $|A| < |B|$, there exists an element $x \in B \backslash A$ such that $A \cup \{x\} \in \mathcal{I}$. A subset of the ground set $\mathcal{V}$ that is not independent is called *dependent*. A maximal independent set—that is, an independent set that becomes dependent upon adding any other element—is called a *basis* for the matroid $\mathcal{M}(\mathcal{V}, \mathcal{I})$. A *circuit* in a matroid $\mathcal{M}(\mathcal{V}, \mathcal{I})$ is a minimal dependent subset of $\mathcal{V}$—that is, a dependent set whose proper subsets are all independent. Let A be a subset of V. The rank of A, denoted by $rank(A)$, is the maximum cardinality of an independent subset of A.

Let $f : 2^{\mathcal{V}} \to \mathbb{R}^{\geq 0}$ be a monotone submodular function defined on the ground set $\mathcal{V}$ of a matroid $\mathcal{M}(\mathcal{V}, \mathcal{I})$. We denote by $OPT = \max_{I \in \mathcal{I}} f(I)$ the maximum submodular value of an independent set in $\mathcal{I}$. We denote by $I^* \in \mathcal{I}$ an independent set that achieves the optimal value $OPT = f(I^*)$.

Here, we bring two lemmas about matroids that will be used in our paper.

Lemma 1.1. ([131]) *The family C of circuits of a matroid $\mathcal{M}(\mathcal{V}, \mathcal{I})$ has the following properties:*

- *(C1) $\emptyset \notin C$.*
- *(C2) if $C_1, C_2 \in C$ and $C_1 \subseteq C_2$, then $C_1 = C_2$.*
- *(C3) if $C_1, C_2 \in C$, $C_1 \neq C_2$ and $e \in C_1 \cap C_2$, then there exists $C_3 \in C$ such that $C_3 \subseteq C_1 \cup C_2 \setminus \{e\}$.*

Lemma 1.2. *Let $e \in V$ be an element, and $I \in \mathcal{I}$ be an independent set. Then $I \cup \{e\}$ has at most one circuit.*

Proof. For the sake of contradiction, suppose there are two circuits $C_1, C_2 \subseteq I \cup \{e\}$, where $C_1 \neq C_2$. As I is an independent set, $C_1, C_2 \nsubseteq I$, which means $e \in C_1 \cap C_2$. Then using Lemma 1.1, there exists a circuit $C_3 \subseteq C_1 \cup C_2 \setminus \{e\}$. Since $C_1 \cup C_2 \setminus \{e\} \subseteq I$, we have $C_3 \subseteq I$, which is a contradiction to the fact that the set I is an independent set in $\mathcal{I}$. □

Dynamic Model. Let $\mathcal{S}$ be a sequence of insertions and deletions of elements of an underlying ground set $\mathcal{V}$. Let $\mathcal{S}_t$ be the sequence of the first t updates (insertion or deletion) of the sequence $\mathcal{S}$. By time t, we mean the time after the first t updates of the sequence $\mathcal{S}$, or simply when the updates of $\mathcal{S}_t$ are done. We define V_t as the set of elements that have been inserted until time t but have not been deleted after their latest insertion.

Given a monotone submodular function $f : 2^{\mathcal{V}} \to \mathbb{R}^+$ defined on the ground set $\mathcal{V}$, the aim of *dynamic monotone submodular maximization problem under the cardinality constraint* k is to have (an approximation of) $OPT_t = \max_{I_t \subseteq V_t : |I| \leq k} f(I_t)$ at any time t. Similarly, the aim of *dynamic monotone submodular maximization problem under a matroid $\mathcal{M}(\mathcal{V}, \mathcal{I})$ constraint* for a monotone function f defined over the ground set $\mathcal{V}$ is to have (an approximation of) $OPT_t = \max_{I_t \subseteq V_t : I_t \in \mathcal{I}} f(I_t)$ at any time t. We also define MAX_t to be $\max_{e \in V_t} f(e)$. For simplicity, during the analysis for a fixed time t, we use V, OPT, and MAX instead of V_t, OPT_t, and MAX_t respectively.

Our dynamic algorithm is in the oblivious adversarial model as is common for analysis of randomized data structures such as universal hashing [42]. The model allows the adversary, who is aware of the submodular function f and the algorithm that is going to be used, to determine all the arrivals and departures of the elements in the ground set $\mathcal{V}$. However, the adversary is unaware of the random bits used in the algorithm and so cannot choose updates adaptively in response to the randomly guided choices of the algorithm. Equivalently, we can suppose that the adversary prepares the full input (insertions and deletions) before the algorithm runs.

The *query complexity* of an α-approximate dynamic algorithm is the number of oracle queries that the algorithm must make to maintain an α-approximate of the solution at time t, given all computations that have been done till time $t - 1$.

We measure the *time complexity* of our dynamic algorithm in terms of its *query complexity*, taking into account queries made to either the submodular oracle for f or the matroid independence oracle for $\mathcal{I}$.

The query complexities of the algorithms in our paper will be worst-case expected. An algorithm is said to have worst-case expected update time (or query time) α if for every update x, the expected time to process x is at most α. We refer to Bernstein, Forster, and Henzinger [30] for a discussion about the worst-case expected bound for dynamic algorithms.

1.2 Our contribution and overview of techniques

Our dynamic algorithms for the submodular maximization problems with cardinality and matroid constraints consist of the following three building blocks.

- *Fast leveling algorithms:* We first develop linear-time leveling algorithms for these problems based on random permutations of elements. These algorithms are used in *rare occasions* when we need to (partially or totally) reset a solution that we maintain.
- *Insertion and deletion subroutines:* We next design subroutines for inserting and deleting a new element. Upon

insertion or deletion of an element, these subroutines often perform *light* computations, but in rare occasions, they perform *heavy* operations by invoking the leveling algorithms to (partially or totally) reset a solution that we maintain.

- *Relax OPT or MAX assumptions:* For the leveling algorithms, and the insertion and the deletion subroutines, we assume we know either an approximation of OPT (as for the cardinality constraint) or an approximation of the maximum submodular value $MAX = \max_{e \in V} f(e)$ of an element (as for the matroid constraint). In the final block of our dynamic algorithms, we show how to relax such an assumption.

1.2.1 Submodular maximization problem under the cardinality constraint Designing and analyzing the above building blocks for the cardinality constraint is simpler than designing and developing them for the matroid constraint. Therefore, we outline them first for the cardinality constraint. That gives the intuition and sheds light on how we develop these building blocks for the matroid constraint which are more involved. Since the main contribution of our paper is developing a dynamic algorithm for the matroid constraint, we explain the algorithms (in Section 2) and the analysis (in Section 3) for the matroid first. The dynamic algorithm and its analysis for the cardinality constraint are given in Section 4 and Appendix B, respectively.

Suppose for now, we know the optimal value $OPT = \max_{I^* \subseteq V : |I^*| \leq k} f(I^*)$ of any subset of the set V of size at most k. We consider the fixed threshold $\tau = \frac{OPT}{2k}$.

First building block: Fast leveling algorithm. Our leveling algorithm constructs a set of levels $L_0, L_1, \cdots, L_T$, where T is a random variable guaranteed to be $T \leq k$. Every level L_ℓ consists of two sets R_ℓ and I_ℓ, and an element e_ℓ so that:

1. $R_0 = V$, $I_0 = \emptyset$, and $R_1 = \{e \in R_0 : f(e) \geq \tau\}$
2. $R_0 \supseteq R_1 \supset \cdots \supset R_T \supset R_{T+1} = \emptyset$
3. For $1 \leq \ell \leq T$, we have $I_\ell = I_{\ell-1} \cup \{e_\ell\}$
4. We report the set I_T as the solution

The illustration of our construction is shown in Figure 1.

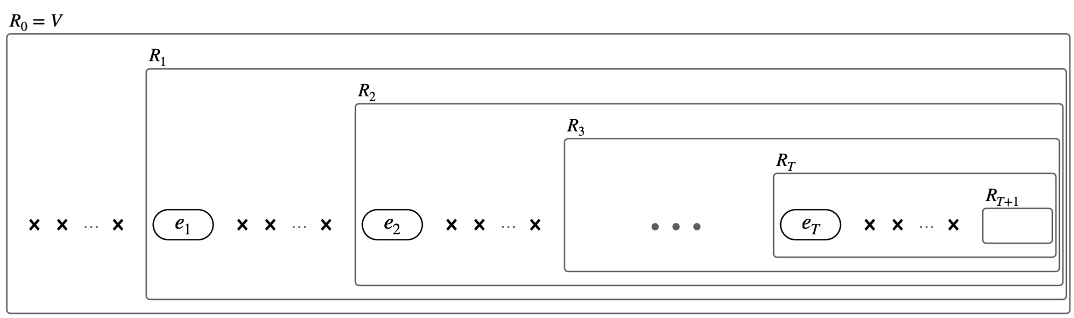

Figure 1: The illustration of our leveling algorithm.

The key concept in constructing the levels is the notion of *promoting elements*.

Definition 1.1. (Promoting elements) *Let $L_{1 \leq \ell \leq T}$ be a level. We call an element $e \in R_\ell$, a promoting element for the set I_ℓ if $f(I_\ell \cup \{e\}) - f(I_\ell) \geq \tau$ and $|I_\ell| < k$.*

The levels are constructed as follows: Let $\ell = 1$. We first randomly permute the elements of the set R_1 and denote by P this random permutation. We next iterate through the elements of P and for every element $e \in P$, we check if e is a promoting element with respect to the set $I_{\ell-1}$ or not.

- If e is a promoting element for the set $I_{\ell-1}$, we then let e_ℓ be e and let I_ℓ be $I_{\ell-1} \cup \{e_\ell\}$. Observe that now we have the set I_ℓ and the element e_ℓ, however, the set R_ℓ is not complete yet, as some of its elements may come after e in the permutation P. We create the next level $L_{\ell+1}$ by setting $R_{\ell+1} = \emptyset$. We then proceed to the next element in P. Note that in this way, for all levels $L_{1<j\leq\ell}$, the sets R_j are not complete, and they will be complete when we reach the end of the permutation P.
- Next, we consider the case when e is not a promoting element for the set $I_{\ell-1}$. This essentially means that we need to find the largest $z \in [1, \ell-1]$ so that e is promoting for the set I_{z-1}, but it is not promoting for the set I_z. A naive way of doing that is to perform a linear scan for which we need one oracle query to compute $f(I_x \cup \{e\}) - f(I_x)$ for every $x \in [1, \ell-1]$. However, we do a binary search in the interval $[i, \ell-1]$, which needs $O(\log k)$ oracle queries to find z.

Once we find such z, we add e to all sets R_r for $2 \le r \le z$[3]. Observe that adding e to all these sets may need $O(k)$ time, but we do not need to do oracle queries in order to add e to these sets.

The permutation P has at most n elements, and we do the above operations for every such element. Thus, the leveling algorithm may require a total of $O(n \log k)$ oracle queries. Observe that the implicit property that we use to perform the binary search is the *monotonicity property* which says if an element is a promoting for a set I_{z-1}, it is promoting for all sets $I_{\le z-1}$. For the cardinality constraint, the monotone property is trivial to see. However, it is ***intricate*** for the ***matroid*** constraint. We will develop a leveling algorithm for the matroid constraint, which satisfies a monotonicity property, allowing us to perform the binary search.

Second building block: Insertion and deletion of an element. Next, we explain the insertion and deletion subroutines. Let $\mathcal{S}$ be a sequence of insertions and deletions of elements of an underlying ground set $\mathcal{V}$. First, suppose we would like to delete an element v. Observe that the set R_0 should contain all elements that have been inserted but not deleted so far. Thus, we remove v from R_0. Now, two cases can occur:

- *Light computation:* The first case is when $v \notin I_i$ for all $i \in [T]$. Then, we do a light computation by iterating through the levels $L_1, \cdots, L_T$, and for each level L_i, we delete v from R_i. Handling this light computation takes zero query complexity as we do not make any oracle query.
- *Heavy computation:* However, if there exists a level $i \in [T]$ where $e_i = v$, we then rebuild all levels $L_{i \le j \le T}$. To this end, we invoke the leveling algorithm for the level L_i to rebuild the levels $L_i, \cdots, L_T$. This computation is heavy, for which we need to make $O(|R_i| \log k)$ oracle queries.

When we invoke the leveling algorithm for a level L_i, it randomly permutes the elements R_i and iterates through this random permutation to compute I_ℓ, R_ℓ, and e_ℓ for $i \le \ell \le T$. We prove that this means for every level $L_{\ell \ge i}$, the promoting element e_ℓ that we picked is sampled uniformly at random from the set R_ℓ. This ensures that the probability that the sampled element e_ℓ being deleted is $\frac{1}{|R_\ell|}$. Therefore, the expected number of oracle queries to reconstruct the levels $[i, T]$ is at most $O(\frac{1}{|R_i|}|R_i| \log k) = O(\log k)$. Since there are at most $T \le k$ levels, by the linearity of expectation, the expected oracle queries that a deletion can incur is $O(k \log k)$.

Next, suppose we would like to insert an element v. First of all, the set R_0 should contain all elements that have been inserted but not deleted so far. Thus, we add v to R_0. Later, we iterate through levels $L_1, \cdots, L_{T+1}$, and for each level L_i, we check if v is a promoting element for the previous level or not. If it is not, we break the loop and exit the insertion subroutine. However, if v is a promoting element for the level L_{i-1}, we then add it to the set R_i and with probability $\frac{1}{|R_i|}$, we let e_i be v and invoke the leveling algorithm for the level L_{i+1} to rebuild the levels $L_{i+1}, \cdots, L_T$. The proof of correctness for insertion uses similar techniques to the proof for deletion.

Third building block: Relax the assumption of having OPT. Our dynamic algorithm assumes the optimal value $OPT = \max_{I^* \subseteq V : |I^*| \le k} f(I^*)$ is given as a parameter. However, in reality, the optimal value is not known in advance and it may change after every update. To remove this assumption, we use the well-known technique that has been also used in [109]. Indeed, we run parallel instances of our dynamic algorithm for different guesses of the optimal value OPT_t for the set V_t of elements that have been inserted till time t, but not deleted, such that for any time t, $\max_{I^* \subseteq V : |I^*| \le k} f(I^*) \in (OPT_t/(1+\epsilon), OPT_t]$ in one of the runs. These guesses are $(1+\epsilon)^i$ where $i \in \mathbb{Z}$. We apply each update on only $O(\log(k)/\epsilon)$ instances of our algorithm. See Section 4 for the details.

1.2.2 Submodular maximization problem under the matroid constraint

The dynamic algorithm that we develop for the matroid constraint has similar building blocks as the cardinality constraint, but it is more intricate. We outline these building blocks for the matroid constraint next.

First building block: Leveling algorithm. Let $\mathcal{M}(\mathcal{V}, \mathcal{I})$ be a matroid whose rank is $k = rank(\mathcal{M})$. We first assume that we have the maximum submodular value $MAX = \max_{e \in V} f(e)$. We relax this assumption later. Our leveling algorithm builds a set of levels $L_0, L_1, \cdots, L_T$, where T is a random variable guaranteed to satisfy $T = O(k \log(k/\epsilon))$. Every level L_i consists of three sets R_i, I_i, and I'_i, and an element e_i. For these sets, we have the following properties:

1. $V = R_0 \supseteq R_1 \supset \cdots \supset R_T \supset R_{T+1} = \emptyset$
2. The sets I_i are independent sets, i.e., $I_i \in \mathcal{I}$
3. Each I'_i is the union of all I_j for $j \le i$, i.e., $I'_i = \bigcup_{j \le i} I_j$
4. The sets I'_i *are not necessarily* independent
5. We report the set I_T as the solution

The illustration of our construction is similar to the one for the cardinality constraint and is shown in Figure 1. A key

[3]Observe that z is already in R_1

concept in our algorithm is again the notion of *promoting elements*. However, the definition of promoting elements for the matroid constraint is more complex than that of the cardinality constraint, and is inspired by the streaming algorithm of Chakrabarti and Kale [43]. The complexity comes from the fact that adding an element e to an independent set, say I may preserve the independency of I or it may violate it by creating a circuit[4]. In Lemma 1.2, we prove that adding e to an independent set can create at most one circuit. Thus, we need to handle both cases when we define the notion of promoting elements.

Definition 1.2. (Promoting elements) *Let $L_{1\le\ell\le T}$ be a level.We call an element e, a promoting element for the level L_ℓ if*

- ***Property 1:*** $f(I'_\ell + e) - f(I'_\ell) \ge \frac{\epsilon}{10k} \cdot MAX$, *and*
- *One of the following properties hold:*
  - ***Property 2:*** *$I_\ell + e$ is independent set (i.e., $I_\ell + e \in \mathcal{I}$)* or
  - ***Property 3:*** *$I_\ell + e$ is not independent and the minimum weight element $\hat{e} = \arg\min_{e' \in C} w(e')$ of the set $C = \{e' \in I_\ell : I_\ell + e - e' \in \mathcal{I}\}$ satisfies $2w(\hat{e}) \le f(I'_\ell + e) - f(I'_\ell)$.*

We next explain the leveling algorithm. We first initialize I_0 and I'_0 as empty sets and let R_0 be the set of existing elements V. We then let R_1 be all elements of the set R_0 that are promoting with respect to L_0. Observe that since I_0 and I'_0 are empty sets, an element is filtered out from level L_0 because of Property 1.

The leveling algorithm can be called for any level L_i and starting at that level, it builds the rest of levels $L_{i\le j\le T}$. Suppose our leveling algorithm is called for a level L_i for $i \ge 1$. Let $\ell = i$. We randomly permute the set R_i and let P be this random permutation. We iterate through the elements of P and upon seeing a new element e, we check if e is a promoting element for the level $L_{\ell-1}$.

- The first case occurs if e is a promoting element for the level $L_{\ell-1}$. Note that e is promoting if satisfies Property 1 and one of Properties 2 and 3.
  - If the element e satisfies Properties 1 and 2, we set $I_\ell = I_{\ell-1} + e$.
  - If e satisfies Properties 1 and 3, we set $I_\ell = I_{\ell-1} + e - \hat{e}$.

  In both cases, the resulting I_ℓ is an independent set in $\mathcal{I}$. We then fix the weight of e to be $w(e) = f(I'_{\ell-1} + e) - f(I'_{\ell-1})$. Later, we let $I'_\ell := I'_{\ell-1} + e$, and $e_\ell = e$. Similar to the leveling algorithm that we develop for the cardinality case, we now have the sets I_ℓ and I'_ℓ, and the element e_ℓ. However, the set R_ℓ is not complete yet, as some of its elements may come after e in the permutation P. We create the next level $L_{\ell+1}$ by setting $R_{\ell+1} = \emptyset$. We then proceed to the next element in P. Note that in this way, for all levels $L_{i<j\le\ell}$, the sets R_j are not complete and they will be complete when we reach the end of the permutation P.
- The second case is when e is not a promoting element for $L_{\ell-1}$. Here, similar to the cardinality constraint, our goal is to perform the binary search to find the smallest $z \in [i, \ell - 1]$ so that e is promoting for the level L_{z-1}, but it is not promoting for the level L_z. ***Interestingly, we prove the monotonicity property holds for the matroid constraint***. (The proof of this subtle property is given in Section 3.1.) That is, we prove if e is promoting for a level L_{x-1}, it is promoting for all levels $L_{r\le x-1}$ and if e is not promoting for a level L_x, it is not promoting for all levels $L_{r\ge x}$. Thus, we can do the binary search to find the smallest $z \in [i, \ell - 1]$ so that e is promoting for the level L_{z-1}, but it is not promoting for the level L_z, which needs $O(\log(T)) = O(\log(k \log(k/\epsilon))$ steps of binary search. Once we find such z, we add e to all sets R_r for $i < r \le z$. Unlike the previous case, however, we stay in the current level L_ℓ and proceed to the next element of P. Observe that adding e to all these sets may need $T = O(k \log(k/\epsilon))$ time, but we do not need to do oracle queries in order to add e to these sets.

Overview of the analysis: In order to prove the correctness of our leveling algorithm and compute its query complexity, we define two invariants; ***level*** and ***uniform*** invariants. The level invariant itself is a set of 5 invariants ***starter***, ***survivor***, ***independent***, ***weight***, and ***terminator***. We show that these invariants are fulfilled by the end of the leveling algorithm (in Section 3.2) and after every insertion and deletion of an element (in Section 3.3).

The level invariants assert that all elements that are added to R_i at a level L_i are promoting elements for the previous level. In other words, those elements of the set $R_{i-1} \backslash e_{i-1}$ that are not promoting will be filtered out and not be seen in R_i. Intuitively, this invariant provides us the approximation guarantee. The uniform invariant asserts that for every level $L_{i\in[T]}$, conditioned on the random sets $R_{j\le i}$ and random elements $e_{j<i}$, the element e_i is chosen uniformly at random from the set R_i.

[4]A *circuit* in a matroid $\mathcal{M}$ is a minimal dependent subset of V—that is, a dependent set whose proper subsets are all independent.

That is, $\mathbb{P}\left[e_i = e | R_{j\leq i} \wedge e_{j<i}\right] = \frac{1}{|R_i|} \cdot \mathbb{1}\left[e \in R_i\right]$[5]. Intuitively, this invariant provides us with the randomness that we need to fool the adversary in the (fully) dynamic model which in turn helps us to develop a dynamic algorithm for the submodular maximization problem under a matroid constraint.

The proof that the level and uniform invariants hold after every insertion and deletion is ***novel*** and ***subtle***. This proof is given in Sections 3.2 and 3.3. The technical part is to show that all promoting elements that are added to R_i at a level L_i (from the previous level) will be promoting after every update (i.e., insert or delete) and also, the sets I_i will remain independent after updates. In addition, we need to show that uniformly chosen elements e_i from survivor set R_i will be uniform after every update.

Now, we overview how we analyze the query complexity of our leveling algorithm. Checking if an element e is promoting for a level L_i can be done using $O(\log(k))$ oracle queries using a binary search argument. The proof is given in Section 3.1. The binary search that we perform in order to place an element e in the correct level requires $O(\log T)$ such promoting checks. Thus, if we initiate the leveling algorithm with a set R_i, our algorithm needs $O(|R_i| \log(k) \log(T))$ oracle queries to build the levels $L_i, \cdots, L_T$ for $T = O(k \log(k/\epsilon))$.

Second building block: Insertion and deletion of an element. Now, we explain how to maintain the independent set I_T upon insertions and deletions of elements. First, suppose we would like to delete an element e. We iterate through levels $L_1, \cdots, L_T$ and for each level L_i we delete e from R_i and we later check if e is the element e_i that we have picked for the level L_i. If this is the case, we then invoke the leveling algorithm for the set R_i to reset the levels $L_i, \cdots, L_T$. Since, the invocation of the leveling algorithm for the level L_i may initiate $O(|R_i| \log(k) \log(T))$ oracle queries (to build the levels $L_i, \cdots, L_T$) and since the element e_i is chosen uniformly at random from the set R_i and we iterate through levels $L_1, \cdots, L_T$, thus, the worst-case expected query complexity of deletion is $\sum_{i=1}^{T} \frac{1}{|R_i|} \cdot O(|R_i| \cdot \log(k) \cdot \log(T)) = O(k \log(k) \log^2(k/\epsilon))$.

Next, suppose we would like to insert an element e. First of all, the set R_0 should contain all elements that have been inserted but not deleted so far. Thus, we add e to R_0. Later, we iterate through levels $L_1, \cdots, L_T$ and for each level L_i, we check if e is a promoting element for that level or not. If it is not, we break the loop and exit the insertion subroutine. However, if e is indeed, a promoting element for the level L_i, we then add it to the set R_i and with probability $1/|R_i|$, we set $e_i = e$ and invoke the leveling algorithm (with the input index $i+1$) to reset the subsequent levels $L_{i+1}, \cdots, L_T$. The query complexity of an insertion is proved similar to what we showed for a deletion.

The third block of our dynamic algorithm for the matroid constraint is to relax the assumption of knowing MAX. Relaxing this assumption is similar to what we did for the cardinality constraint. See Section 2 for the details.

1.3 Related Work

In this section, we state some known results for the submodular maximization problem under the matroid and cardinality constraints or some other related problems in the streaming, distributed, and dynamic models. In Table 1, we summarize the results in streaming and dynamic models for the submodular maximization problem under the matroid or cardinality constraint.

model	result	problem	approx.	query complexity	ref.
dynamic streaming model	algorithm	cardinality	$2+\epsilon$	$O(\epsilon^{-1} dk \log(k))$	[124]
		cardinality	$2+\epsilon$	$O(dk \log^2(k) + d \log^3(k))$	[98]
		matroid	$5.582 + O(\epsilon)$	$O(k + \epsilon^{-2} d \log(k))$	[63]
insertion-only dynamic model	algorithm	matroid	$2+\epsilon$	$k^{\tilde{O}(1/\epsilon)}$	[52]
			$\frac{e}{e-1} + \epsilon$	$k^{\tilde{O}(1/\epsilon^2)} \cdot \log(n)$	[52]
fully dynamic model	algorithm	matroid	$4+\epsilon$	$O(k \log(k) \log^3(k/\epsilon))$	this paper
				$O(\frac{k^2}{\epsilon} \log(k) \log^2(n) \log^3(\frac{k}{\epsilon}))$	[65]
		cardinality	$2+\epsilon$	$O(\epsilon^{-3} k^2 \log^4(n))$	[125]
				$O(\epsilon^{-4} \log^4(k) \log^2(n))$	[110]
				$O(\text{poly}(\log(n), \log(k), \frac{1}{\epsilon}))$	[17]
				$O(k\epsilon^{-1} \log^2(k))$	this paper
	lower bound	cardinality	$2-\epsilon$	$n^{\Omega(\epsilon)}/k^3$	[52]
			1.712	$\Omega(n/k^3)$	[52]

Table 1: Results for the submodular maximization subject to cardinality and matroid constraints. The lower bounds presented in [52] assume that we know the optimal submodular maximization value of the sub-sequence $\mathcal{S}_t$, where $\mathcal{S}_t$ is the set of elements that are inserted but not deleted from the beginning of the sequence $\mathcal{S}$ till any time t.

[5] For an event A, we define $\mathbb{1}[A]$ as the *indicator function* of A. That is, $\mathbb{1}[A]$ is set to one if A holds and is set to zero otherwise.

Known dynamic algorithms. The study of the submodular maximization in the dynamic model is initiated at NeurIPS 2020 based on two independent works. The first work is due to Lattanzi, Mitrovic, Norouzi-Fard, Tarnawski, and Zadimoghaddam [109] who present a randomized dynamic algorithm that maintains an expected $(2+\epsilon)$-approximate solution of the maximum submodular (under the cardinality constraint k) of a dynamic sequence $\mathcal{S}$. The amortized expected query complexity of their algorithm is $O(\epsilon^{-11}\log^6(k)\log^2(n))$. The second work is due to Monemizadeh [125] who presents a randomized dynamic algorithm with approximation guarantee $(2+\epsilon)$. The amortized expected query complexity of his algorithm is $O(\epsilon^{-3}k^2\log^5(n))$. The original version of the algorithm in Lattanzi et al. [109] has some correctness issues, as pointed out by Banihashem, Biabani, Goudarzi, Hajiaghayi, Jabbarzade, and Monemizadeh [17] at ICML 2023, who also provide an alternative algorithm for solving this problem with polylogarithmic update time. Those issues were also subsequently fixed by Lattanzi et al. [110] by modifying their algorithm. The query complexity of their new algorithm is $O(\epsilon^{-4}\log^4(k)\log^2(n))$ per update. Peng's work at NeurIPS 2021 [133] focuses on the dynamic influence maximization problem, which is a white box dynamic submodular maximization problem. Work of Banihashem, Biabani, Goudarzi, Hajiaghayi, Jabbarzade, and Monemizadeh [18] at NeurIPS 2023 solves dynamic non-monotone submodular maximization under cardinality constraint k.

At STOC 2022, Chen and Peng [52] show two lower bounds for the submodular maximization in the dynamic model. Both of these lower bounds hold even if we know the optimal submodular maximization value of the sequence $\mathcal{S}$ at any time t. Their first lower bound shows that any randomized algorithm that achieves an approximation ratio of $2-\epsilon$ for dynamic submodular maximization under cardinality constraint k requires amortized query complexity $n^{\tilde{\Omega}(\epsilon)}/k^3$. They also prove a stronger result by showing that any randomized algorithm for dynamic submodular maximization under cardinality constraint k that obtains an approximation guarantee of 1.712 must have amortized query complexity at least $\Omega(n/k^3)$.

Chen and Peng [52] also studied the complexity of the submodular maximization under matroid constraint in the insertion-only dynamic model (a restricted version of the fully dynamic model where deletions are not allowed) and they developed two algorithms for this problem. The first algorithm maintains a $(2+\epsilon)$-approximate independent set of a matroid $\mathcal{M}(\mathcal{V},\mathcal{I})$ such that the expected number of oracle queries per insertion is $k^{\tilde{O}(1/\epsilon)}$. Their second algorithm is a $(\frac{e}{e-1}+\epsilon)$-approximation algorithm using an amortized query complexity of $k^{\tilde{O}(1/\epsilon^2)}\cdot\log(n)$, where k is the rank of $\mathcal{M}(\mathcal{V},\mathcal{I})$ and $n=|\mathcal{V}|$. However, these results do not work for the classical (fully) dynamic model, and they posed developing a dynamic algorithm for the submodular maximization problem under the matroid constraint in the (fully) dynamic model as an open problem.

And as discussed previously, the concurrent work of Dütting et al. [65] at ICML 2023 provides an algorithm for dynamic submodular optimization under matroid constraint. Their algorithm has a $4+\epsilon$ approximation guarantee and $O(\frac{k^2}{\epsilon}\log(k)\log^2(n)\log^3(\frac{k}{\epsilon}))$ amortized expected query complexity.

Known (insertion-only) streaming algorithms. The first streaming algorithm for the submodular maximization under the cardinality constraint was developed by Badanidiyuru, Mirzasoleiman, Karbasi, and Krause [10]. In this seminal work, the authors developed a $(2+\epsilon)$-approximation algorithm for this problem using $O(k\epsilon^{-1}\log k)$ space. Later, Kazemi, Mitrovic, Zadimoghaddam, Lattanzi and Karbasi [97] proposed a space streaming algorithm for this problem that improves the space complexity down to $O(k\epsilon^{-1})$.

In a groundbreaking work, Chakrabarti and Kale [43] at IPCO'14 designed a streaming framework for submodular maximization problems under the matroid and matching constraints, as well as other constraints where independent sets are given either by a hypermatching constraint in p-hypergraphs or by the intersection of p matroids. In particular, their streaming framework gives a 4-approximation streaming algorithm for the submodular maximization under the matroid constraint using $O(k)$ space, where k is the rank of the underlying matroid $\mathcal{M}(\mathcal{V},\mathcal{I})$. The approximation ratio was recently improved to 3.15 by Feldman, Liu, Norouzi-Fard, Svensson, and Zenklusen [78].

Later, Chekuri, Gupta, and Quanrud [48] developed one-pass streaming algorithms for (non-monotone) submodular maximization problems under p-matchoid[6] constraint as well as simpler streaming algorithms for the monotone case that have the same bounds as those of Chakrabarti and Kale [43]. (These two works [43, 48] were inspiring works for us as well).

Known streaming algorithms for related submodular problems. For non-monotone submodular objectives, the first streaming result was obtained by Buchbinder, Feldman, and Schwartz [41], who designed a randomized streaming algorithm achieving an 11.197-approximation for the problem of maximizing a non-monotone submodular function subject to a single cardinality constraint.

Chekuri, Gupta, and Quanrud [48] further extended the work of Chakrabarti and Kale by developing $(5p+2+1/p)/(1-\epsilon)$-

[6]A set system $(N,\mathcal{I})$ is p-matchoid if there exists m matroids $(N_1,\mathcal{I}_1),\cdots,(N_m,\mathcal{I}_m)$ such that every element of N appears in the ground set of at most p of these matroids and $\mathcal{I}=\{S\subseteq 2^{\mathcal{N}}:\forall_{1\le i\le m}S\cap N_i\in\mathcal{I}_i\}$.

approximation algorithm for the non-monotone submodular maximization problems under p-matchoid constraints in the insertion-only streaming model. They also devised a deterministic streaming algorithm achieving $(9p + O(\sqrt{p}))/(1-\epsilon)$-approximation for the same problem. Later, Mirzasoleiman, Jegelka, and Krause [123] designed a different deterministic algorithm for the same problem achieving an approximation ratio of $4p + 4\sqrt{p} + 1$.

At NeurIPS'18, Feldman, Karbasi and Kazemi [77] improved these results for monotone and non-monotone submodular maximization under the p-matchoid constraint with respect to the space usage and approximation factor. As an example, their streaming algorithm for non-monotone submodular under p-matchoid achieves $4p + 2 - o(1)$-approximation that improves upon the randomized streaming algorithm proposed in [48].

Known dynamic streaming algorithms. Mirzasoleiman, Karbasi and Krause [124] and Kazemi, Zadimoghaddam and Karbasi [98] proposed dynamic streaming algorithms for the cardinality constraint. In particular, the authors in [124] developed a dynamic streaming algorithm that given a stream of inserts and deletes of elements of an underlying ground set $\mathcal{V}$, $(2+\epsilon)$-approximates the submodular maximization under cardinality constraint using $O((dk\epsilon^{-1}\log k)^2)$ space and $O(dk\epsilon^{-1}\log k)$ average update time, where d is an upper-bound for the number of deletes that are allowed.

The follow-up paper [98] studies approximating submodular maximization under cardinality constraint in three models, (1) centralized model, (2) dynamic streaming where we are allowed to insert and delete (up to d) elements of an underlying ground set $\mathcal{V}$, and (3) distributed (MapReduce) model. In order to design a generic framework for all three models, they compute a coreset for submodular maximization under cardinality constraint. Their coreset has a size of $O(k\log k + d\log^2 k)$. Out of this coreset, we can extract a set S of size at most k whose $f(S)$ in expectation is at least 2-approximation of the optimal solution. The time to extract such a set S from the coreset is $O(dk\log^2 k + d\log^3 k)$.

The algorithms presented in [124] and [98] are dynamic streaming algorithms (not fully dynamic algorithms) whose time complexities depend on the number of deletions (Theorem 1 of the second reference). Therefore, their query complexities will be high if we recompute a solution after each insertion or deletion. Indeed, if the number of deletions is linear in terms of the maximum size of the ground set $\mathcal{V}$, it is in fact better to re-run the known leveling algorithms (say, [129]) after every insertion and deletion. A similar result was recently obtained for the submodular maximization under the matroid constraint. At ICML 2022, Duetting, Fusco, Lattanzi, Norouzi-Fard, Zadimoghaddam [63] presented a streaming $(5.582 + O(\epsilon))$-approximation algorithm for the deletion robust version of this problem, where the number of deletions is known to the algorithm, and they are revealed at the end of the stream. The space usage of their algorithm is $O(k + \epsilon^{-2}d\log(k))$, which is again linear if the number of deletions (d) is linear in terms of the maximum size of the ground set $\mathcal{V}$. This was subsequently improved by Zhang, Tatti, and Gionis [147].

Known MapReduce algorithms. The first distributed algorithm for the cardinality constrained submodular maximization was due to Mirrokni and Zadimoghaddam [122] who gave a 3.70-approximation in 2 rounds without duplication and a 1.834-approximation with significant duplication of the ground set (each element being sent to $\Theta(\frac{1}{\epsilon}\log(\frac{1}{\epsilon}))$ machines). Later, Barbosa, Ene, Nguyen and Ward [60] achieves a $(2+\epsilon)$-approximation in 2 rounds and was the first to achieve a $(\frac{e}{e-1}+\epsilon)$ approximation in $O(\frac{1}{\epsilon})$ rounds. Both algorithms require $\Omega(\frac{1}{\epsilon})$ duplication. [60] mentions that without duplication, the two algorithms could be implemented in $O(\frac{1}{\epsilon}\log(\frac{1}{\epsilon}))$ and $O(\frac{1}{\epsilon^2})$ rounds, respectively.

In a subsequent work, Liu and Vondrak [115] develop a simple thresholding algorithm that with one random partitioning of the dataset (no duplication) achieves the following: In 2 rounds of MapReduce, they obtain a $(2+\epsilon)$-approximation and in $2/\epsilon$ rounds, they achieve $(\frac{e}{e-1}-\epsilon)$-approximation. Their algorithm is inspired by the streaming algorithms that are presented in [105] and [120]. It is also similar to the algorithm of Assadi and Khanna [7] who study the communication complexity of the maximum coverage problem.

2 Dynamic algorithm for submodular matroid maximization

In this section, we present our dynamic algorithm for the submodular maximization problem under the matroid constraint. The pseudocode of our algorithm is provided in Algorithms 1 and 2. The overview of our dynamic algorithm is given in Section 1.2 "Our contribution".

Promoting Elements As we explained in Section 1.2 "Our contribution", a key concept in our algorithm is the notion of *promoting elements*.

DEFINITION 2.1. (PROMOTING ELEMENTS) *Let $L_{1\leq \ell \leq T}$ be a level. We call an element e, a promoting element for the level L_j if*

- ***Property 1:*** $f(I'_\ell + e) - f(I'_\ell) \geq \frac{\epsilon}{10k} \cdot MAX$, *and*
- *One of the following properties hold:*
  - ***Property 2:*** *$I_\ell + e$ is independent set (i.e., $I_\ell + e \in \mathcal{I}$)* or
  - ***Property 3:*** *$I_\ell + e$ is not independent and the minimum weight element $\hat{e} = \arg\min_{e' \in C} w(e')$ of the set $C = \{e' \in I_\ell : I_\ell + e - e' \in \mathcal{I}\}$ satisfies $2w(\hat{e}) \leq f(I'_\ell + e) - f(I'_\ell)$.*

Algorithm 1 MATROIDLEVELING($\mathcal{M}(\mathcal{V}, \mathcal{I}), MAX$)

function INIT(V)
  $I_0 \leftarrow \emptyset, \quad I'_0 \leftarrow \emptyset, \quad R_0 \leftarrow V$
  $R_1 \leftarrow \{e \in R_0 : \text{PROMOTE}(I_0, I'_0, e, w[I_0]) \neq \text{FAIL}\}$
  Invoke MATROIDCONSTRUCTLEVEL($i = 1$)

---

function MATROIDCONSTRUCTLEVEL(i)
  Let P be a random permutation of elements of R_i and $\ell \leftarrow i$
  for e in P **do**
    if PROMOTE($I_{\ell-1}, I'_{\ell-1}, e, w[I_{\ell-1}]$) $\neq$ FAIL **then**
      $y \leftarrow$ PROMOTE($I_{\ell-1}, I'_{\ell-1}, e, w[I_{\ell-1}]$)) and $z \leftarrow \ell$
      Fix the weight $w(e) \leftarrow f(I'_{\ell-1} + e) - f(I'_{\ell-1})$, and set the element $e_\ell \leftarrow e$
      Let $I_\ell \leftarrow (I_{\ell-1} + e) \backslash y, \quad I'_\ell \leftarrow I'_{\ell-1} + e, \quad R_{\ell+1} \leftarrow \emptyset,$ and then $\ell \leftarrow \ell + 1$
    else
      Run binary search to find the lowest $z \in [i, \ell - 1]$ such that PROMOTE($I_z, I'_z, e, w[I_z]$) = FAIL
    for $r \leftarrow i + 1$ **to** z **do**
      $R_r \leftarrow R_r + e$
  return $T \leftarrow \ell - 1$ which is the final ℓ that the for-loop above returns subtracted by one

---

function PROMOTE($I, I', e, w[I]$)
  if $f(I' \cup \{e\}) - f(I') \notin [\frac{\epsilon}{10k} \cdot MAX, MAX]$ **then**
    return FAIL
  if $I + e \in \mathcal{I}$ **then**
    return $\emptyset$
  $C \leftarrow \{e' \in I : I + e - e' \in \mathcal{I}\}$ and let $\hat{e} \leftarrow \arg\min_{e' \in C} w(e')$
  if $2 \cdot w(\hat{e}) \leq f(I' + e) - f(I')$ **then**
    return $\{\hat{e}\}$
  else
    return FAIL

We define the function PROMOTE($I_\ell, I'_\ell, e, w[I_\ell]$) for an element $e \in V$ with respect to the level L_ℓ which

- returns $\emptyset$ if properties 1 and 2 hold;
- returns $\hat{e}$ if properties 1 and 3 hold;
- returns FAIL otherwise.

Subroutine PROMOTE in Algorithm 1 implements this function. This subroutine checks if an element $e \in V$ is a promoting element for a level L_ℓ or not. In case that e is a promoting element for L_ℓ, the subroutine PROMOTE finds an element e' (if it exists) that satisfies Property 3 of definition 2.1 and replaces it by e.

Our leveling algorithm consists of three subroutines, INIT, MATROIDCONSTRUCTLEVEL, and PROMOTE. We explained in above Subroutine PROMOTE. In Subroutine INIT, we first initialize I_0 and I'_0 as empty set and set R_0 to the ground set V. We then let R_1 be all elements of the set R_0 that are promoting with respect to L_0. Observe that since I_0 and I'_0 are empty sets, if an element e filtered out from the level L_0, i.e., $e \in L_0$ but $e \notin L_1$, then e was filtered because of Property 1. Finally, we invoke MATROIDCONSTRUCTLEVEL for the level L_1, to build all the remaining levels. Subroutine MATROIDCONSTRUCTLEVEL implements our leveling algorithm that we gave an overview of it in Section 1.2 "Our contribution".

Algorithm 2 MatroidUpdates($\mathcal{M}(\mathcal{V}, \mathcal{I})$, MAX)

function Delete(v)
    $R_0 \leftarrow R_0 - v$
    for $i \leftarrow 1$ **to** T **do**
        if $v \notin R_i$ **then**
            break
        $R_i \leftarrow R_i - v$
        if $e_i = v$ **then**
            Invoke MatroidConstructLevel(i)
            break

---

function Insert(v)
    $R_0 \leftarrow R_0 + v$.
    for $i \leftarrow 1$ **to** $T + 1$ **do**
        if Promote($I_{i-1}, I'_{i-1}, v, w[I_{i-1}]$) = Fail **then**
            break
        $R_i \leftarrow R_i + v$.
        Let $p_i = 1$ with probability $\frac{1}{|R_i|}$, and otherwise $p_i = 0$
        if $p_i = 1$ **then**
            $e_i \leftarrow v, \quad w(e_i) \leftarrow f(I'_{i-1} + v) - f(I'_{i-1}), \quad y \leftarrow$ Promote($I_{i-1}, I'_{i-1}, v, w[I_{i-1}]$)
            $I_i \leftarrow I_{i-1} + v - y, \quad I'_i \leftarrow I'_i + v$
            $R_{i+1} = \{e' \in R_i :$ Promote($I_i, I'_i, e', w[I_i]$) $\neq$ Fail$\}$
            MatroidConstructLevel($i + 1$)
            break

Relaxing MAX assumption. Our dynamic algorithm assumes the maximum value $\max_{e \in V} f(e)$ is given as a parameter. However, in reality, the maximum value is not known in advance and it may change after every insertion or deletion. To remove this assumption, we run parallel instances of our dynamic algorithm for different guesses of the maximum value MAX_t at any time t of the sequence $\mathcal{S}_t$, such that $\max_{e \in V_t} f(e) \in (MAX_t/2, MAX_t]$ in one of the runs. Recall that V_t is the set of elements that have been inserted but not deleted from the beginning of the sequence till time t. These guesses that we take are 2^i where $i \in \mathbb{Z}$. If ρ is the ratio between the maximum and minimum non-zero possible value of an element in V, then the number of parallel instances of our algorithm will be $O(\log \rho)$. This incurs an extra $O(\log \rho)$-factor in the query complexity of our dynamic algorithm.

Algorithm 3 Unknown MAX

Let $\mathcal{A}_i$ be the instance of our dynamic algorithm, for which $MAX = 2^i$

---

function UpdateWithoutKnowingMAX(e)
    for each $i \in \left[\lceil \log f(e) \rceil, \left\lfloor \log\left(\frac{10k}{\epsilon} \cdot f(e)\right)\right\rfloor\right]$ **do**   ▷ $\frac{\epsilon}{10k} \cdot 2^i \leq f(e) \leq 2^i$
        Invoke Update(e) for instance $\mathcal{A}_i$

Next, we show how to replace this extra factor with an extra factor of $O(\log(k/\epsilon))$ which is independent of ρ. We use the well-known technique that has been also used in [109]. In particular, for every element e, we add it to those instances i for which we have $\frac{\epsilon}{10k} \cdot 2^i \leq f(e) \leq 2^i$. The reason is if the maximum value of V_t is within the range $(2^{i-1}, 2^i]$ and $f(e) > 2^i$, then $f(e)$ is greater than the maximum value and can safely be ignored for the instance i that corresponds to the guess 2^i. On the other hand, we can safely ignore all elements e whose $f(e) < \frac{\epsilon}{10k} \cdot 2^i$, since these elements will never be a promoting element in the run with $MAX = 2^i$. This essentially means that every element e is added to at most $O(\log(k/\epsilon))$ parallel instances. Thus, after every insertion or deletion, we need to update only $O(\log(k/\epsilon))$ instances of our dynamic algorithm.

3 Analysis of dynamic algorithm for submodular matroid

In this section, we prove the correctness of our MatroidConstructLevel, Insert, and Delete algorithms. We will also compute the query complexity of each one of them. To analyze our randomized algorithm, for any variable x in our pseudo-code, we use $\mathbf{x}$ to denote it as a random variable and use x itself to denote its value in an execution. The most frequently used random variables in our analysis are as follows:

- We denote by $\mathbf{e}_i$ the random variable corresponding to the element e_i picked at level L_i.
- We denote by $\mathbf{R}_i$ the random variable that corresponds to the set R_i.
- The random variable $\mathbf{T}$ corresponds to T, which is the index of the last non-empty level created. Indeed, for a level L_i to be existent and non-empty, $\mathbf{T} \geq i$ should hold.
- We define $H_i = (e_1, \ldots, e_{i-1}, R_0, \ldots, R_i)$ as *the partial configuration up to the level* L_i. Note that R_i is included in this definition, while e_i is not. $\mathbf{H}_i := (\mathbf{e}_1, \ldots, \mathbf{e}_{i-1}, \mathbf{R}_0, \mathbf{R}_1, \ldots, \mathbf{R}_i)$ is the random variable corresponding to the partial configuration H_i.

We break the analysis of our algorithm into a few steps.

Step 1: Analysis of binary search. In the first step, we prove that the binary search that we use to speed up the process of finding the right levels for non-promoting elements works. Indeed, we prove that if $e \in V$ is a promoting element for a level L_{z-1}, it is promoting for all levels $L_{r \leq z-1}$ and if e is not promoting for the level L_z, it is not promoting for all levels $L_{r \geq z}$. Therefore, because of this monotonicity property, we can do a binary search to find the smallest $z \in [i, \ell - 1]$ so that e is promoting for the level L_{z-1}, but it is not promoting for the level L_z. Additionally, we show that checking whether e is promoting for a level L_z can be done with $O(\log(k))$ queries using a binary search argument.

Step 2: Maintaining invariants. We define six invariants, and we show that these invariants *hold* when Init is run, and our whole data structure gets built, *and are preserved* after every insertion and deletion of an element.

Invariants:

1. **Level invariants.**
   1.1 **Starter.** $R_0 = V$ and $I_0 = I'_0 = \emptyset$
   1.2 **Survivor.** For $1 \leq i \leq T + 1$, $R_i = \{e \in R_{i-1} - e_{i-1} : \textsc{Promote}(I_{i-1}, I'_{i-1}, e, w[I_{i-1}]) \neq \textsc{Fail}\}$
   1.3 **Independent.** For $1 \leq i \leq T$, $I_i = I_{i-1} + e_i - \textsc{Promote}(I_{i-1}, I'_{i-1}, e_i, w[I_{i-1}])$, and $I'_i = \cup_{j \leq i} I_j$
   1.4 **Weight.** For $1 \leq i \leq T$, $e_i \in R_i$ and $w(e_i) = f(I'_{i-1} + e_i) - f(I'_{i-1})$
   1.5 **Terminator.** $R_{T+1} = \emptyset$
2. **Uniform invariant.** For all $i \geq 1$, conditioned on the random variables $\mathbf{T}$ and $\mathbf{H}_i$, the element e_i is chosen uniformly at random from the set R_i. That is, $\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i \text{ and } \mathbf{H}_i = H_i] = \frac{1}{|R_i|} \cdot \mathbb{1}[e \in R_i]$.

The survivor invariant says that all elements that are added to R_i at a level L_i are promoting elements for that level. In other words, those elements of the set $R_{i-1} - e_{i-1}$ that are not promoting will be filtered out and not be seen in R_i. The terminator invariant shows that the recursive construction of levels stops when the survivor set becomes empty. The independent invariant shows that the sets I_i are independent sets of the matroid $\mathcal{M}(\mathcal{V}, \mathcal{I})$, and I'_i is equal to the union of $I_1, \ldots, I_i$. The weight invariant explains that the weight of every element e_i added to the independent set I_i is defined with respect to the marginal gain it adds to the set I'_{i-1}, and it is fixed later on. Intuitively, the level invariants provide the approximation guarantee.

The uniform invariant asserts that, conditioned on $\mathbf{T} \geq i$ which means that L_i is a non-empty level and $\mathbf{H}_i = H_i$, which implies that $e_1, \cdots, e_{i-1}$ are chosen and R_i is well-defined, the element $\mathbf{e}_i$ is uniform random variable over the set R_i. That is, $\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i \text{ and } \mathbf{H}_i = H_i] = \frac{1}{|R_i|} \cdot \mathbb{1}[e \in R_i]$. Intuitively, this invariant provides us with the randomness that we need to fool the adversary in the (fully) dynamic model which in turn helps us to develop a dynamic algorithm for the submodular matroid maximization.

Step 3: Query complexity. In the third part of the proof, we show that if the **uniform** invariant holds, we can bound the worst-case expected query complexity of the leveling algorithm, and later, the worst-case expected query complexity of the insertion and deletion operations.

Step 4: Approximation guarantee. Finally, in the last step of the proof, we show that if the survivor, terminator, independent and weights invariants hold, we can report an independent set $I_T \in \mathcal{I}$ whose submodular value is an $(4 + \epsilon)$-approximation of the optimal value.

3.1 Monotone property and binary search argument Recall that we defined the function $\textsc{Promote}(I_j, I'_j, e, w[I_j])$ for an element $e \in V$ with respect to the level L_j which

- returns $\emptyset$ if properties 1 and 2 hold;

- returns $\hat{e}$ if properties 1 and 3 hold;
- returns Fail otherwise.

Here properties 1, 2, and 3 are the ones that we defined in Definition 2.1. Recall that if the first two cases occur, we say that e is a promoting element with respect to the level L_j. In this section, we consider a boolean version of the function Promote$(I_j, I'_j, e, w[I_j])$. We denote this boolean function by BoolPromote(e, L_j) which is $True$ if either of the first two cases happen. That is, when Promote$(I_j, I'_j, e, w[I_j])$ returns either $\emptyset$ or $\hat{e}$; otherwise, BoolPromote(e, L_j) returns $False$.

Lemma 3.1. *Let L_j be an arbitrary level of the Algorithm* DynamicMatroid*, where $1 \leq j \leq T$. Let e be an arbitrary element of the ground set. If* BoolPromote(e, L_{j-1}) *returns $False$, then* BoolPromote(e, L_j) *returns $False$.*

Suppose for the moment that this lemma is correct. Then by applying a simple induction, we can show the function BoolPromote(e, L_j) is monotone which means that the function Promote$(I_j, I'_j, e, w[I_j])$ is monotone. Thus, for every arbitrary element e, it is possible to perform a binary search on the interval $[i, \ell - 1]$ to find the smallest $z \in [i, \ell - 1]$ such that BoolPromote$(e, L_{z-1}) = True$ and BoolPromote$(e, L_z) = False$.

Now we prove the lemma.

Proof. Suppose that BoolPromote(e, L_{j-1}) returns $False$. It means that either property 1 or both properties 2 and 3 do(es) not hold. If property 1 does not hold, then $f(I'_{j-1} + e) - f(I'_{j-1}) < \frac{\varepsilon}{10k} \cdot MAX$. Since $I'_{j-1} \subseteq I'_j$ and f is submodular, we have $f(I'_j + e) - f(I'_j) \leq f(I'_{j-1} + e) - f(I'_{j-1}) < \frac{\varepsilon}{10k} \cdot MAX$, which means that BoolPromote$(e, L_j) = False$.

For the remainder of the proof, we assume that both properties 2 and 3 do not hold. This means that $I_{j-1} + e$ is not independent, and for the minimum weight element $\hat{e} := \arg\min_{e' \in C} w(e')$ of the set $C := \{e' \in I_{j-1} : I_{j-1} + e - e' \in \mathcal{I}\}$, we have $f(I'_{j-1} + e) - f(I'_{j-1}) < 2w(\hat{e})$. Now, let us consider I_j. We consider two cases: $|I_j| = |I_{j-1}| + 1$ and $|I_j| = |I_{j-1}|$.

For the first case, we have $I_j = I_{j-1} + e_j$. Thus, we have $I_{j-1} \subseteq I_j$. Now, let us consider the element e. For the set $C := \{e' \in I_{j-1} : I_{j-1} + e - e' \in \mathcal{I}\}$, we have $C \subseteq I_{j-1} \subseteq I_j$ which means that we have $C + e \subseteq I_j + e$. Note that I_j is an independent set, and so, $C + e$ is the only circuit (dependent set) of $I_j + e$ according to Lemma 1.2. Recall that $f(I'_{j-1} + e) - f(I'_{j-1}) < 2w(\hat{e})$ where $\hat{e}$ is the minimum weight element $\hat{e} := \arg\min_{e' \in C} w(e')$. Since $I_{j-1} \subseteq I_j$, then by the submodularity of the function f, we have

$$f(I'_j + e) - f(I'_j) \leq f(I'_{j-1} + e) - f(I'_{j-1}) < 2w(e') \ .$$

Hence, BoolPromote(e, L_j) returns $False$ in this case.

For the second case, we have $I_j = I_{j-1} - \hat{e}_j + e_j$. This means that $I_{j-1} + e_j$ is not an independent set. Thus, the set $C' := \{e' \in I_{j-1} : I_{j-1} + e_j - e' \in \mathcal{I}\}$ has a minimum weight element $\hat{e}_j$ that is replaced by e_j to obtain the independent set I_j.

Now, we consider two subcases. Case (I) is $\hat{e}_j \in C$ and Case (II) is $\hat{e}_j \notin C$.

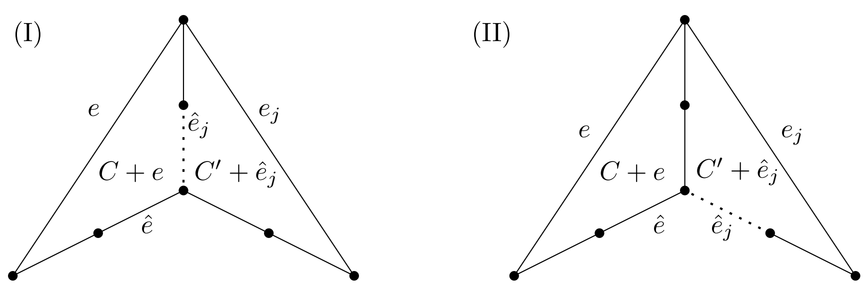

Figure 2: Illustration of $I_j + e$ for the subcases (I) and (II) in Lemma 3.1. $C + e$ and $C' + \hat{e}_j$ are circuits. Case (I) is $\hat{e}_j \in C$. Then there is a circuit $C'' \subseteq (C + e) \cup (C' + e_j) - \hat{e}_j$. Case (II) is $\hat{e}_j \notin C$. Then $C + e \subseteq I_j + e$.

First, we consider Case (I) which is $\hat{e}_j \in C$. Thus, $\hat{e}_j \in C \cap C'$. Note that $C \subseteq I_{j-1}$ and $e_j \notin I_{j-1}$, then $e_j \notin C$. Observe that $e_j \in C'$, since otherwise, $C' \subseteq I_{j-1}$, and so, I_{j-1} is not an independent set which cannot be the case. Therefore, $e_j \in C' \setminus C$, that implies $C \neq C'$. Since $C \neq C'$ and $\hat{e}_j \in (C + e) \cap (C' + e_j)$, there is a circuit $C'' \subseteq (C + e) \cup (C' + e_j) - \hat{e}_j$ according to Lemma 1.1. In addition, $(C + e) \cup (C' + e_j) \subseteq I_{j-1} + e + e_j = I_j + e - \hat{e}_j$. Since $\hat{e}_j \notin C''$, we then have $C'' \subseteq I_j + e$. Recall that $\hat{e}$ and $\hat{e}_j$ are the minimum weight element in C and C', respectively. Since $\hat{e}_j \in C$, then $w(\hat{e}) \leq w(\hat{e}_j)$.

Let e'' be the minimum weight element in $C'' - e$. Since $C'' \subseteq (C + e) \cup (C' + e_j)$ and $w(e_j) > w(\hat{e}_j)$, we have $w(e'') \geq \min(w(\hat{e}), w(\hat{e}_j)) = w(\hat{e})$. Since $I'_{j-1} \subseteq I'_j$ and f is a submodular function, we obtain the following:

$$f(I'_j + e) - f(I'_j) \leq f(I'_{j-1} + e) - f(I'_{j-1}) < 2 \cdot w(\hat{e}) \leq 2 \cdot w(e'') \ .$$

This essentially means that $C'' \subseteq I_j + e$ and $f(I'_j + e) - f(I'_j) < 2 \cdot w(e'')$, where e'' is the minimum weight element in C''. Thus, BoolPromote(e, L_j) returns *False*.

Finally, we consider Case (II) which is $\hat{e}_j \notin C$. In this case, $C + e \subseteq I_{j-1} - \hat{e}_j + e \subseteq I_j + e$. Note that $C + e$ is the only circuit of $I_j + e$ by Lemma 1.2. Recall $f(I'_{j-1} + e) - f(I'_{j-1}) < 2 \cdot w(\hat{e})$ and $I'_{j-1} \subseteq I'_j$. Hence, by the submodularity of f we have $f(I'_j + e) - f(I'_j) \le f(I'_{j-1} + e) - f(I'_{j-1}) < 2 \cdot w(\hat{e})$. Thus, BoolPromote$(e, L_j)$ returns *False* proving the lemma. □

Lemma 3.2. *Let $I \in \mathcal{I}$ be an independent set and e be an element such that $I \cup \{e\} \notin \mathcal{I}$. Define $C := \{e' : I + e - e' \in \mathcal{I}\}$. Let $w : I \cup \{e\} \to \mathbb{R}^{\ge 0}$ be an arbitrary weight function and define $\hat{e} := \arg\min_{e' \in C} w(e')$. The element $\hat{e}$ can be found using at most $O(\log(|I|))$ oracle queries.*

Proof. Let $e_1, \dots, e_{|I|+1}$ denote an ordering of $I \cup \{e\}$ such that $w(e_1) \ge w(e_2) \cdots \ge w(e_{|I|+1})$. Let i denote the smallest index such that $\{e_1, \dots, e_i\} \notin \mathcal{I}$. Such an index exists because $\{e_1, \dots, e_{|I|+1}\} = I \cup \{e\} \notin \mathcal{I}$. We claim that $\hat{e} = e_i$. We note that the element e_i can be found using a binary search over $[|I| + 1]$ because for any j, if $\{e_1, \dots, e_j\} \notin \mathcal{I}$, then $\{e_1, \dots, e_{j+1}\} \notin \mathcal{I}$ as well.

To prove this, we first claim that $e_i \in C$. To see why this holds, we first observe that since $\{e_1, \dots, e_{i-1}\}$ is independent but $\{e_1, \dots, e_i\}$ is not, we have $e_i \in \text{Span}(\{e_1, \dots, e_{i-1}\}) \subseteq \text{Span}(I + e - e_i)$. Therefore, since $e_j \in \text{Span}(I + e - e_i)$ for all $j \ne i$, we have $I + e \subseteq \text{Span}(I + e - e_i)$, which implies

$$rank(I + e - e_i) \ge rank(I + e) \ge rank(I) = |I| = |I + e - e_i|,$$

which implies $I + e - e_i \in \mathcal{I}$ as claimed.

We need to show that for any $e' \in C$, we have $w(e_i) \le w(e')$. Assume for contradiction that $w(e') < w(e_i)$. It follows that $e' = e_j$ for some $j > i$. By definition of C, we must have $I + e - e_j \in \mathcal{I}$, which implies $\{e_1, \dots, e_{j-1}\} \in \mathcal{I}$. Since $i < j$, this further implies $\{e_1, \dots, e_i\} \in \mathcal{I}$, which is not possible by definition of i. □

3.2 Correctness of invariants after MatroidConstructLevel is called

In this section, we focus on the previously defined invariants at the end of the execution of the algorithm MatroidConstructLevel(j). We first provide a definition explaining what we mean by stating that level invariants partially hold.

Definition 3.1. *For $j \ge 1$, we say that* the level invariants partially hold for the first j levels *if the followings hold.*
1. ***Starter.*** *$R_0 = V$ and $I_0 = I'_0 = \emptyset$*
2. ***Survivor.*** *For $1 \le i \le j$, $R_i = \{e \in R_{i-1} - e_{i-1} : \text{Promote}(I_{i-1}, I'_{i-1}, e, w[I_{i-1}]) \ne \text{Fail}\}$*
3. ***Independent.*** *For $1 \le i \le j-1$, $I_i = I_{i-1} + e_i - \text{Promote}(I_{i-1}, I'_{i-1}, e_i, w[I_{i-1}])$, and $I'_i = \cup_{j \le i} I_j$*
4. ***Weight.*** *For $1 \le i \le j-1$, $e_i \in R_i$ and $w(e_i) = f(I'_{i-1} + e_i) - f(I'_{i-1})$*

Next, we have the following theorem, in which we ensure that all level invariants hold after the execution of MatroidConstructLevel(j) given the assumption that level invariants partially hold for the first j levels when MatroidConstructLevel(j) is invoked. This theorem will be of use in the following sections in showing that level invariants hold after each update. It can also independently prove that level invariants hold after Init is run.

Theorem 3.1. *If before calling* MatroidConstructLevel(j)*, the level invariants partially hold for the first j levels, then after the execution of* MatroidConstructLevel(j)*, level invariants fully hold.*

Proof. Considering that the starter invariant holds by the assumption of the theorem and needs no further proof, we have broken the proof of this theorem into four lemmas, each considering one of the survivor, independent, weight, and terminator invariants separately. These mentioned lemmas and their proofs can be found in detail in Appendix A as Lemmas A.1, A.2, A.3, and A.4 in Section A.1. □

Finally, we prove a lemma that says knowing that the level invariants are going to hold after the execution of MatroidConstructLevel(j), a modified version of uniform invariant will also hold after this execution. We use this lemma in the next sections to prove that the uniform invariant holds after each update. It also shows that uniform invariant holds after Init is run since the previous theorem had proved that level invariants would hold.

Lemma 3.3. (Uniform invariant) *If* MatroidConstructLevel(j) *is invoked and the level invariants are going to hold after its execution, then for any $i \ge j$ we have $\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \ge i \text{ and } \mathbf{H}_i = H_i] = \frac{1}{|R_i|} \cdot \mathbb{1}[e \in R_i]$.*

Proof. At the beginning of MatroidConstructLevel(j), we take a random permutation of elements in R_j. Making a random permutation is equivalent to sampling all elements without replacement. In other words, instead of fixing a random permutation P of R_j and iterating through P in Line 7, we can repeatedly sample a random element e from the unseen elements of R_j until we have seen all of the elements. Hence, in the following proof, we assume our algorithm uses sampling without replacement.

Given this view, we make the following claims.

Observation 1. e_i is the first element of R_i seen in the permutation.

This is because before e_i is seen, the value of ℓ is at most i. It is also clear from the algorithm that when an element e is considered, it can only be added to sets R_x for $x \leq \ell$, both when $y =$ Fail and when $y \neq$ Fail. Furthermore, e can only be added to R_ℓ if $e = e_\ell$. Therefore, no element can be added to R_i before e_i is seen.

Observation 2. Once $e_1, \ldots, e_{i-1}$ have been seen, the set R_i is uniquely determined.

Note that R_i is uniquely determined *even though the algorithm has not observed its elements yet*. This is because regardless of the randomness of MatroidConstructLevel(j), the level invariants will hold after its execution. This implies that the content of the set R_i only depends on the value of $(\mathbf{e}_1, \ldots, \mathbf{e}_{i-1})$, which is not going to change after it is set to be equal to $(e_1, \ldots, e_{i-1})$.

Let the random variable $\mathbf{M}_i$ denote the sequence of elements that our algorithm observes until setting $\mathbf{e}_{i-1}$ to be e_{i-1}, including e_{i-1} itself. In other words, if e_{i-1} is the x-th element of the permutation P, M_i is the first x elements of P.

Based on the above facts, conditioned on $\mathbf{M}_i = M_i$, **(a)** the value of $\mathbf{R}_i$, or in other words R_i is uniquely determined. **(b)** e_i is going to be the first element of R_i that the algorithm observes. Therefore, since we assumed that the algorithm uses sampling without replacement, $\mathbf{e}_i$ is going to have a uniform distribution over R_i, i.e.,

$$\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{M}_i = M_i] = \frac{1}{|R_i|} \mathbb{1}[e \in R_i] \quad .$$

By the law of total probability, we have

$$\mathbb{P}[\mathbf{e}_i = e_i | \mathbf{T} \geq i, \mathbf{H}_i = H_i] = \mathbb{E}_{M_i}[\mathbb{P}[\mathbf{e}_i = e_i | \mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{M}_i = M_i]] \quad ,$$

where the expectation is taken over all M_i with positive probability.

Also, note that knowing that $\mathbf{M}_i = M_i$ uniquely determines the value of $\mathbf{H}_i$ as well. This is because M_i includes $(e_1, \ldots, e_{i-1})$ and, with similar reasoning to what we used for Observation 2, we can say that $R_1, \ldots, R_i$ are uniquely determined by $(e_1, \ldots, e_{i-1})$.

Since we are only considering M_i with positive probability, and $\mathbf{H}_i$ is a function of $\mathbf{M}_i$ given the discussion above, all the forms of M_i that we consider in our expectation are the ones that imply $\mathbf{H}_i = H_i$. Therefore, we can drop the condition $\mathbf{H}_i = H_i$ from the condition $\mathbf{H}_i = H_i, \mathbf{M}_i = M_i$, which implies

$$\mathbb{P}[\mathbf{e}_i = e_i | \mathbf{T} \geq i, \mathbf{H}_i = H_i] = \mathbb{E}_{M_i}[\mathbb{P}[\mathbf{e}_i = e_i | \mathbf{T} \geq i, \mathbf{M}_i = M_i]] = \mathbb{E}_{M_i}\left[\frac{1}{|R_i|} \mathbb{1}[e_i \in R_i]\right] = \frac{1}{|R_i|} \mathbb{1}[e_i \in R_i] \quad ,$$

as claimed. ∎

3.3 Correctness of invariants after an update In our dynamic model, we consider a sequence $\mathcal{S}$ of updates to the underlying ground set V where at time t of the sequence $\mathcal{S}$, we observe an update which can be the deletion of an element $e \in V$ or insertion of an element $e \in V$. We assume that an element e can be deleted at time t, if it is in V meaning that it was not deleted after the last time it was inserted.

We use several random variables for our analysis, including $\mathbf{e}_i$, $\mathbf{R}_i$, $\mathbf{T}$, and $\mathbf{H}_i$. Upon observing an update at time t, we should distinguish between each of these random variables and their corresponding values before and after the update. To do so, we use the notations $\mathbf{Y}^-$ and Y^- to denote a random variable and its value before time t when e is either deleted or inserted, and we keep using $\mathbf{Y}$ and Y to denote them at the current time after the execution of update. As an example, $\mathbf{H}_i^- := (\mathbf{e}_1^-, \ldots, \mathbf{e}_{i-1}^-, \mathbf{R}_0^-, \mathbf{R}_1^-, \ldots, \mathbf{R}_i^-)$ is the random variable that corresponds to the partial configuration $H_i^- = (e_1^-, \ldots, e_{i-1}^-, R_0^-, \ldots, R_i^-)$.

3.3.1 Correctness of invariants after every insertion We first consider the case when the update at time t of the sequence $\mathcal{S}$ is an insertion of an element v. In this section, we prove the following theorem.

Theorem 3.2. *If before the insertion of an element v, the level invariants and uniform invariant hold, then they also hold after the execution of* Insert(v)*.*

We break the proof of this theorem into Lemmas 3.4 and 3.5. Note that we use Lemma 3.4 in the proof of Lemma 3.5. However, Lemma 3.5 would not be used in the proof of 3.4, so no loop would form when combined to prove the theorem. Proof of Lemma 3.4 is written in detail in Appendix A Section A.2.

Lemma 3.4. (Level invariants) *If before the insertion of an element v the level invariants (i.e., starter, survivor, independent, weight, and terminator) hold, then they also hold after the execution of* Insert(v)*.*

Lemma 3.5. (Uniform invariant) *If before the insertion of an element v the level and uniform invariants hold, then the uniform invariant also holds after the execution of* Insert(v).

Proof. By the assumption that the uniform invariant holds before the insertion of the element v, we mean that for any arbitrary i and any arbitrary element e, the following holds:

$$\mathbb{P}\left[\mathbf{e}_i^- = e | \mathbf{T}^- \geq i, \mathbf{H}_i^- = H_i^-\right] = \frac{1}{|R_i^-|} \cdot \mathbb{1}\left[e \in R_i^-\right] \quad .$$

We aim to prove that given our assumptions, after the execution of Insert(v), for each arbitrary i and each arbitrary element e, we have

$$\mathbb{P}\left[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i\right] = \frac{1}{|R_i|} \cdot \mathbb{1}\left[e \in R_i\right] \quad .$$

Note that $\mathbb{P}[\mathbf{e}_i = e|\mathbf{T} \geq i, \mathbf{H}_i = H_i]$, is only defined when $\mathbb{P}[\mathbf{T} \geq i, \mathbf{H}_i = H_i] > 0$, which means that given the input and considering the behavior of our algorithm including its random choices, it is possible to reach a state where $\mathbf{T} \geq i$ and $\mathbf{H}_i = H_i$. In this proof, we use $\mathbf{p}_i$ to denote to the variable p_i used in the Insert as a random variable.

Fix any arbitrary i and any arbitrary element e. Since $\mathbf{H}_i^- = (\mathbf{e}_1^-, \ldots, \mathbf{e}_{i-1}^-, \mathbf{R}_0^-, \mathbf{R}_1^-, \ldots, \mathbf{R}_i^-)$ refers to our data structure levels before the insertion of the element v, it is clear that the following facts hold about $\mathbf{H}_i^-$.

Fact 3.1. *For any $j < i$, $\mathbf{e}_j^- \neq v$.*

Fact 3.2. *For any $j \leq i$, $v \notin \mathbf{R}_j^-$.*

We consider the following cases based on which of the following holds for $H_i = (e_1, \ldots, e_{i-1}, R_0, R_1, \ldots, R_i)$:

- Case 1: If the $e_j = v$ for some $j < i$.
- Case 2: If $v \notin \{e_1, \ldots, e_{i-1}\}$.

We handle these two cases separately (Lemma 3.1 for the first case and Lemma 3.3 for the second case). We show that, no matter the case, $\mathbb{P}[\mathbf{e}_i = e|\mathbf{T} \geq i, \mathbf{H}_i = H_i]$ is equal to $\frac{1}{|R_i|} \cdot \mathbb{1}[e \in R_i]$, which completes the proof of the Lemma.

Claim 3.1. *If H_i is such that there is a $1 \leq j < i$ that $e_j = v$, then $\mathbb{P}[\mathbf{e}_i = e|\mathbf{T} \geq i, \mathbf{H}_i = H_i] = \frac{1}{|R_i|} \cdot \mathbb{1}[e \in R_i]$.*

Proof. We know that, $\mathbf{p_j}$ must have been equal to 1, as otherwise, instead of having $\mathbf{e_j} = e_j = v$, we would have had $\mathbf{e}_j = \mathbf{e}_j^-$, which would not have been equal to v as stated in Fact 3.1. According to our algorithm, since $\mathbf{p_j}$ has been equal to 1, we have invoked MatroidConstructLevel($j+1$). By Lemma 3.4, we know that the level invariants hold at the end of the execution of Insert, which is also the end of the execution of MatroidConstructLevel($j+1$). Thus, Lemma 3.3, proves that $\mathbb{P}[\mathbf{e}_i = e|\mathbf{T} \geq i, \mathbf{H}_i = H_i] = \frac{1}{|R_i|} \cdot \mathbb{1}[e \in R_i]$. ◻

Claim 3.2. *Assume that H_i is such that $e_j \neq v$ for any $1 \leq j < i$ and define H_i^- based on H_i as $H_i^- := (R_0 \backslash \{v\}, \ldots, R_i \backslash \{v\}, e_1, \ldots, e_{i-1})$. The events $[\mathbf{T} \geq i, \mathbf{H}_i = H_i]$ and $[\mathbf{T}^- \geq i, \mathbf{H}_i^- = H_i^-, \mathbf{p_1} = 0, \ldots, \mathbf{p_{i-1}} = 0]$ are equivalent and imply each other, thusly they are interchangeable.*

Proof. First, we show that if $\mathbf{T} \geq i, \mathbf{H}_i = H_i$, then $\mathbf{T}^- \geq i, \mathbf{H}_i^- = H_i^-, \mathbf{p_1} = 0, \ldots, \mathbf{p_{i-1}} = 0$. Considering that case 2 holds for H_i, $\mathbf{H}_i = H_i$, means that for any $j < i$, $\mathbf{e}_i \neq v$, which means there is no $j < i$ with $\mathbf{p_j} = 1$. Note that if $\mathbf{p_j} = 1$, then we would have set $\mathbf{e_j}$ to be equal to v, and we would have invoked MatroidConstructLevel($j+1$). Thus, in addition to knowing that for any $j < i$, $\mathbf{p_j} = 0$, we also know that, we have not invoked MatroidConstructLevel($j+1$) for any $j < i$. As for any $j < i$, $\mathbf{p_j} = 0$ and MatroidConstructLevel($j+1$) was not invoked, we have the following results:

1. Level i also existed before the insertion of v, i.e. $\mathbf{T}^- \geq i$.
2. We have made no change in the values of $(\mathbf{e_1}, \ldots, \mathbf{e_{i-1}})$, and they still have the values they had before the insertion of v, i.e. for any $j < i$, $\mathbf{e_j} = \mathbf{e_j^-}$, and so $\mathbf{e_j^-} = e_j$.
3. All the change we might have made in our data structure is limited to adding the element v to a subset of $\{\mathbf{R_0^-}, \ldots, \mathbf{R_i^-}\}$. Hence, for any $j \leq i$, whether $\mathbf{R_j}$ is equal to $\mathbf{R_j^-}$ or $\mathbf{R_j^-} \cup \{v\}$, $\mathbf{R_j^-} = \mathbf{R_j} \backslash \{v\} = R_j \backslash \{v\}$.

So far, we have proved that throughout our algorithm, we reach the state, where $\mathbf{T} \geq i, \mathbf{H}_i = H_i$, only if $\mathbf{T}^- \geq i, \mathbf{H}_i^- = H_i^-, \mathbf{p_1} = 0, \ldots, \mathbf{p_{i-1}} = 0$.

We know that in our insertion algorithm, there is not any randomness other than setting the value of $\mathbf{p_j}$ as long as we have not invoked MatroidConstructLevel, which only happens when for a j, $\mathbf{p_j}$ is set to be 1. It means that the value of $\mathbf{H}_i$ can be determined uniquely if we know the value of $\mathbf{H}_i^-$, and we know that $\mathbf{p_1}, \ldots, \mathbf{p_{i-1}}$ are all equal to 0. Since we have assumed that $\mathbf{T} \geq i, \mathbf{H}_i = H_i$ is a valid and reachable state in our algorithm, $\mathbf{T}^- \geq i, \mathbf{H}_i^- = H_i^-$ must have been a reachable state as well. Plus, $\mathbf{T}^- \geq i, \mathbf{H}_i^- = H_i^-, \mathbf{p_1} = 0, \ldots, \mathbf{p_{i-1}} = 0$, should imply that $\mathbf{T} \geq i$ and $\mathbf{H}_i = H_i$. Otherwise, $\mathbf{T} \geq i, \mathbf{H}_i = H_i$ could not be a reachable state, which is in contradiction with our assumption. ◻

Claim 3.3. *If H_i is such that $e_j \neq v$ for any $1 \leq j < i$, then $\mathbb{P}[\mathbf{e}_i = e|\mathbf{T} \geq i, \mathbf{H}_i = H_i] = \frac{1}{|R_i|} \cdot \mathbb{1}[e \in R_i]$.*

Proof. Define H_i^- based on H_i as $H_i^- := (R_0 \backslash \{v\}, \ldots, R_i \backslash \{v\}, e_1, \ldots, e_{i-1})$.

We calculate $\mathbb{P}[\mathbf{e}_i = e|\mathbf{T} \geq i, \mathbf{H}_i = H_i]$. As stated above, considering that Case 2 holds for H_i, we know that $\mathbf{T} \geq i, \mathbf{H}_i = H_i$

implies that MatroidConstructLevel has not been invoked for any $j < i$. Thus, the value of $\mathbf{e}_i$ will be determined based on the random variable $\mathbf{p_i}$. And we have:

$$\mathbb{P}\left[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i\right] = \sum_{p_i \in \{0,1\}} \left(\mathbb{P}\left[\mathbf{p_i} = p_i | \mathbf{T} \geq i, \mathbf{H}_i = H_i\right] \cdot \mathbb{P}\left[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{p_i} = p_i\right]\right) \ .$$

According to the algorithm, if $v \in H_i$, then $\mathbb{P}\left[\mathbf{p_i} = 1 | \mathbf{T} \geq i, \mathbf{H}_i = H_i\right]$ is equal to $\frac{1}{|R_i|}$. Otherwise, if $v \notin H_i$, then $\mathbf{p_i}$ would be zero by default, and $\mathbb{P}\left[\mathbf{p_i} = 1 | \mathbf{T} \geq i, \mathbf{H}_i = H_i\right] = 0$. Hence, we can say that:

$$\mathbb{P}\left[\mathbf{p_i} = 1 | \mathbf{T} \geq i, \mathbf{H}_i = H_i\right] = \frac{1}{|R_i|} \cdot \mathbb{1}\left[v \in R_i\right] \ .$$

Additionally, Having $\mathbf{T} \geq i, \mathbf{H}_i = H_i$, if $\mathbf{p_i} = 1$, then $\mathbf{e}_i$ would be v. Otherwise, if $\mathbf{p_i} = 0$, then $\mathbf{e}_i^-$ would remain unchanged, i.e. $\mathbf{e}_i = \mathbf{e}_i^-$. Hence, $\mathbb{P}\left[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i\right]$ is equal to

$$\frac{1}{|R_i|} \cdot \mathbb{1}\left[v \in R_i\right] \cdot \mathbb{P}\left[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{p_i} = 1\right] + \left(1 - \frac{1}{|R_i|} \cdot \mathbb{1}\left[v \in R_i\right]\right) \cdot \mathbb{P}\left[\mathbf{e}_i^- = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{p_i} = 0\right].$$

We consider the following cases based on the value of e:

- Case (i): $e = v$
  In this case $\mathbb{P}\left[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{p_i} = 1\right] = 1$, and $\mathbb{P}\left[\mathbf{e}_i^- = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{p_i} = 0\right] = 0$. Thus, we have:

$$\mathbb{P}\left[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i\right] = \frac{1}{|R_i|} \cdot \mathbb{1}\left[v \in R_i\right] \cdot 1 + \left(1 - \frac{1}{|R_i|} \cdot \mathbb{1}\left[v \in R_i\right]\right) \cdot 0 = \frac{1}{|R_i|} \cdot \mathbb{1}\left[v \in R_i\right] = \frac{1}{|R_i|} \cdot \mathbb{1}\left[e \in R_i\right].$$

- Case (ii): $e \neq v$ In this case, $\mathbb{P}\left[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{p_i} = 1\right] = 0$. So we have:

$$\mathbb{P}\left[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i\right] = \frac{1}{|R_i|} \cdot \mathbb{1}\left[v \in R_i\right] \cdot 0 + \left(1 - \frac{1}{|R_i|} \cdot \mathbb{1}\left[v \in R_i\right]\right) \cdot \mathbb{P}\left[\mathbf{e}_i^- = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{p_i} = 0\right].$$

  According to the claim that we proved beforehand, $\mathbf{T} \geq i, \mathbf{H}_i = H_i$ and $\mathbf{T}^- \geq i, \mathbf{H}_i^- = H_i^-, \mathbf{p_1} = 0, \dots, \mathbf{p_{i-1}} = 0$ are interchangeable. So we have:

$$\mathbb{P}\left[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i\right] = \left(1 - \frac{1}{|R_i|} \cdot \mathbb{1}\left[v \in R_i\right]\right) \cdot \mathbb{P}\left[\mathbf{e}_i^- = e | \mathbf{T}^- \geq i, \mathbf{H}_i^- = H_i^-, \mathbf{p_1} = 0, \dots, \mathbf{p_i} = 0\right].$$

  Since for any $j \leq i$, $\mathbf{e}_i^-$ and $\mathbf{p_i}$ are independent random variables, we have:

$$\begin{aligned}\mathbb{P}\left[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i\right] &= \left(1 - \frac{1}{|R_i|} \cdot \mathbb{1}\left[v \in R_i\right]\right) \cdot \mathbb{P}\left[\mathbf{e}_i^- = e | \mathbf{T}^- \geq i, \mathbf{H}_i^- = H_i^-\right] \\ &= \left(1 - \frac{1}{|R_i|} \cdot \mathbb{1}\left[v \in R_i\right]\right) \cdot \left(\frac{1}{|R_i^-|} \cdot \mathbb{1}\left[e \in R_i^-\right]\right) \ ,\end{aligned}$$

  where the last equality holds because of the assumption stated in Lemma. From the definition of H_i^-, we have $R_i^- = R_i \backslash \{v\}$. Therefore,

$$\mathbb{P}\left[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i\right] = \frac{|R_i| - \mathbb{1}\left[v \in R_i\right]}{|R_i|} \cdot \left(\frac{1}{|R_i| - \mathbb{1}\left[v \in R_i\right]} \cdot \mathbb{1}\left[e \in R_i \backslash \{v\}\right]\right) \ .$$

  And since, $e \neq v$, we have:

$$\mathbb{P}\left[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i\right] = \frac{1}{|R_i|} \cdot \mathbb{1}\left[e \in R_i\right] \ .$$

□

As stated before, proof of these claims completes the Lemma's proof. □

3.3.2 Correctness of invariants after every deletion Now, we consider the case when the update at time t of the sequence $\mathcal{S}$, is a deletion of an element v, and prove the following theorem.

THEOREM 3.3. *If before the deletion of an element v, the level invariants and the uniform invariant hold, then they also hold after the execution of* DELETE(v).

Similar to Theorem 3.2, we break the proof of this theorem into Lemmas 3.6 and 3.7. Proofs of Lemmas 3.6 is given in Appendix A Sections A.3.

LEMMA 3.6. (LEVEL INVARIANTS) *If before the deletion of an element v the level invariants (i.e., starter, survivor, independent, weight, and terminator) hold, then they also hold after the execution of* DELETE(v).

LEMMA 3.7. (UNIFORM INVARIANT) *If before the deletion of an element v, the level and uniform invariants hold, then the uniform invariant also holds after the execution of* DELETE(v).

Proof. In other words, we want to prove that if for any i and any element e

$$\mathbb{P}\left[\mathbf{e}_i^- = e | \mathbf{T}^- \geq i, \mathbf{H}_i^- = H_i^-\right] = \frac{1}{|R_i^-|} \cdot \mathbb{1}\left[e \in R_i^-\right] \quad ,$$

then, after execution DELETE(v), for each i and each element e, we have

$$\mathbb{P}\left[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i\right] = \frac{1}{|R_i|} \cdot \mathbb{1}\left[e \in R_i\right] \quad .$$

Fix any arbitrary i and e. We define a random variable $\mathbf{X_i}$ attaining values from the set $\{0, 1, 2\}$, as follows:

1. If the execution of DELETE(v) has terminated after invoking MATROIDCONSTRUCTLEVEL(j), then we set $\mathbf{X_i}$ to 2.
2. If the execution of DELETE(v) has terminated in a level $L_{j\leq i}$ because $v \notin R_j^-$, then we set $\mathbf{X_i}$ to 1.
3. Otherwise, we set $\mathbf{X_i}$ to 0. That is, this case occurs if $v \in R_i^-$ and DELETE(v) terminates because in a level $L_{j>i}$, either $e_j = v$ or $v \notin R_j$.

In Claims 3.4, 3.7, and 3.8, we show that for each value $X_i \in \{0, 1, 2\}$, $\mathbb{P}\left[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{X_i} = X_i\right] = \frac{1}{|R_i|} \cdot \mathbb{1}\left[e \in R_i\right]$. This would imply the statement of our Lemma and completes the proof since

$$\mathbb{P}\left[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i\right] = \mathbb{E}_{X_i \sim \mathbf{X_i}}\left[\mathbb{P}\left[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{X_i} = X_i\right]\right]$$

by the law of total probability.

CLAIM 3.4. $\mathbb{P}\left[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{X_i} = 0\right] = \frac{1}{|R_i|} \cdot \mathbb{1}\left[e \in R_i\right]$.

Proof. First, we prove the following claim.

CLAIM 3.5. *If* $\mathbf{X_i} = 0$*, then for every* $j < i$*,* $e_j \neq v$ *and* $v \notin R_i$*.*

Proof. Since $\mathbf{X_i} = 0$, then MATROIDCONSTRUCTLEVEL(j) has not been invoked for any $j \leq i$. Thus, $\mathbf{e_j^-} = \mathbf{e_j} = e_j$ for any $j < i$. However, if $e_j = v$ for a level index $j < i$, then $\mathbf{e_j^-} = v$ would have held for that $j < i$, which means that MATROIDCONSTRUCTLEVEL(j) would have been executed for that j. This contradicts the assumption that $\mathbf{X_i} = 0$. Therefore, for all $j < i$, we must have $e_j \neq v$ proving the first part of this claim.

Next, we prove the second part. Since $\mathbf{X_i} = 0$, the algorithm DELETE(v) neither has called MATROIDCONSTRUCTLEVEL nor it terminates its execution until level L_i. Thus, $\mathbf{R_i} = \mathbf{R_i^-} - v$, which implies that $v \notin \mathbf{R_i}$. However, if we had $v \in R_i$, then the event $[\mathbf{H}_i = H_i, \mathbf{X_i} = \mathbf{0}]$ would have been impossible. □

Using Claim 3.5, we know that $e_j \neq v$ for $j < i$ and $v \notin R_i$. However, we also know that $v \in R_j^-$ for $j \leq i$. Thus, we can define $H_i^- = (e_1^-, \ldots, e_{i-1}^-, R_0^-, \ldots, R_i^-)$ based on $H_i = (e_1, \ldots, e_{i-1}, R_0, \ldots, R_i)$ as follows:

$$H_i^- = (e_1, \ldots, e_{i-1}, R_0 \cup \{v\}, \ldots, R_i \cup \{v\}) \quad .$$

CLAIM 3.6. *Two events* $[\mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{X_i} = 0]$ *and* $[\mathbf{T}^- \geq i, \mathbf{H}_i^- = H_i^-, \mathbf{e}_i^- \neq v]$ *are equivalent (i.e., they imply each other).*

Proof. We first prove that the event $[\mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{X_i} = 0]$ implies the event $[\mathbf{T}^- \geq i, \mathbf{H}_i^- = H_i^-, \mathbf{e}_i^- \neq v]$. Indeed, since $\mathbf{X_i} = 0 \neq 2$ we know that the algorithm MATROIDCONSTRUCTLEVEL(j) was not invoked for any $j \leq i$ and the element v was contained in $\mathbf{R_j^-}$ for all $j \leq i$. In this case, according to the algorithm DELETE(v), we conclude that for any $j \leq i$, we have $\mathbf{e_j^-} \neq v$ and $\mathbf{e_j^-} = \mathbf{e_j}$, and $\mathbf{R_j} = \mathbf{R_j^-} - v$. This means that $\mathbf{R_j^-} = \mathbf{R_j} \cup \{v\}$. Therefore, since $\mathbf{H}_i = H_i$, we must have $\mathbf{H}_i^- = H_i^-$, $\mathbf{e}_i^- \neq v$, and $\mathbf{e}_i^- = \mathbf{e}_i$.

Next, we prove the other way around. That is, the event $[\mathbf{T}^- \geq i, \mathbf{H}_i^- = H_i^-, \mathbf{e}_i^- \neq v]$ implies the event $[\mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{X_i} = 0]$. Indeed, since $\mathbf{H}_i^- = H_i^- = (e_1, \ldots, e_{i-1}, R_0 \cup \{v\}, \ldots, R_i \cup \{v\})$, then, for any $j \leq i$, $v \in \mathbf{R_j^-}$ and for any $j < i$, $\mathbf{e_j^-} = e_j$.

Recall from Claim 3.5 that for all $j < i$, $e_j \neq v$ and $v \notin R_i$. Thus, for any $j < i$, we know that $\mathbf{e}_{\mathbf{j}}^- \neq v$. However, we also know that $\mathbf{e}_i^- \neq v$. Thus, $\mathbf{e}_{\mathbf{j}}^- \neq v$ for any $j \leq i$. This essentially means that the algorithm Delete(v) neither invokes MatroidConstructLevel nor terminates its execution till the level L_i. This implies that $\mathbf{X_i} = 0$. On the other hand, the algorithm Delete(v) only removes v from R_i^- and does not make any change in $\mathbf{e}_{\mathbf{1}}^-, \ldots, \mathbf{e}_{\mathbf{i}}^-$. Thus, $\mathbf{R_i} = R_i^- - \{v\} = R_i \cup \{v\} - v = R_i$ and $\mathbf{e}_i = \mathbf{e}_i^-$. Therefore, we have $\mathbf{H}_i = H_i$. ∎

Therefore, we have the following corollary.

Corollary 3.1. $\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{X_i} = 0] = \mathbb{P}\left[\mathbf{e}_i^- = e | \mathbf{T}^- \geq i, \mathbf{H}_i^- = H_i^-, \mathbf{e}_i^- \neq v\right]$.

Thus, in order to prove $\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{X_i} = 0] = \frac{1}{|R_i|} \cdot \mathbb{1}[e \in R_i]$, we can prove

$$\mathbb{P}[\mathbf{e}_i^- = e | \mathbf{T}^- \geq i, \mathbf{H}_i^- = H_i^-, \mathbf{e}_i^- \neq v] = \frac{1}{|R_i|} \cdot \mathbb{1}[e \in R_i] \quad .$$

Recall that the assumption of this lemma is $\mathbb{P}\left[\mathbf{e}_i^- = e | \mathbf{T}^- \geq i, \mathbf{H}_i^- = H_i^-\right] = \frac{1}{|R_i^-|} \cdot \mathbb{1}\left[e \in R_i^-\right]$. That is, conditioned on the event $[\mathbf{T}^- \geq i, \mathbf{H}_i^- = H_i^-]$, the random variable $\mathbf{e}_i^- \sim U(R_i^-)$ is a uniform random variable over the set R_i^-. (i.e., the value e_i of the random variable $\mathbf{e}_i^-$ takes ones of the elements of the set R_i^- uniformly at random.) However, since $X_i = 0$ and using Claim 3.6, we have $\mathbf{e}_i^- \neq v$. Thus, conditioned on the event $[\mathbf{T}^- \geq i, \mathbf{H}_i^- = H_i^-, \mathbf{e}_i^- \neq v]$, we have that the random variable $\mathbf{e}_i^- \sim U(R_i^- \setminus \{v\}) = U(R_i)$ should be a uniform random variable over the set $R_i^- \setminus \{v\} = R_i$. Indeed, we have

$$\begin{aligned}\mathbb{P}[\mathbf{e}_i^- = e | \mathbf{T}^- \geq i, \mathbf{H}_i^- = H_i^-, \mathbf{e}_i^- \neq v] &= \frac{\mathbb{P}\left[\mathbf{e}_i^- = e, \mathbf{e}_i^- \neq v | \mathbf{T}^- \geq i, \mathbf{H}_i^- = H_i^-\right]}{\mathbb{P}\left[\mathbf{e}_i^- \neq v | \mathbf{T}^- \geq i, \mathbf{H}_i^- = H_i^-\right]} = \frac{\frac{1}{|R_i^-|} \cdot \mathbb{1}\left[e \in R_i^- \setminus \{v\}\right]}{1 - \frac{1}{|R_i^-|}} \\ &= \frac{1}{|R_i^-| - 1} \cdot \mathbb{1}[e \in R_i^- \setminus \{v\}] = \frac{1}{|R_i|} \cdot \mathbb{1}[e \in R_i] \quad ,\end{aligned}$$

where the second equality holds because of our assumption that the uniform invariant holds before the deletion, and the fourth invariant holds because $R_i^- = R_i \cup \{v\}$ and $v \notin R_i$ proving the case $X = 0$.

∎

Claim 3.7. $\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{X_i} = 1] = \frac{1}{|R_i|} \cdot \mathbb{1}[e \in R_i]$.

Proof. We will be conditioning on possible values of $\mathbf{H}_i^-$.

$$\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i, X_i = 1] = \mathbb{E}_{H_i^-}[\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{X_i} = 1, \mathbf{H}_i^- = H_i^-]] \quad ,$$

where the expectation is taken over all H_i for which $\mathbb{P}\left[\mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{X_i} = 1, \mathbf{H}_i^- = H_i^-\right] > 0$. For all such H_i^-, we claim that this can be further rewritten as $\mathbb{P}\left[\mathbf{T} \geq i, \mathbf{H}_i^- = H_i^-\right]$. This is because Delete(v) is executed deterministically if it does not invoke the algorithm MatroidConstructLevel. Furthermore, the value of $\mathbf{X_i}$ is deterministically determined by $\mathbf{H}_i^-$. Therefore, for any value of H_i^-, either $\mathbf{H}_i^- = H_i^-$ implies $\mathbf{X_i} \neq 1$, in which case $\mathbb{P}\left[\mathbf{T} \geq i, \mathbf{H}_i^- = H_i^-, \mathbf{X_i} = 1\right] = 0$, which is in contradiction with our assumption, or $\mathbf{H}_i^- = H_i^-$ imply $\mathbf{X_i} = 1$. Therefore, for all such H_i^- implies $\mathbf{X_i} = 1$, which also means that MatroidConstructLevel never gets invoked, in which case $\mathbf{H}_i$ is uniquely determined. Hence $\mathbf{H}_i^- = H_i^-$ should also imply that $\mathbf{H}_i = H_i$, as otherwise $\mathbb{P}\left[\mathbf{T} \geq i, \mathbf{H}_i^- = H_i^-, \mathbf{H}_i = H_i\right] = 0$. We therefore obtain:

$$\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{X_i} = 1, \mathbf{H}_i^- = H_i^-] = \mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i^- = H_i^-]$$

as claimed.

Also, we know that $\mathbf{H}_i = H_i, \mathbf{X_i} = 1$, implies that:

$$\mathbf{T}^- = \mathbf{T}, \;\; \mathbf{R_i^-} = \mathbf{R_i}, \;\; \mathbf{e}_i^- = \mathbf{e}_i,$$

since it means that the execution of Delete(v) has terminated before level i, thus no change has been made for that level. Therefore, for a H_i^- used in our expectation, we know that $\mathbf{T} \geq i, \mathbf{H}_i^- = H_i^-$ also implies

$$\mathbf{T}^- \geq i, \;\; R_i^- = R_i, \;\; \mathbf{e}_i^- = \mathbf{e}_i,$$

we have:

$$\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i^- = H_i^-] = \mathbb{P}[\mathbf{e}_i^- = e | \mathbf{T}^- \geq i, \mathbf{H}_i^- = H_i^-] = \frac{1}{|R_i^-|} \cdot \mathbb{1}[e \in R_i^-] = \frac{1}{|R_i|} \cdot \mathbb{1}[e \in R_i] \quad ,$$

where the third equality holds because of our assumption that the uniform invariant holds before the deletion of element v. Therefore, $\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{X_i} = 1] = \frac{1}{|R_i|} \cdot \mathbb{1}[e \in R_i]$.

□

Claim 3.8. $\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{X_i} = 2] = \frac{1}{|R_i|} \cdot \mathbb{1}[e \in R_i]$.

Proof. By Lemma 3.6, we know that the level invariants hold at the end of the execution of Delete, which is also the end of the execution of MatroidConstructLevel(j). Using Lemma 3.3, we know that since the level invariants are going to hold after the execution of MatroidConstructLevel(j), for i which is greater than j, we have:

$$\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{X_i} = 2] = \frac{1}{|R_i|} \cdot \mathbb{1}[e \in R_i] \quad ,$$

which proves this claim. □

□

3.4 Application of Uniform Invariant: Query complexity

As for the query complexity of this algorithm, observe that checking if an element e is promoting for a level L_z needs $O(\log(k))$ oracle queries because of Lemma 3.2 and the fact that the size of the independent set I_z is at most k. The binary search that we perform needs $O(\log T)$ number of such suitability checks for the element e. Thus, if we initiate the leveling algorithm with a set R_i, our algorithm needs $O(|R_i| \cdot \log(k) \cdot \log(T))$ oracle queries to build the levels $L_i, \cdots, L_T$.

Lemma 3.8. *The number of levels T is at most $k \log(\frac{k}{\epsilon})$.*

Proof. Consider a directed graph G with elements $I'_T = \{e_1, \cdots, e_T\}$ as vertices of this graph. For each element $e_i \in I'_T$, we know that e_i is a promoting element for L_{i-1}, i.e. Promote($I_{i-1}, I'_{i-1}, e_i, w[I_{i-1}]$) $\neq$ Fail. Therefore, we define $parent(e_i) =$ Promote($I_{i-1}, I'_{i-1}, e_i, w[I_{i-1}]$). This value is $\emptyset$ if $I_i = I_{i-1} + e_i$. Otherwise, if $I_i = I_{i-1} - e' + e_i$, this value would be e'. For each $e_i \in I'_T$, if $parent(e_i) \neq \emptyset$, we add an edge $e_i \to parent(e_i)$ to the graph.

Since an element can only be replaced once, we have $|\{e' | e' \in I'_T, parent(e') = e\}| = 1$, i.e. the in-degree of each $e \in I_T$ is at most 1. Furthermore, the out-degree of each vertex is 1, because for each element $e \in I'_T$, $|parent(e)| \leq 1$. Therefore, it follows that the graph is a union of disjoint paths and each $e_i \in I'_T$ is in exactly one path.

An element e is a starting element in a path (its in-degree is 0), if and only if it has not been replaced by another element. That means, e remains in I_T at the end of the algorithm. Given that $|I_T| \leq k$, there are at most k paths in G. Furthermore, for two successive elements (u, v) in the path where $parent(u) = v$, $w(u) \geq 2w(v)$. As the weights of all elements in I'_T satisfy $w(e) \in [\frac{\epsilon}{10k} MAX, MAX]$, the length of each path is bounded by $\log(k/\epsilon) + 4$. Consequently, it follows that the total number of vertices in the graph is at most $O(k \log(\frac{k}{\epsilon}))$. □

Next, we analyze the query complexity of MatroidConstructLevel.

Lemma 3.9. *The total cost of calling* MatroidConstructLevel(i) *is at most $O\left(|R_i| \log(k) \log\left(\frac{k}{\epsilon}\right)\right)$.*

Proof. Checking if an element e is promoting needs $O(\log(k))$ query calls, because of Lemma 3.2 and the fact that $|I| \leq k$ for any $I \in \mathcal{I}$. The algorithm MatroidConstructLevel(i) iterates over all elements in R_i. For each element e, it first calls the Promote function, and select e if it is a promoting element, i.e. Promote($I_{\ell-1}, I'_{\ell-1}, e, w[I_{\ell-1}]$) $\neq$ Fail. In this case, we only need $O(\log(k))$ query calls. However, if e is not a promoting element, it reaches Line 13 and runs the binary search on the interval $[i, \ell - 1]$. Based on Lemma 3.8, the length of this interval is $O\left(k \log\left(\frac{k}{\epsilon}\right)\right)$. Therefore, the number of steps in binary search is at most $O\left(\log\left(k \log\left(\frac{k}{\epsilon}\right)\right)\right) = O\left(\log\left(\frac{k}{\epsilon}\right)\right)$. In each step of the binary search, the algorithm calls Promote one time. Thus, for each element we need $O\left(\log(k) \log\left(\frac{k}{\epsilon}\right)\right)$, and for all elements, we need $O\left(|R_i| \log(k) \log\left(\frac{k}{\epsilon}\right)\right)$ query calls. □

Lemma 3.10. *For a specified value of MAX, each update operation in Algorithm 2 has query complexity at most $O\left(k \log(k) \log^2\left(\frac{k}{\epsilon}\right)\right)$.*

Proof. We divide the queries made by the algorithm into two categories: the queries made directly by the update operations Insert and Delete, and the queries made indirectly, if the update triggers a call to MatroidConstructLevel. For the first category, the number of queries for each update is always $O(T)$ for insertion which can be bounded by $O\left(k \log\left(\frac{k}{\epsilon}\right)\right)$, and there are no queries made for deletion. We therefore focus on the second category.

Based on uniform invariant, when we insert/delete an element, for each natural number $i \leq T$, we call MatroidConstructLevel(i) with probability $\dfrac{1}{|R_i|} \cdot \mathbb{1}[e \in R_i]$ which is at most $\dfrac{1}{|R_i|}$. Using Lemma 3.9, the query complexity for calling MatroidConstructLevel(i) is $O\left(|R_i| \log(k) \log\left(\frac{k}{\epsilon}\right)\right)$. Therefore, the expected number of queries caused by level i is bounded by $\dfrac{1}{|R_i|} \cdot O\left(|R_i| \log(k) \log\left(\frac{k}{\epsilon}\right)\right) = O\left(\log(k) \log\left(\frac{k}{\epsilon}\right)\right)$. As the Lemma 3.8 bounded the number of levels by $T = O\left(k \log\left(\frac{k}{\epsilon}\right)\right)$, we calculate the expected number of query calls for each update by summing the expected number of query calls at each level:

$$\sum_{i=1}^{T} O\left(\log(k) \log\left(\frac{k}{\epsilon}\right)\right) \le O\left(k \log(k) \log^2\left(\frac{k}{\epsilon}\right)\right) \;.$$

□

In order to obtain an algorithm that works regardless of the value of MAX, we guess MAX up to a factor of 2 using parallel runs. Each element is inserted only to $\log(k/\epsilon)$ copies of the algorithm. Therefore, we obtain the total query complexity claimed in Theorem 3.4.

THEOREM 3.4. *The expected query complexity of each insert/delete for all runs is* $O\left(k \log(k) \log^3\left(\frac{k}{\epsilon}\right)\right)$.

3.5 Application of Level Invariants: Approximation guarantee Recall that we run parallel instances of DYNAMICMATROID for different guesses of the maximum value MAX such that after each update, there is a run with $\max_{e\in V_t} f(e) \in (MAX/2, MAX]$, where V_t is the set of elements that have been inserted but not deleted yet. In this section, we only talk about the run with $\max_{e\in V_t} f(e) \in (MAX/2, MAX]$. We prove that if the level invariants hold, then after each update the submodular value of the set I_T in this run is a $(4+\epsilon)$-approximation of the optimal value OPT. Formally, we state this claim as follows:

THEOREM 3.5. *Suppose that the level invariants hold in every run of* DYNAMICMATROID. *Let I_T be the independent set of the final level L_T in the run with* $\max_{e\in V} f(e) \in (MAX/2, MAX]$. *Then, the set I_T satisfies* $(4+\epsilon)\cdot f(I_T) \ge OPT$, *where* $OPT = \max_{I^*\in\mathcal{I}} f(I^*)$.

To this end, we first define a few notations.

DEFINITION 3.2. *For an element $e \in \mathcal{V}$, we let $z(e)$ denote the largest i such that $e \in R_i$. In Algorithms 1 and 2, $w(e)$ is defined for all elements $e \in I'_T$, but we need to define it for other elements as well. Therefore, if $e_{z(e)} = e$, we set $w(e) = f(I'_{z(e)-1} + e) - f(I'_{z(e)-1})$, to match the value defined in the Algorithm. Otherwise, we set $w(e) = f(I'_{z(e)} + e) - f(I'_{z(e)})$. For a set $E \subseteq \mathcal{V}$, we define $w(E) = \sum_{e\in E} w(e)$.*

We split the proof of Theorem 3.5 into four steps. We first (in Lemma 3.11) prove that $w(I'_T) \le 2w(I_T)$. Later, in Lemma 3.12 we show that the sum of the weight of the elements in I_T is upper-bounded by the submodular function of I_T. That is, $w(I_T) \le f(I_T)$. Recall that $OPT = \max_{I\in\mathcal{I}} f(I)$ and we used the notation $I^* = \arg\max_{I\in\mathcal{I}} f(I)$ for an independent set in $\mathcal{I}$ whose submodular value is maximum. In the third step of the proof of Theorem 3.5, we show that $f(I^*) \le 2w(I_T) + w(I^*)$. We prove this in Lemma 3.13. Our proofs for these lemmas are inspired by the analysis in Chakrabarti and Kale [43] who study the streaming version of the problem. Finally, we show that $w(I^*) \le 2w(I_T) + \frac{\epsilon}{5}\cdot f(I^*)$. This is proven in Lemma 3.14 using an argument inspired by the analysis of Ashwinkumar [11].

Having all these tools in hand, we can then finish the proof of Theorem 3.5. Indeed, we have

$$f(I^*) \overset{(a)}{\le} 2w(I_T) + w(I^*) \overset{(b)}{\le} 4w(I_T) + \frac{\epsilon}{5}\cdot f(I^*) \overset{(c)}{\le} 4f(I_T) + \frac{\epsilon}{5}\cdot f(I^*) \;, \tag{3.1}$$

where (a), (b), and (c) follow from Lemmas 3.13, 3.14 and 3.12 respectively.

This essentially means that $f(I^*) \le \frac{4}{1-\frac{\epsilon}{5}}\cdot f(I_T)$. Now observe that $\frac{4}{1-\frac{\epsilon}{5}} \le 4+\epsilon$. Indeed, if we want to have this claim correct, we must have $20 - 4\epsilon + 5\epsilon - \epsilon^2 \ge 20$ which means we must have $\epsilon(\epsilon-1) \le 0$. However, this is correct since $0 < \epsilon \le 1$, which finishes the proof of Theorem 3.5.

Next, we prove the four steps that we explained above.

DEFINITION 3.3. (SPAN) *Let $E \subseteq V$ be a set of elements. We define* $\text{SPAN}(E) = \{e \in V : rank(E+e) = rank(E)\}$.

LEMMA 3.11. $w(I'_T) \le 2w(I_T)$.

Proof. We prove by induction on i that $w(I'_i) \le 2w(I_i)$ for all i. Setting $i = T$ will finish the proof.
The claim holds for $i = 0$ as $w(I_i) = w(I'_i) = w(\emptyset) = 0$. Assume that the claim holds for $i-1$, we prove it holds for i as well. Given independent invariant, $I'_i = I'_{i-1} + e_i$ and either $I_i = I_{i-1} + e_i$ or $I_i = I_{i-1} + e_i - \hat{e}$ for some $\hat{e}$ satisfying $w(\hat{e}) \le \frac{w(e_i)}{2}$. In either case,

$$\begin{aligned} w(I_i) &\ge w(I_{i-1}) + w(e_i) - \frac{w(e_i)}{2} \ge w(I_{i-1}) + \frac{w(e_i)}{2} \\ &\overset{(a)}{\ge} \frac{1}{2}w(I'_{i-1}) + \frac{w(e_i)}{2} = \frac{1}{2}w(I'_i) \;, \end{aligned}$$

where for (a), we have used the induction assumption for $i-1$. □

LEMMA 3.12. *The sum of the weight of the elements in I_T is upper-bounded by the submodular function of I_T. That is, $w(I_T) \le f(I_T)$.*

Proof. For each $i \in [T]$, define $\widetilde{I}_i$ as $I_i \cap I_T$. We prove by induction on i that $w(\widetilde{I}_i) \le f(\widetilde{I}_i)$. Setting $i = T$ proves the claim.

The case of $i = 0$ holds trivially as $w(\widetilde{I}_0) = f(\widetilde{I}_0) = 0$. Assume that $w(\widetilde{I}_{i-1}) \le f(\widetilde{I}_{i-1})$, we will prove that $w(\widetilde{I}_i) \le f(\widetilde{I}_i)$. If $e_i \notin I_T$, then the claim holds trivially as $\widetilde{I}_i = \widetilde{I}_{i-1}$. Note that in this case, if an element has appeared in I_{i-1}, but it is removed from I_i, then it is not included in I_T and hence $\widetilde{I}_{i-1}$. We therefore assume that $e_i \in I_T$. In this case, we note that

$$I'_{i-1} = \bigcup_{j \le i-1} I_j \supseteq I_{i-1} \supseteq \widetilde{I}_{i-1} \quad . \tag{3.2}$$

Therefore,

$$w(\widetilde{I}_i) - w(\widetilde{I}_{i-1}) = w(e_i) \overset{(a)}{=} f(I'_{i-1} + e) - f(I'_{i-1}) \overset{(b)}{\le} f(\widetilde{I}_{i-1} + e) - f(\widetilde{I}_{i-1}) = f(\widetilde{I}_i) - f(\widetilde{I}_{i-1}) \quad ,$$

where for (a) we have used weight invariant, and for (b) we have used the definition of submodularity together with (3.2). Summing the above inequality with the induction hypothesis $w(\widetilde{I}_{i-1}) \le f(\widetilde{I}_{i-1})$ proves the claim. □

LEMMA 3.13. *Recall that $OPT = \max_{I \in \mathcal{I}} f(I)$ and we used the notation $I^* = \arg\max_{I \in \mathcal{I}} f(I)$ for an independent set in $\mathcal{I}$ whose submodular value is maximum. Then, $f(I^*) \le 2w(I_T) + w(I^*)$.*

Proof. We first note that

$$f(I'_T) = \sum_{i=1}^{T} f(I'_i) - f(I'_{i-1}) = \sum_{i=1}^{T} f(I'_{i-1} + e_i) - f(I'_{i-1}) \overset{(a)}{=} \sum_{i=1}^{T} w(e_i) = w(I'_T) \overset{(b)}{\le} 2w(I_T) \quad , \tag{3.3}$$

where (a) follows from weight invariant, and (b) follows from Lemma 3.11.

We now bound $f(I^*)$. Enumerate $I^* \backslash I'_T$ as $\{e^*_1, \ldots, e^*_{|I^* \backslash I'_T|}\}$ in an arbitrary order. Define $D_0 = I'_T$ and $D_i = I'_T \cup \{e^*_1, \ldots e^*_i\}$. It is clear that $D_{i-1} \supseteq I'_T \supseteq I'_{z(e^*_i)}$. Therefore,

$$f(D_i) - f(D_{i-1}) = f(D_{i-1} + e^*_i) - f(D_{i-1}) \overset{(a)}{\le} f(I'_{z(e^*_i)} + e^*_i) - f(I'_{z(e^*_i)}) \overset{(b)}{=} w(e^*_i) \quad ,$$

where for (a) we have used the definition of submodularity, and (b) holds because $e^*_i \notin I'_T$. Summing over all i, we obtain

$$\begin{aligned} \sum_{i=1}^{|I^* \backslash I'_T|} f(D_i) - f(D_{i-1}) &\le \sum_{i=1}^{|I^* \backslash I'_T|} w(e^*_i) \\ f(D_{|I^* \backslash I'_T|}) - f(D_0) &\le w(I^* \backslash I'_T) \\ &\le w(I^*) \quad . \end{aligned}$$

Given that $D_0 = I'_T$ and $D_{|I^* \backslash I'_T|} = I^* \cup I'_T$, we have

$$f(I^*) \le f(I^* \cup I'_T) \le f(I'_T) + w(I^*) \le 2w(I_T) + w(I^*) \quad ,$$

where the last inequality follows from (3.3).

□

LEMMA 3.14. $w(I^*) \le 2w(I_T) + \frac{\epsilon}{5} \cdot f(I^*)$.

We first give a sketch of the proof of Lemma 3.14.

We split the I^* into two parts. The first part consists of elements with weights $w(e) \le \frac{\epsilon}{10k} MAX$. As we will show, the total weight of these elements can be bounded by $\frac{\epsilon}{5} \cdot f(I^*)$. As for the second group, we will show that each element can be mapped one-to-one to an element in I_T with at least half its weight.

Formally, let I^*_W consist of all the elements in I^* such that $w(e) \le \frac{\epsilon}{10k} MAX$. We first bound $w(I^*_W)$:

$$\begin{aligned} w(I^*_W) = \sum_{e \in I^*_W} w(e) &\le |I^*_W| \cdot \frac{\epsilon}{10k} \cdot MAX \\ &\le |I^*| \cdot \frac{\epsilon}{10k} \cdot MAX \\ &\le \frac{\epsilon}{10} \cdot MAX < \frac{\epsilon}{5} \cdot f(I^*) \quad . \end{aligned} \tag{3.4}$$

The last conclusion comes from the fact that $f(I^*) \geq \max_{e \in V} f(e) \in (\frac{MAX}{2}, MAX]$.

In order to bound $w(I^* \backslash I^*_W)$, we will use the following lemmas:

LEMMA 3.15. *Let sets $E_1, E_2 \subseteq V$ and elements $e_1, e_2 \in V$. If $e_1 \in \textsc{Span}(E_1)$ and $e_2 \in \textsc{Span}(E_2 - e_2)$, then $e_1 \in \textsc{Span}((E_1 \cup E_2) - e_2)$.*

Proof. Note that if we have two sets $S_1, S_2 \subseteq E$ such that $S_1 \subseteq S_2$, then $\textsc{Span}(S_1) \subseteq \textsc{Span}(S_2)$ and $rank(S_1) \leq rank(S_2)$. Now, using Definition 3.3 with the fact $E_1 \subseteq (E_1 \cup E_2)$ gives us $e_1 \in \textsc{Span}(E_1) \subseteq \textsc{Span}(E_1 \cup E_2)$. Therefore, we have $rank((E_1 \cup E_2) + e_1) = rank(E_1 \cup E_2)$.

In a similar way, since $E_2 - e_2 \subseteq ((E_1 \cup E_2) - e_2)$, we can conclude that $e_2 \in \textsc{Span}(E_2 - e_2) \subseteq \textsc{Span}((E_1 \cup E_2) - e_2)$ what implies that $rank(E_1 \cup E_2) = rank((E_1 \cup E_2) - e_2)$.

By using these two results, we conclude that $rank((E_1 \cup E_2) - e_2) = rank((E_1 \cup E_2) + e_1)$. Furthermore,

$$rank((E_1 \cup E_2) - e_2) \leq rank((E_1 \cup E_2) - e_2 + e_1) \leq rank((E_1 \cup E_2) + e_1)$$

where the first and third parts are equal. Therefore, all of them are equal and $rank((E_1 \cup E_2) - e_2 + e_1) = rank((E_1 \cup E_2) - e_2)$, which implies that $e_1 \in \textsc{Span}((E_1 \cup E_2) - e_2)$. □

LEMMA 3.16. *There is a function $N : I^* \backslash I^*_W \to 2^{I_T}$ such that for all $e \in I^* \backslash I^*_W$, $e \in \textsc{Span}(N(e))$ and for all $e' \in N(e)$, $w(e) \leq 2w(e')$.*

Proof. Define $\widetilde{I_i} := \{e \in I^* \backslash I^*_W : z(e) \leq i\}$. We prove by induction on $i \in [T]$ that there is a function $N_i : \widetilde{I_i} \to 2^{I_i}$ such that $e \in \textsc{Span}(N_i(e))$ and $w(e) \leq 2w(e')$ for all $e' \in N_i(e)$.

The induction base holds trivially as $\widetilde{I_0} = \emptyset$. Assume the claim holds for $i-1$. We show it holds for i. Let e be an element of $\widetilde{I_i}$. We define $N_i(e)$ based on three cases as follows.

- **Assume that $\mathbf{e} \in \widetilde{\mathbf{I}}_{\mathbf{i-1}}$.** If $I_i = I_{i-1} + e_i$ or $I_i = I_{i-1} + e_i - \hat{e}$ for some $\hat{e} \notin N_{i-1}(e)$, we set $N_i(e) = N_{i-1}(e)$. N_i has the desirable properties for e by the induction hypothesis. Otherwise, assuming that $I_i = I_{i-1} + e_i - \hat{e}$, for some $\hat{e} \in N_{i-1}(e)$. By Lemma 1.2 there is a unique circuit in $I_{i-1} + e_i$, named C. Define $N_i(e)$ as $(N_{i-1}(e) \cup C) - \hat{e}$. Since $\hat{e} \in N_{i-1}(e)$, by induction hypothesis, $w(e_i) \leq 2w(\hat{e})$. Also, given independent invariant, $\hat{e} = \textsc{Promote}(I_{i-1}, I'_{i-1}, e_i, w[I_{i-1}])$, i.e. $\hat{e} \leftarrow \arg\min_{e' \in C} w(e')$. Thus, $w(e_i) \leq 2w(\hat{e}) \leq 2w(e')$ for all $e' \in C - \hat{e}$. Moreover, $w(e_i) \leq 2w(e')$ for all $e' \in N_{i-1}(e)$ by induction hypothesis. Hence, $w(e_i) \leq 2w(e')$ for all $e' \in ((N_{i-1}(e) \cup C) - \hat{e})$. Furthermore, since $e \in \textsc{Span}(N_{i-1}(e))$ and $\hat{e} \in \textsc{Span}(C - \hat{e})$, we can use Lemma 3.15 to conclude that $e \in \textsc{Span}(N_i(e))$.
- **Assume that $\mathbf{e} = \mathbf{e_i}$.** In this case, we set $N_i(e) = e$.
- **If neither of the two cases above hold,** then $z(e) = i$ but $e \neq e_i$. According to the survivor invariant, e is not a promoting element for L_i. It follows that $I_i + e$ is not independent. By Lemma 1.2 there is a unique circuit in $I_i + e$. Let C denote this circuit, and let $N_i(e) = C - e$. It is clear that $e \in \textsc{Span}(N_i(e))$, and $w(e) \leq 2w(e')$ for all $e' \in C - e$ since otherwise, e would be promoting element for L_i.

Finally, we set $N = N_T$ to get the desired function. □

LEMMA 3.17. *Assume that $E, E' \subseteq V$. If E be an independent set such that $E \subseteq \textsc{Span}(E')$, then $|E| \leq |E'|$.*

Proof. Given that E is independent, we know that $|E| = rank(E)$. In addition, given that $E \subseteq \textsc{Span}(E')$, we have $rank(E) \leq rank(\textsc{Span}(E'))$. Therefore, $|E| = rank(E) \leq rank(\textsc{Span}(E')) = rank(E') \leq |E'|$. □

Proof. [Proof of Lemma 3.14] Let $N : I^* \backslash I^*_W \to 2^{I_T}$ be the function described in Lemma 3.16. Recall that $e \in \textsc{Span}(N(e))$ for all $e \in I^* \backslash I^*_W$. This further implies that for all $E \subseteq I^* \backslash I^*_W$, we have $E \subseteq \textsc{Span}(N(E))$. Therefore, as E is independent, we can use Lemma 3.17 to conclude that $|E| \leq |N(E)|$.

By Hall's marriage theorem, we conclude that there is an injection $H : I^* \backslash I^*_W \to I_T$ such that $H(e) \in N(e)$ for all $e \in I^* \backslash I^*_W$. Therefore,

$$w(I^* \backslash I^*_W) = \sum_{e \in I^* \backslash I^*_W} w(e) \overset{(a)}{\leq} \sum_{e \in I^* \backslash I^*_W} 2w(H(e)) \overset{(b)}{\leq} \sum_{e' \in I_T} 2w(e') = 2w(I_T) \ .$$

where for (a) we have used the fact that $w(e) \leq 2w(e')$ for all $e' \in N(e)$, and for (b) we have used the fact that H is an injection. Summing the above inequality with (3.4) finishes the proof.

$$w(I^*) = w(I^* \backslash I^*_W) + w(I^*_W) \leq 2w(I_T) + \frac{\epsilon}{5} \cdot f(I^*) \ .$$

□

4 Parameterized dynamic algorithm for submodular maximization under cardinality constraint

In this section, we present our dynamic algorithm for the maximum submodular problem under the cardinality constraint k. The pseudo-code of our algorithm is provided in Algorithm 4. The overview of our dynamic algorithm is given in Section "Our contribution" 1.2. The analysis of this algorithm is similar to the dynamic algorithm that we designed for the matroid constraint. Thus, we explain it in Appendix B.

Algorithm 4 CardinalityConstraintLeveling(k, OPT)

function Init(V)
  $\tau \leftarrow \frac{OPT}{2k}$
  $I_0 \leftarrow \emptyset$ and $R_0 \leftarrow V$
  $R_1 \leftarrow \{e \in R_0 : \text{Promote}(I_0, e) = True\}$
  Invoke ConstructLevel($i = 1$)

---

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

---

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

Relaxing OPT **assumption.** Our dynamic algorithm assumes the optimal value $OPT = \max_{I^* \subseteq V : |I^*| \leq k} f(I^*)$ is given as a parameter. However, in reality, the optimal value is not known in advance and may change after every insertion or deletion. To remove this assumption in Algorithm 6, we run parallel instances of our dynamic algorithm for different guesses of the optimal value OPT_t at any time t of the sequence $\mathcal{S}_t$, such that $\max_{I^* \subseteq V_t : |I^*| \leq k} f(I^*) \in (OPT_t/(1+\epsilon), OPT_t]$ in one of the runs. Recall that V_t is the set of elements that have been inserted but not deleted from the beginning of the sequence till time t. These guesses that we take are $(1+\epsilon)^i$ where $i \in \mathbb{Z}$. If ρ is the ratio between the maximum and minimum non-zero possible value of a subset of V with at most k elements, then the number of parallel instances of our algorithm will be $O(\log_{1+\varepsilon} \rho)$. This incurs an extra $O(\log_{1+\varepsilon} \rho)$-factor in the query complexity of our dynamic algorithm.

In fact, we can replace this extra factor with an extra factor of $O(\log(k)/\epsilon)$ which is independent of ρ. To this end, we use the well-known technique that has been also used in [109]. In particular, for every element e, we add it to those instances i for which we have $\frac{(1+\epsilon)^i}{2k} \leq f(e) \leq (1+\epsilon)^i$. The reason is if the optimal value of V_t is within the range $((1+\epsilon)^{i-1}, (1+\epsilon)^i]$ and $f(e) > (1+\epsilon)^i$, then $f(e)$ is greater than the optimal value and can safely be ignored for the instance i that corresponds to the guess $(1+\epsilon)^i$. On the other hand, we can safely ignore all elements e whose $f(e) < \frac{(1+\epsilon)^i}{2k} = \tau$, since these elements will never be a promoting element in the run with $OPT = (1+\epsilon)^i$. This essentially means that every element e is added to at most $O(\log_{1+\epsilon}(2k)) = O(\log(k)/\epsilon)$ parallel instances. Thus, after every insertion or deletion, we need to update only $O(\log(k)/\epsilon)$ instances of our dynamic algorithm.

5 Acknowledgements

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

References

[1] A. Abboud, R. Addanki, F. Grandoni, D. Panigrahi, and B. Saha. Dynamic set cover: improved algorithms and lower bounds. In

Algorithm 5 CardinalityConstraintUpdates(k, OPT)

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

---

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

Algorithm 6 Unknown OPT

Let $\mathcal{A}_i$ be the instance of our dynamic algorithm, for which $OPT = (1+\epsilon)^i$.

---

function UpdateWithoutKnowingOPT(e)
  for each $i \in [\lceil \log_{1+\epsilon} f(e) \rceil, \lfloor \log_{1+\epsilon}(2k \cdot f(e)) \rfloor]$ **do**   ▷ $\frac{(1+\epsilon)^i}{2k} \leq f(e) \leq (1+\epsilon)^i$
    Invoke Update(e) for instance $\mathcal{A}_i$.

Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019, pages 114–125. ACM, 2019.

[2] I. Abraham, S. Chechik, and S. Krinninger. Fully dynamic all-pairs shortest paths with worst-case update-time revisited. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 440–452. SIAM, 2017.

[3] I. Abraham, D. Durfee, I. Koutis, S. Krinninger, and R. Peng. On fully dynamic graph sparsifiers. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 335–344. IEEE Computer Society, 2016.

[4] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong. Diversifying search results. In *Proceedings of the Second International Conference on Web Search and Web Data Mining, WSDM 2009, Barcelona, Spain, February 9-11, 2009*, pages 5–14. ACM, 2009.

[5] K. J. Ahn, S. Guha, and A. McGregor. Analyzing graph structure via linear measurements. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 459–467. SIAM, 2012.

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

[7] S. Assadi and S. Khanna. Tight bounds on the round complexity of the distributed maximum coverage problem. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 2412–2431. SIAM, 2018.

[8] S. Assadi, K. Onak, B. Schieber, and S. Solomon. Fully dynamic maximal independent set with sublinear update time. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages

815–826. ACM, 2018.
[9] M. Babaioff, N. Immorlica, D. Kempe, and R. Kleinberg. Matroid secretary problems. *J. ACM*, 65(6):35:1–35:26, 2018.
[10] A. Badanidiyuru, B. Mirzasoleiman, A. Karbasi, and A. Krause. Streaming submodular maximization: massive data summarization on the fly. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, pages 671–680. ACM, 2014.
[11] A. Badanidiyuru Varadaraja. Buyback problem-approximate matroid intersection with cancellation costs. In *International Colloquium on Automata, Languages, and Programming*, pages 379–390. Springer, 2011.
[12] M. Balcan and N. J. A. Harvey. Learning submodular functions. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 793–802. ACM, 2011.
[13] M. Balcan and N. J. A. Harvey. Learning submodular functions. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2012, Bristol, UK, September 24-28, 2012. Proceedings, Part II*, volume 7524 of *Lecture Notes in Computer Science*, pages 846–849. Springer, 2012.
[14] E. Balkanski, A. Rubinstein, and Y. Singer. The limitations of optimization from samples. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 1016–1027. ACM, 2017.
[15] E. Balkanski and Y. Singer. The adaptive complexity of maximizing a submodular function. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 1138–1151. ACM, 2018.
[16] E. Balkanski and Y. Singer. Approximation guarantees for adaptive sampling. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 393–402. PMLR, 2018.
[17] K. Banihashem, L. Biabani, S. Goudarzi, M. Hajiaghayi, P. Jabbarzade, and M. Monemizadeh. Dynamic constrained submodular optimization with polylogarithmic update time. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 1660–1691. PMLR, 23–29 Jul 2023.
[18] K. Banihashem, L. Biabani, S. Goudarzi, M. Hajiaghayi, P. Jabbarzade, and M. Monemizadeh. Dynamic non-monotone submodular maximization. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.
[19] O. Barinova, V. S. Lempitsky, and P. Kohli. On detection of multiple object instances using hough transforms. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(9):1773–1784, 2012.
[20] S. Baswana. Dynamic algorithms for graph spanners. In *Algorithms - ESA 2006, 14th Annual European Symposium, Zurich, Switzerland, September 11-13, 2006, Proceedings*, volume 4168 of *Lecture Notes in Computer Science*, pages 76–87. Springer, 2006.
[21] S. Baswana, S. Khurana, and S. Sarkar. Fully dynamic randomized algorithms for graph spanners. *ACM Trans. Algorithms*, 8(4):35:1–35:51, 2012.
[22] M. Bateni, L. Chen, H. Esfandiari, T. Fu, V. S. Mirrokni, and A. Rostamizadeh. Categorical feature compression via submodular optimization. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 515–523. PMLR, 2019.
[23] M. Bateni, M. T. Hajiaghayi, and M. Zadimoghaddam. Submodular secretary problem and extensions. *ACM Trans. Algorithms*, 9(4):32:1–32:23, 2013.
[24] S. Behnezhad. Dynamic algorithms for maximum matching size. *CoRR*, abs/2207.07607, 2022.
[25] S. Behnezhad, M. Derakhshan, M. Hajiaghayi, C. Stein, and M. Sudan. Fully dynamic maximal independent set with polylogarithmic update time. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 382–405. IEEE Computer Society, 2019.
[26] A. Bernstein. Fully dynamic (2 + epsilon) approximate all-pairs shortest paths with fast query and close to linear update time. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 693–702. IEEE Computer Society, 2009.
[27] A. Bernstein. *Dynamic Algorithms for Shortest Paths and Matching*. PhD thesis, Columbia University, USA, 2016.
[28] A. Bernstein. Dynamic approximate-apsp. In *Encyclopedia of Algorithms*, pages 602–605. 2016.
[29] A. Bernstein, A. Dudeja, and Z. Langley. A framework for dynamic matching in weighted graphs. In *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 668–681. ACM, 2021.
[30] A. Bernstein, S. Forster, and M. Henzinger. A deamortization approach for dynamic spanner and dynamic maximal matching. *ACM Trans. Algorithms*, 17(4):29:1–29:51, 2021.
[31] A. Bernstein and C. Stein. Fully dynamic matching in bipartite graphs. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, volume 9134 of *Lecture Notes in Computer Science*, pages 167–179. Springer, 2015.
[32] A. Bernstein, J. van den Brand, M. P. Gutenberg, D. Nanongkai, T. Saranurak, A. Sidford, and H. Sun. Fully-dynamic graph sparsifiers against an adaptive adversary. In *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 20:1–20:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.
[33] S. Bhattacharya, F. Grandoni, J. Kulkarni, Q. C. Liu, and S. Solomon. Fully dynamic ($\Delta + 1$)-coloring in $O(1)$ update time. *ACM Trans. Algorithms*, 18(2):10:1–10:25, 2022.
[34] S. Bhattacharya, M. Henzinger, and D. Nanongkai. Fully dynamic approximate maximum matching and minimum vertex cover in $O(\log^3 n)$ worst case update time. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms,*

SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19, pages 470–489. SIAM, 2017.

[35] S. Bhattacharya, M. Henzinger, and D. Nanongkai. A new deterministic algorithm for dynamic set cover. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 406–423. IEEE Computer Society, 2019.

[36] S. Bhattacharya, M. Henzinger, D. Nanongkai, and C. E. Tsourakakis. Space- and time-efficient algorithm for maintaining dense subgraphs on one-pass dynamic streams. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 173–182. ACM, 2015.

[37] S. Bhattacharya, M. Henzinger, D. Nanongkai, and X. Wu. Dynamic set cover: Improved amortized and worst-case update time. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2537–2549. SIAM, 2021.

[38] S. Bhattacharya and P. Kiss. Deterministic rounding of dynamic fractional matchings. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 27:1–27:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[39] S. Bhattacharya, P. Kiss, T. Saranurak, and D. Wajc. Dynamic matching with better-than-2 approximation in polylogarithmic update time. *CoRR*, abs/2207.07438, 2022.

[40] G. Bodwin and S. Krinninger. Fully dynamic spanners with worst-case update time. In *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, volume 57 of *LIPIcs*, pages 17:1–17:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

[41] N. Buchbinder, M. Feldman, and R. Schwartz. Online submodular maximization with preemption. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1202–1216. SIAM, 2015.

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

[43] A. Chakrabarti and S. Kale. Submodular maximization meets streaming: Matchings, matroids, and more. *Mathematical Programming*, 154(1):225–247, 2015.

[44] M. Charikar and S. Solomon. Fully dynamic almost-maximal matching: Breaking the polynomial worst-case time barrier. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 33:1–33:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[45] A. Chaturvedi, H. L. Nguyen, and L. Zakynthinou. Differentially private decomposable submodular maximization. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 6984–6992. AAAI Press, 2021.

[46] S. Chechik and T. Zhang. Fully dynamic maximal independent set in expected poly-log update time. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 370–381. IEEE Computer Society, 2019.

[47] S. Chechik and T. Zhang. Dynamic low-stretch spanning trees in subpolynomial time. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 463–475. SIAM, 2020.

[48] C. Chekuri, S. Gupta, and K. Quanrud. Streaming algorithms for submodular function maximization. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, volume 9134 of *Lecture Notes in Computer Science*, pages 318–330. Springer, 2015.

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

[50] C. Chekuri, J. Vondrák, and R. Zenklusen. Multi-budgeted matchings and matroid intersection via dependent rounding. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 1080–1097. SIAM, 2011.

[51] L. Chen, G. Goranci, M. Henzinger, R. Peng, and T. Saranurak. Fast dynamic cuts, distances and effective resistances via vertex sparsifiers. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1135–1146. IEEE, 2020.

[52] X. Chen and B. Peng. On the complexity of dynamic submodular maximization. In *Proceedings of the Fifty-Fourth Annual ACM on Symposium on Theory of Computing, STOC 2022, to appear*, 2022.

[53] Y. Chen, H. Shioi, C. F. Montesinos, L. P. Koh, S. A. Wich, and A. Krause. Active detection via adaptive submodularity. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, volume 32 of *JMLR Workshop and Conference Proceedings*, pages 55–63. JMLR.org, 2014.

[54] R. Chitnis, G. Cormode, H. Esfandiari, M. Hajiaghayi, A. McGregor, M. Monemizadeh, and S. Vorotnikova. Kernelization via sampling with applications to finding matchings and related problems in dynamic graph streams. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1326–1344. SIAM, 2016.

[55] R. Chitnis, M. Cygan, M. Hajiaghayi, M. Pilipczuk, and M. Pilipczuk. Designing fpt algorithms for cut problems using randomized contractions. *SIAM Journal on Computing*, 45(4):1171–1229, 2016.

[56] R. H. Chitnis, G. Cormode, M. T. Hajiaghayi, and M. Monemizadeh. Parameterized streaming: Maximal matching and vertex cover. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1234–1251. SIAM, 2015.

[57] J. Chuzhoy, Y. Gao, J. Li, D. Nanongkai, R. Peng, and T. Saranurak. A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1158–1167. IEEE, 2020.

[58] M. Cygan, F. V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized algorithms*, volume 5. Springer, 2015.

[59] R. da Ponte Barbosa, A. Ene, H. L. Nguyen, and J. Ward. The power of randomization: Distributed submodular maximization on massive datasets. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 1236–1244. JMLR.org, 2015.

[60] R. da Ponte Barbosa, A. Ene, H. L. Nguyen, and J. Ward. A new framework for distributed submodular maximization. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 645–654. IEEE Computer Society, 2016.

[61] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer Publishing Company, Incorporated, 2012.

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

[63] P. Duetting, F. Fusco, S. Lattanzi, A. Norouzi-Fard, and M. Zadimoghaddam. Deletion robust submodular maximization over matroids. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 5671–5693. PMLR, 2022.

[64] D. Durfee, Y. Gao, G. Goranci, and R. Peng. Fully dynamic spectral vertex sparsifiers and applications. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 914–925. ACM, 2019.

[65] P. Dütting, F. Fusco, S. Lattanzi, A. Norouzi-Fard, and M. Zadimoghaddam. Fully dynamic submodular maximization over matroids. *arXiv preprint arXiv:2305.19918*, 2023.

[66] J. Edmonds. Matroids and the greedy algorithm. *Math. Program.*, 1(1):127–136, 1971.

[67] J. Edmonds and D. R. Fulkerson. Transversals and matroid partition. *Journal of Research of the National Bureau of Standards Section B Mathematics and Mathematical Physics*, page 147, 1965.

[68] S. Ehsani, M. Hajiaghayi, T. Kesselheim, and S. Singla. Prophet secretary for combinatorial auctions and matroids. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 700–714. SIAM, 2018.

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

[70] E. R. Elenberg, A. G. Dimakis, M. Feldman, and A. Karbasi. Streaming weak submodularity: Interpreting neural networks on the fly. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 4044–4054, 2017.

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

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

[73] A. Ene, H. L. Nguyen, and L. A. Végh. Decomposable submodular function minimization: Discrete and continuous. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 2870–2880, 2017.

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

[75] S. Fafianie and S. Kratsch. Streaming kernelization. In *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part II*, volume 8635 of *Lecture Notes in Computer Science*, pages 275–286. Springer, 2014.

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

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

[78] M. Feldman, P. Liu, A. Norouzi-Fard, O. Svensson, and R. Zenklusen. Streaming submodular maximization under matroid constraints. *arXiv preprint arXiv:2107.07183*, 2021.

[79] M. Feldman, A. Norouzi-Fard, O. Svensson, and R. Zenklusen. The one-way communication complexity of submodular maximization with applications to streaming and robustness. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 1363–1374. ACM, 2020.

[80] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.

[81] F. V. Fomin and T. Korhonen. Fast fpt-approximation of branchwidth. In *Proceedings of 54th Annual ACM Symposium on Theory of Computing (STOC)*, 2022. to appear.

[82] Y. Gao, Y. P. Liu, and R. Peng. Fully dynamic electrical flows: Sparse maxflow faster than goldberg-rao. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 516–527. IEEE, 2021.

[83] S. O. Gharan and J. Vondrák. On variants of the matroid secretary problem. *Algorithmica*, 67(4):472–497, 2013.

[84] A. Gupta, R. Krishnaswamy, A. Kumar, and D. Panigrahi. Online and dynamic algorithms for set cover. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 537–550. ACM, 2017.

[85] A. Gupta, E. Lee, and J. Li. An fpt algorithm beating 2-approximation for k-cut. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '18, page 2821–2837, USA, 2018. Society for Industrial and Applied Mathematics.

[86] A. Gupta and R. Levin. Fully-dynamic submodular cover with bounded recourse. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1147–1157. IEEE, 2020.

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

[88] K. Han, Z. Cao, S. Cui, and B. Wu. Deterministic approximation for submodular maximization over a matroid in nearly linear time. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[89] N. J. A. Harvey, C. Liaw, and T. Soma. Improved algorithms for online submodular maximization via first-order regret bounds. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[90] M. Henzinger, S. Krinninger, and D. Nanongkai. Dynamic approximate all-pairs shortest paths: Breaking the o(mn) barrier and derandomization. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 538–547. IEEE Computer Society, 2013.

[91] M. Henzinger, S. Krinninger, and D. Nanongkai. Dynamic approximate all-pairs shortest paths: Breaking the o(mn) barrier and derandomization. *SIAM J. Comput.*, 45(3):947–1006, 2016.

[92] M. Henzinger, S. Krinninger, and D. Nanongkai. Dynamic approximate all-pairs shortest paths: Breaking the o(mn) barrierand derandomization. In *Encyclopedia of Algorithms*, pages 600–602. 2016.

[93] M. R. Henzinger and V. King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *J. ACM*, 46(4):502–516, 1999.

[94] B. M. Kapron, V. King, and B. Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1131–1142. SIAM, 2013.

[95] N. Kashyap. Code decomposition: Theory and applications. In *2007 IEEE International Symposium on Information Theory*, pages 481–485, 2007.

[96] K.-i. Kawarabayashi and M. Thorup. The minimum k-way cut of bounded size is fixed-parameter tractable. In *Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, FOCS '11, page 160–169, USA, 2011. IEEE Computer Society.

[97] E. Kazemi, M. Mitrovic, M. Zadimoghaddam, S. Lattanzi, and A. Karbasi. Submodular streaming in all its glory: Tight approximation, minimum memory and low adaptive complexity. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 3311–3320. PMLR, 2019.

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

[99] D. Kempe, J. M. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 24 - 27, 2003*, pages 137–146. ACM, 2003.

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

[101] R. Kleinberg and S. M. Weinberg. Matroid prophet inequalities. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 123–136. ACM, 2012.

[102] R. Kleinberg and S. M. Weinberg. Matroid prophet inequalities and applications to multi-dimensional mechanism design. *Games Econ. Behav.*, 113:97–115, 2019.

[103] T. Korhonen. A single-exponential time 2-approximation algorithm for treewidth. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 184–192. IEEE, 2021.

[104] A. Krause. Submodularity in machine learning and vision. In *British Machine Vision Conference, BMVC 2013, Bristol, UK, September 9-13, 2013*. BMVA Press, 2013.

[105] R. Kumar, B. Moseley, S. Vassilvitskii, and A. Vattani. Fast greedy algorithms in mapreduce and streaming. *ACM Trans. Parallel Comput.*, 2(3):14:1–14:22, 2015.

[106] L. Kumari and J. A. Bilmes. Submodular span, with applications to conditional data summarization. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 12344–12352. AAAI Press, 2021.

[107] R. Kupfer, S. Qian, E. Balkanski, and Y. Singer. The adaptive complexity of maximizing a gross substitutes valuation. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[108] B. Kveton, Z. Wen, A. Ashkan, H. Eydgahi, and B. Eriksson. Matroid bandits: Fast combinatorial optimization with learning. In *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence, UAI 2014, Quebec City, Quebec, Canada, July 23-27, 2014*, pages 420–429. AUAI Press, 2014.

[109] S. Lattanzi, S. Mitrovic, A. Norouzi-Fard, J. Tarnawski, and M. Zadimoghaddam. Fully dynamic algorithm for constrained submodular optimization. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[110] S. Lattanzi, S. Mitrovic, A. Norouzi-Fard, J. Tarnawski, and M. Zadimoghaddam. Fully dynamic algorithm for constrained submodular optimization. *CoRR*, abs/2006.04704v2, 2023.

[111] J. Lee, M. Sviridenko, and J. Vondrák. Matroid matching: the power of local search. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 369–378. ACM, 2010.

[112] J. Lee, M. Sviridenko, and J. Vondrák. Submodular maximization over multiple matroids via generalized exchange properties. *Math. Oper. Res.*, 35(4):795–806, 2010.

[113] M. W. Libbrecht, J. A. Bilmes, and W. S. Noble. Choosing non-redundant representative subsets of protein sequence data sets using submodular optimization. In *Proceedings of the 2018 ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics, BCB 2018, Washington, DC, USA, August 29 - September 01, 2018*, page 566. ACM, 2018.

[114] H. Lin and J. A. Bilmes. A class of submodular functions for document summarization. In *The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Conference, 19-24 June, 2011, Portland, Oregon, USA*, pages 510–520. The Association for Computer Linguistics, 2011.

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

[116] S. MacLane. Some interpretations of abstract linear dependence in terms of projective geometry. *American Journal of Mathematics*, 58(1):236–240, 1936.

[117] F. Maeda and S. Maeda. *Matroid Lattices*, pages 56–71. Springer Berlin Heidelberg, Berlin, Heidelberg, 1970.

[118] D. Magos, I. Mourtos, and L. S. Pitsoulis. The matching predicate and a filtering scheme based on matroids. *J. Comput.*, 1(6):37–42, 2006.

[119] D. Marx. Parameterized complexity and approximation algorithms. *The Computer Journal*, 51(1):60–78, 2008.

[120] A. McGregor and H. T. Vu. Better streaming algorithms for the maximum coverage problem. *Theory Comput. Syst.*, 63(7):1595–1619, 2019.

[121] G. J. MINTY. On the axiomatic foundations of the theories of directed linear graphs, electrical networks and network-programming. *Journal of Mathematics and Mechanics*, 15(3):485–520, 1966.

[122] V. S. Mirrokni and M. Zadimoghaddam. Randomized composable core-sets for distributed submodular maximization. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 153–162. ACM, 2015.

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

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

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

[126] D. Nanongkai and T. Saranurak. Dynamic spanning forest with worst-case update time: adaptive, las vegas, and $o(n^{1/2 - \epsilon})$-time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 1122–1129. ACM, 2017.

[127] D. Nanongkai, T. Saranurak, and C. Wulff-Nilsen. Dynamic minimum spanning forest with subpolynomial worst-case update time. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 950–961. IEEE Computer Society, 2017.

[128] O. Neiman and S. Solomon. Simple deterministic algorithms for fully dynamic maximal matching. *ACM Trans. Algorithms*, 12(1):7:1–7:15, 2016.

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

[130] K. Onak and R. Rubinfeld. Maintaining a large matching and a small vertex cover. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 457–464. ACM, 2010.

[131] J. G. Oxley. *Matroid theory*. Oxford University Press, 1992.

[132] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1982.

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

[134] G. Radanovic, A. Singla, A. Krause, and B. Faltings. Information gathering with peers: Submodular optimization with peer-prediction constraints. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 1603–1610. AAAI Press, 2018.

[135] M. Rauch. Fully dynamic biconnectivity in graphs. In *33rd Annual Symposium on Foundations of Computer Science, Pittsburgh, Pennsylvania, USA, 24-27 October 1992*, pages 50–59. IEEE Computer Society, 1992.

[136] S. Sawlani and J. Wang. Near-optimal fully dynamic densest subgraph. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 181–193. ACM, 2020.

[137] J. M. Schreiber, J. A. Bilmes, and W. S. Noble. apricot: Submodular selection for data summarization in python. *J. Mach. Learn. Res.*, 21:161:1–161:6, 2020.

[138] R. Sipos, A. Swaminathan, P. Shivaswamy, and T. Joachims. Temporal corpus summarization using submodular word coverage. In *21st ACM International Conference on Information and Knowledge Management, CIKM'12, Maui, HI, USA, October 29 - November 02, 2012*, pages 754–763. ACM, 2012.

[139] S. Solomon. Fully dynamic maximal matching in constant update time. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 325–334. IEEE Computer Society, 2016.

[140] S. Solomon and N. Wein. Improved dynamic graph coloring. *ACM Trans. Algorithms*, 16(3):41:1–41:24, 2020.

[141] S. Stan, M. Zadimoghaddam, A. Krause, and A. Karbasi. Probabilistic submodular maximization in sub-linear time. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 3241–3250. PMLR, 2017.

[142] E. Tohidi, R. Amiri, M. Coutino, D. Gesbert, G. Leus, and A. Karbasi. Submodularity in action: From machine learning to signal processing applications. *IEEE Signal Process. Mag.*, 37(5):120–133, 2020.

[143] J. van den Brand, Y. Gao, A. Jambulapati, Y. T. Lee, Y. P. Liu, R. Peng, and A. Sidford. Faster maxflow via improved dynamic spectral vertex sparsifiers. In *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 543–556. ACM, 2022.

[144] J. van den Brand and D. Nanongkai. Dynamic approximate shortest paths and beyond: Subquadratic and worst-case update time. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 436–455. IEEE Computer Society, 2019.

[145] K. Wei, R. K. Iyer, and J. A. Bilmes. Submodularity in data subset selection and active learning. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 1954–1963. JMLR.org, 2015.

[146] W. Yang, J. A. Bilmes, and W. S. Noble. Submodular sketches of single-cell rna-seq measurements. In *BCB '20: 11th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics, Virtual Event, USA, September 21-24, 2020*, pages 61:1–61:6. ACM, 2020.

[147] G. Zhang, N. Tatti, and A. Gionis. Coresets remembered and items forgotten: submodular maximization with deletions. In *2022 IEEE International Conference on Data Mining (ICDM)*, pages 676–685. IEEE, 2022.

A Some of the Proofs regarding Invariants of the algorithm for submodular matroid maximization

A.1 Proof of Theorem 3.1

LEMMA A.1. (SURVIVOR INVARIANT) *If before calling* MATROIDCONSTRUCTLEVEL(j)*, the level invariants partially hold for the first j levels, then after its execution, the survivor invariant fully holds.*

Proof. First of all, we assume that $R_j \neq \emptyset$, otherwise $T = j - 1$ and we are done. As we have in Algorithm MATROIDCONSTRUCTLEVEL, let P be a random permutation of the set R_j. Let us fix an arbitrary element $e \in P$ and suppose that at the time when we see $e \in P$, the current level is L_ℓ for $\ell \geq j$. We have two cases. Either e is a promoting element for the level $L_{\ell-1}$ or it is not promoting for the level $L_{\ell-1}$.

First, assume that e is a promoting element for the level $L_{\ell-1}$. We then let e_ℓ be e, perform a set of computations, and then start the new level. In particular, the element e is not added to $R_{\ell+1}$ and so, it will not appear in any set $R_{z>\ell}$. Recall that Lemma 3.1 proves if e is not a promoting element with respect to a level L_z, it will not be a promoting element for the next level L_x where $z \leq x \leq T$. On the other hand, since e is a promoting element for the level $L_{\ell-1}$, we add e to all previous sets $R_{j+1}, \cdots, R_\ell$.

Next, we consider the latter case where e is not a promoting element for the level $L_{\ell-1}$. That is, PROMOTE($e, L_{\ell-1}$) is $False$. This essentially means that if we inductively apply the argument of Lemma 3.1, there exists an integer $z \in [j, \ell)$ for which BOOLPROMOTE(e, L_{z-1}) is $True$, but BOOLPROMOTE(e, L_z) is $False$. This means e is a promoting element for all levels $L_j, \cdots, L_{z-1}$ and it is not promoting for levels $L_z, \cdots, L_T$. According to function MATROIDCONSTRUCTLEVEL, we insert the element e into sets $R_{j+1}, \cdots, R_z$. Hence, after the execution of MATROIDCONSTRUCTLEVEL(j), the survivor invariant holds. □

LEMMA A.2. (INDEPENDENT INVARIANT) *If before calling* MATROIDCONSTRUCTLEVEL(j)*, the level invariants partially hold for the first j levels, then after its execution, the independent invariant fully holds.*

Proof. In the execution of MATROIDCONSTRUCTLEVEL(j), the variable ℓ is set to $j, j+1, \cdots, T, T+1$. Therefore, for each $\ell \in [j, T]$, we set I_ℓ to $(I_{\ell-1} + e_\ell) \backslash y$ in Line 11, where y is defined as PROMOTE($I_{\ell-1}, I'_{\ell-1}, e, w[I_{\ell-1}]$) in Line 9. Adding this to the assumption of lemma implies that $I_\ell = (I_{\ell-1} + e_\ell) -$ PROMOTE($I_{\ell-1}, I'_{\ell-1}, e, w[I_{\ell-1}]$) holds for any $\ell \in [T]$.

Next, we prove $I'_\ell = \cup_{m \leq \ell} I_m$ for any $\ell \in [T]$ using induction. By the assumption of lemma, $I'_\ell = \cup_{m \leq \ell} I_m$ holds for any $i \leq j - 1$. For the induction step, assume $\ell \in [j, T]$ and $I'_{\ell-1} = \cup_{m \leq \ell-1} I_m$ holds. In Line 11 we set $I_\ell = (I_{\ell-1} + e) \backslash y$. Since $e \notin I + \ell - 1$, it means $I_\ell \setminus I_{\ell-1} = e$. We then set $I'_\ell = I'_{\ell-1} + e$ in Line 11. Putting everything together we have

$$I'_\ell = I'_{\ell-1} + e = \cup_{m \leq \ell-1} I_m + e = \cup_{m \leq \ell-1} I_m + (I_\ell \setminus I_{\ell-1}) = \cup_{m \leq \ell} I_m \quad .$$

It completes the proof the of lemma. □

LEMMA A.3. (WEIGHT INVARIANT) *If before calling* MATROIDCONSTRUCTLEVEL(j)*, the level invariants partially hold for the first j levels, then after its execution, the weight invariant fully holds.*

Proof. To prove the lemma, we need to show $e_\ell \in R_\ell$ and $w(e_\ell) = f(I'_{\ell-1} + e_\ell) - f(I'_{\ell-1})$ hold for each $\ell \in [j, T]$. Recall that in the execution of MATROIDCONSTRUCTLEVEL(j), after constructing the level L_ℓ, we increase the variable ℓ by one. Hence, the variable ℓ is set to $j, j+1, \cdots, T, T+1$ during the execution of MATROIDCONSTRUCTLEVEL(j).

We first prove $e_\ell \in R_\ell$ holds for each $\ell \in [j, T]$. Let ℓ be a fixed integer in $[j, T]$. Since e_ℓ is an element of P, and P is a random permutation of R_j, we have $e_\ell \in R_j$. We know that PROMOTE($I_{\ell-1}, I'_{\ell-1}, e_\ell, w[I_{\ell-1}]$) $\neq$ FAIL. Then the monotone property that we prove in Lemma 3.1 implies that PROMOTE($I_{m-1}, I'_{m-1}, e_\ell, w[I_{m-1}]$) $\neq$ FAIL for any $m \in [i, \ell]$. Also, it is obvious that $e_\ell \neq e_m$ when $m < \ell$. Recall that $R_m = \{e \in R_{m-1} - e_{m-1} :$ PROMOTE($I_{m-1}, I'_{m-1}, e, w[I_{m-1}]$) $\neq$ FAIL$\}$. Therefore, by a simple induction on m, we can show that $e_\ell \in R_m$ holds for each $m \in [i, \ell]$, which implies $e_\ell \in R_\ell$.

Moreover, we fix the weight $w(e_\ell) = f(I'_{\ell-1} + e_\ell) - f(I'_{\ell-1})$ for each $\ell \in [j, T]$ in Line 10. Adding this to the assumption of Lemma finishes the proof. □

LEMMA A.4. (TERMINATOR INVARIANT) *If before calling* MATROIDCONSTRUCTLEVEL(j)*, the level invariants partially hold for the first j levels, then after its execution, the terminator invariant fully holds.*

Proof. According to Line 15 and the variable z, if we add an element e to R_r at some point of time, then $r \leq z \leq \ell - 1$ holds at that moment. Since the variable ℓ never decreases during the execution of MATROIDCONSTRUCTLEVEL(j) and we return $\ell - 1$ as T at the end, we can conduct that no element has been added to R_{T+1}, and then $R_{T+1} = \emptyset$, which means the terminator invariant holds. □

A.2 Proof of Lemma 3.4 To prove the lemma, we first mention some useful facts and then show that the starter, weight, independent, and survivor invariants partially hold. Finally, we prove that all level invariants hold.

We begin with defining variables i^* and j^* as follows.

- i^***:** If during the execution of INSERT(v) there is $i \in [T]$ such that e_i has been set to be v, which also implies that we have invoked MATROIDCONSTRUCTLEVEL($i + 1$), then we set i^* to be i. Otherwise, we set i^* to be $T + 1$.
- j^***:** Let j^* be the largest $i \in [0, T^- + 1]$ such that we have added v to R_i^-.

We consider these two cases in this proof.

- Case 1: $i^* \leq T$, which means $e_{i^*} = v$ and therefore $j^* = i^*$. It also means that we have invoked MatroidConstructLevel($i^* + 1$).
- Case 2: $i^* = T + 1$, which means MatroidConstructLevel has never been invoked during the insertion of v. Note that in this case, $T = T^-$ and it is not possible for j^* to be equal to $T^- + 1$, since that would have caused invoking MatroidConstructLevel. Thus, $j^* < T^- + 1 = T + 1 = i^*$.

Considering our algorithm in Insert(v), it is clear that for any $i < i^*$, we have not made any kind of change in e_i^-, $w(e_i)$, I_i^-, or $I_i'^-$ at least until MatroidConstructLevel is invoked, if it ever gets invoked. Additionally, according to MatroidConstructLevel, we know that if we have invoked MatroidConstructLevel($i^* + 1$), there has not been any alteration to the variables regarding previous levels. Hence, we can conduct the following facts.

Fact A.1. *For any $i \in [1, i^*)$, we have $e_i = e_i^-$.*

Fact A.2. *For any $i \in [1, i^*)$, we have $w(e_i) = w^-(e_i)$.*

Fact A.3. *For any $i \in [0, i^*)$, we have $I_i = I_i^-$.*

Fact A.4. *For any $i \in [0, i^*)$, we have $I'_i = I'^-_i$.*

By the definition of j^*, we have added the element v to the set R_i^-, for each $i \in [0, j^*]$. Recall that $j^* \leq i^*$, and by invoking MatroidConstructLevel($i^* + 1$), there has not been any alteration to the variables regarding previous levels. It leads to the following fact.

Fact A.5. *For any $i \in [0, j^*]$, we have $R_i = R_i^- + v$.*

We know that if Case 2 holds, which means MatroidConstructLevel has never been invoked during the insertion of v, we have $R_i = R_i^-$ for any $i \in [j^* + 1, T + 1]$. Recall that in Case 2, $i^* = T + 1$, and therefore $[j^* + 1, T + 1] = [j^* + 1, i^*]$. Also if Case 1 holds, $j^* = i^*$, so $[j^* + 1, i^*] = \emptyset$. Thus, independent of the case, we can have the following fact.

Fact A.6. *For any $i \in [j^* + 1, i^*]$, we have $R_i = R_i^-$.*

In the following, we first prove that the starter invariant holds after executing Insert(v). We next show that the weight, independent, and survivor invariants partially hold for the first $i^* + 1$ levels. Finally, we complete the proof by proving that all the level invariants hold.

Starter invariant. To show that the starter invariant holds after Insert(v), we need to prove $R_0 = V$ and $I_0 = I'_0 = \emptyset$. By the assumption of this lemma, we have $R_0^- = V^-$, and Fact A.5 results that $R_0 = R_0^- + v$. Thus $R_0 = R_0^- + v = V^- + v = V$.

Again by the assumption of this lemma, $I_0^- = I'^-_0 = \emptyset$. Due to Fact A.3, we have $I_0 = I_0^-$, and therefore, it is clear that $I_0 = I_0^- = \emptyset$. Similarly, we have $I'_0 = I'^-_0$ because of Fact A.4, and then $I'_0 = I'^-_0 = \emptyset$.

Weight invariant (partially). To show that weight invariant partially holds for the first i^* levels, we first prove $e_i \in R_i$ and we prove then $w(e_i) = f(I'_{i-1} + e_i) - f(I'_{i-1})$.

By the assumption of this lemma, we know that $e_i^- \in R_i^-$ for $i \in [1, i^*)$. Besides, according to Fact A.6 and Fact A.5, for any $i \in [0, i^*]$, either $R_i = R_i^- + v$ or $R_i = R_i^-$, and thus $R_i^- \subseteq R_i$. Also for any $i \in [1, i^*)$ we have $e_i = e_i^-$ by Fact A.1. Putting everything together, we have $e_i = e_i^- \in R_i^- \subseteq R_i$ for any $i \in [1, i^*)$. which means $e_i \in R_i$.

Next we need to show $w(e_i) = f(I'_{i-1} + e_i) - f(I'_{i-1})$. For any $i \in [1, i^*)$, we have $w(e_i) = w^-(e_i)$ by Fact A.2. Besides, $e_i = e_i^-$ by Fact A.1, and then $w^-(e_i) = w^-(e_i^-)$. Moreover, the assumption of this lemma implies that $w^-(e_i^-) = f(I'^-_{i-1} + e_i^-) - f(I'^-_{i-1})$. Also $I'^-_{i-1} = I'_{i-1}$ for any $i \in [1, i^*)$ because of Fact A.4. Adding it to $e_i = e_i^-$ implies $f(I'^-_{i-1} + e_i^-) - f(I'^-_{i-1}) = f(I'_{i-1} + e_i) - f(I'_{i-1})$. Putting everything together, for any $i \in [1, i^*)$ we have

$$w(e_i) = w^-(e_i) = w^-(e_i^-) = f(I'^-_{i-1} + e_i^-) - f(I'^-_{i-1}) = f(I'_{i-1} + e_i) - f(I'_{i-1}) \ .$$

Independent invariant (partially). Now we show that the independent invariant partially holds for the first i^* levels after Insert(v). To do this, we prove $I_i = I_{i-1} + e_i - \text{Promote}(I_{i-1}, I'_{i-1}, e_i, w[I_{i-1}])$ and $I'_i = \cup_{j \leq i} I_j$ holds for all $i \in [1, i^*)$.

Using Fact A.3, we have $I_i = I_i^-$ for any $i \in [1, i^*)$. Also, $I_i^- = I_{i-1}^- + e_i^- - \text{Promote}(I_{i-1}^-, I'^-_{i-1}, e_i^-, w^-[I_{i-1}^-])$ by the assumption of this lemma. Then,

$$I_i = I_i^- = I_{i-1}^- + e_i^- - \text{Promote}(I_{i-1}^-, I'^-_{i-1}, e_i^-, w^-[I_{i-1}^-]) \ .$$

Recall that $w[X]$ means w restricted to domain X. Now, we show that $w^-[I_{i-1}^-] = w[I_{i-1}]$ holds for any $i \in [1, i^*)$. By Fact A.3, $I_{i-1} = I_{i-1}^-$, which means the domain of $w^-[I_{i-1}^-]$ and $w[I_{i-1}]$ are equal. Moreover, since the independent invariant holds before the insertion of v by the assumption of this lemma, we have $I_{i-1}^- \subseteq \{e_1^-, \cdots, e_{i-1}^-\}$. Besides, for any $j \leq i - 1 < i^*$, Fact A.1 implies that $e_j^- = e_j$, which results in $I_{i-1}^- \subseteq \{e_1, \cdots, e_{i-1}\}$. We also have $w(e_j) = w^-(e_j)$ for any $j \leq i - 1 < i^*$ using Fact A.2. It completes the proof of $w^-[I_{i-1}^-] = w[I_{i-1}]$. Hence,

$$I_i = I_{i-1}^- + e_i^- - \text{Promote}(I_{i-1}^-, I'^-_{i-1}, e_i^-, w[I_{i-1}]) \ .$$

For any $i \in [1, i^*)$, we have $I_{i-1} = I^-_{i-1}$ by Fact A.3, $I'_{i-1} = I'^-_{i-1}1$ by Fact A.4, and $e_i = e^-_i$ by Fact A.1. Then we can further conclude that:

$$I_i = I_{i-1} + e_i - \text{Promote}(I_{i-1}, I'_{i-1}, e_i, w[I_{i-1}]) \ .$$

Finally, we prove $I'_i = \cup_{j \le i} I_j$ for all $i \in [1, i^*)$. Because of Fact A.4, $I'_i = I'^-_i$ holds for any $i \in [1, i^*)$. Besides, $I'^-_i = \cup_{j \le i} I^-_j$ a result of the assumption of this lemma. Also, $I^-_j = I_j$ for any $j \le i < i^*$ due to Fact A.3. Putting everything together we have

$$I'_i = I'^-_i = \cup_{j \le i} I^-_j = \cup_{j \le i} I_j \ .$$

Survivor invariant (partially). Next, we show that the survivor invariant partially holds for the first i^* levels by proving that $R_i = \{e \in R_{i-1} - e_{i-1} : \text{Promote}(I_{i-1}, I'_{i-1}, e, w[I_{i-1}]) \neq \text{Fail}\}$ holds for any $i \in [1, i^*]$. In the following, we first consider $i \in [1, j^*]$ and then $i \in [j^* + 1, i^*]$. Recall that j^* is the largest j that we added v to R_j in $\text{Insert}(v)$.

First we study $i \in [1, j^*]$. Using Fact A.5, $R_i = R^-_i + v$ holds for each $i \in [1, j^*]$, and $R^-_i = \{e \in R^-_{i-1} - e^-_{i-1} : \text{Promote}(I^-_{i-1}, I'^-_{i-1}, e, w^-[I^-_{i-1}]) \neq \text{Fail}\}$ according to the assumption of this lemma. Besides, by the definition of j^* and according to the break condition in Line 13, we can conduct that $\text{Promote}(I^-_{i-1}, I'^-_{i-1}, v, w^-[I^-_{i-1}]) \neq \text{Fail}$. Putting everything together we have,

$$\begin{aligned} R_i &= R^-_i + v \\ &= \{e \in R^-_{i-1} - e^-_{i-1} : \text{Promote}(I^-_{i-1}, I'^-_{i-1}, e, w^-[I^-_{i-1}]) \neq \text{Fail}\} + v \\ &= \{e \in R^-_{i-1} + v - e^-_{i-1} : \text{Promote}(I^-_{i-1}, I'^-_{i-1}, e, w^-[I^-_{i-1}]) \neq \text{Fail}\} \ . \end{aligned}$$

As stated in the previous part, we know that $w^-[I^-_{i-1}] = w[I_{i-1}]$ for any i that $i - 1 < j^* \le i^*$. Using this fact alongside $R_{i-1} = R^-_{i-1} + v$ by Fact A.5, $e_{i-1} = e^-_{i-1}$ by Fact A.1, $I_{i-1} = I^-_{i-1}$ by Fact A.3, and $I'_{i-1} = I'^-_{i-1}$ by Fact A.4, we have:

$$R_i = \{e \in R_{i-1} - e_{i-1} : \text{Promote}(I_{i-1}, I'_{i-1}, e, w[I_{i-1}]) \neq \text{Fail}\} \ .$$

Recall that if case 1 holds, $i^* = j^*$, and then the survivor invariant partially holds for the first i^* levels. Otherwise, if Case 2 holds, it remains to study $i \in [j^* + 1, i^*] = [j^* + 1, T + 1]$.

It holds that $R_i = R^-_i$ for any $i \in [j^*, i^*]$ according to Fact A.6. Adding it to the assumption of this lemma, we have:

$$R_i = R^-_i = \{e \in R^-_{i-1} - e^-_{i-1} : \text{Promote}(I^-_{i-1}, I'^-_{i-1}, e, w^-[I^-_{i-1}]) \neq \text{Fail}\} \ .$$

Besides, $e_{i-1} = e^-_{i-1}$ by Fact A.1, $I_{i-1} = I^-_{i-1}$ by Fact A.3, and $I'_{i-1} = I'^-_{i-1}$ by Fact A.4. Using these fact alongside $w^-[I^-_{i-1}] = w[I_{i-1}]$, which was stated beforehand we have

$$R_i = \{e \in R^-_{i-1} - e_{i-1} : \text{Promote}(I_{i-1}, I'_{i-1}, e, w[I_{i-1}]) \neq \text{Fail}\} \ .$$

We have either $i = j^* + 1$ or $i \in (j^* + 1, i^*]$. If $i \in (j^* + 1, i^*]$, then $R_{i-1} = R^-_{i-1}$ by Fact A.6. Hence,

$$R_i = \{e \in R_{i-1} - e_{i-1} : \text{Promote}(I_{i-1}, I'_{i-1}, e, w[I_{i-1}]) \neq \text{Fail}\} \ .$$

Now consider $i = j^* + 1$. According to Fact A.5, $R_{j^*} = R^-_{j^*} + v$, and then $R^-_{j^*} = R_{j^*} \setminus \{v\}$. Due to the definition of j^*, we know v is not added to R_{j^*+1}. Hence, according to the break condition in Line 13, we can conduct that $\text{Promote}(I_{j^*}, I'_{j^*}, e, w[I_{j^*}]) = \text{Fail}$. Putting everything we have

$$\begin{aligned} R_{j^*+1} &= \{e \in R^-_{j^*} - e_{j^*} : \text{Promote}(I_{j^*}, I'_{j^*}, e, w^-[I_{j^*}]) \neq \text{Fail}\} \\ &= \{e \in R_{j^*} \setminus \{v\} - e_{j^*} : \text{Promote}(I_{j^*}, I'_{j^*}, e, w^-[I_{j^*}]) \neq \text{Fail}\} \\ &= \{e \in R_{j^*} - e_{j^*} : \text{Promote}(I_{j^*}, I_{j^*}, e, w[I_{j^*}]) \neq \text{Fail}\} \ , \end{aligned}$$

as claimed.

Completing the proof. Now having everything in hand, we can complete the proof of this lemma. Above, we show that the starter, survivor, weight, and independent invariants partially hold for the first i^* levels.

If Case 1 holds, it means we set e_{i^*} to v. We also set $w(e_{i^*}) = f(I'_{i^*-1} + e_{i^*}) - f(I'_{i^*-1})$ in Line 18, $I_{i^*} = I_{i^*-1} + e_{i^*} - \text{Promote}(I_{i^*-1}, I'_{i^*-1}, e_{i^*}, w[I_{i^*-1}])$ and $I'_i = I'_i + e_{i^*}$ in Line 19. Then in the Line 20, we set $R_{i^*+1} = \{e' \in R_{i^*} : \text{Promote}(I_{i^*}, I'_{i^*}, e', w[I_{i^*}]) \neq \text{Fail}\}$. It implies that the level invariants partially hold for the first $i^* + 1$ levels. Next, we invoke $\text{MatroidConstructLevel}(i^* + 1)$, and then all the invariants hold by Theorem 3.1.

Otherwise, if Case 2 holds, $i^* = T + 1 = T^- + 1$. It means the starter, survivor, weight, and independent invariants hold and it remains to show that the terminator invariant holds to complete the proof. Recall that in this case, $j^* < T^- + 1$, which implies $(T^- + 1) \in [j^* + 1, i^*]$ and then $R_{T^-+1} = R^-_{T^-+1}$ by Fact A.5. Also, the assumption of this lemma implies $R^-_{T^-+1} = \emptyset$. Therefore, $R_{T+1} = R_{T^-+1} = R^-_{T^-+1} = \emptyset$, which means the terminator invariant holds and completes the proof.

A.3 Proof of Lemma 3.6 Let i^* be the level for which MatroidConstructLevel is invoked, and if MatroidConstructLevel is never invoked during the execution of Delete(v), we set i^* to be $T+1$. Considering our algorithm in Delete(v), we know that in levels before i^*, or in other words in each level $i \in [1, i^*)$ we do not make any change in our data structure other than removing the element v from R_i^- if it has this element in it. Hence, we have the following facts about e, w, I, and I', which are similar to the Facts A.1, A.2, A.3, and A.4 in the proof of Lemma 3.4.

Fact A.7. *For any $i \in [1, i^*)$, it holds that $e_i = e_i^-$.*

Fact A.8. *For any $i \in [1, i^*)$, it holds that $w(e_i) = w^-(e_i)$.*

Fact A.9. *For any $i \in [0, i^*)$, it holds that $I_i = I_i^-$.*

Fact A.10. *For any $i \in [0, i^*)$, it holds that $I'_i = I'^-_i$.*

In Delete(v), we removes the element v from R_i^- for any $i \in [0, \min(i^*, T^-)]$. By the definition of i^*, we have $i^* \leq T^- + 1$, and $i^* = T^- + 1$ happens only if MatroidConstructLevel has never been invoked during the execution of Delete(v). In this case, $R_{T^-+1} = R^-_{T^-+1}$, and since $R^-_{T^-+1} = \emptyset$ according to the assumption of this lemma, we have $R_{T^-+1} = R^-_{T^-+1} = \emptyset$. Therefore, $R_i = R_i^- \backslash \{v\}$ also holds when $i = T^- + 1$, and as $\min(i^*, T^- + 1) = i^*$, we can conduct the following fact.

Fact A.11. *For any $i \in [0, i^*]$, it holds that $R_i = R_i^- \backslash \{v\}$.*

Starter invariant. Similar to Lemma 3.4, we prove the starter invariant holds, which means $R_0 = V$ and $I_0 = I'_0 = \emptyset$.

Fact A.11 implies that $R_0 = R_0^- \backslash \{v\}$. Besides, $R_0^- = V^-$ since the starter invariant holds before the deletion of v by the assumption. We also know $V = V^- \backslash \{v\}$. Therefore, $R_0 = R_0^- \backslash \{v\} = V^- \backslash \{v\} = V$.

We have $I_0 = I_0^-$ by Fact A.9 and $I'_0 = I'^-_0$ by Fact A.10. Adding these to $I_0^- = I'^-_0 = \emptyset$, which is the assumption of lemma, implies that $I_0 = I'_0 = \emptyset$.

Weight invariant (partially). We next prove that the weight invariant partially holds for the first i^* levels. To do this, we show $e_i \in R_i$ and $w(e_i) = f(I'_{i-1} + e_i) - f(I'_{i-1})$ hold for any $i \in [1, i^*)$.

We first prove $e_i \in R_i$ holds for each $i \in (1, i^*)$. According to the definition of i^* and Delete Algorithm, we know that $e_i^- \neq v$ for any $i < i^*$. Moreover, the assumption of this lemma implies that $e_i^- \in R_i^-$. Hence we can say that $e_i^- \in R_i^- \backslash \{v\}$. We also know $R_i^- \backslash \{v\} = R_i$ for any $i \in [0, i^*)$ according to Fact A.11. Adding it to $e_i = e_i^-$, which is a result of Fact A.7, we have

$$e_i = e_i^- \in R_i^- \backslash \{v\} = R_i \ .$$

We next show $w(e_i) = f(I'_{i-1} + e_i) - f(I'_{i-1})$ in the same way of Lemma 3.4. For any $i \in [1, i^*)$, we have $w(e_i) = w^-(e_i)$ by Fact A.8, and $e_i = e_i^-$ by Fact A.7. Moreover, $w^-(e_i^-) = f(I'^-_{i-1} + e_i^-) - f(I'^-_{i-1})$ because of the assumption of this lemma. Also, $I'^-_{i-1} = I'_{i-1}$ holds for any $i \in [1, i^*)$ because of Fact A.10. Putting everything together, for any $i \in [1, i^*)$ we have

$$w(e_i) = w^-(e_i) = w^-(e_i^-) = f(I'^-_{i-1} + e_i^-) - f(I'^-_{i-1}) = f(I'_{i-1} + e_i) - f(I'_{i-1}) \ ,$$

which completes the proof.

Independent invariant (partially). We show that the independent invariant partially holds for the first i^* levels, which means $I_i = I_{i-1} + e_i - \textsc{Promote}(I_{i-1}, I'_{i-1}, e_i, w[I_{i-1}])$ and $I'_i = \cup_{j \leq i} I_j$ hold for any $i \in [1, i^*)$.

Same as Lemma 3.4, we first prove $I_i = I_{i-1} + e_i - \textsc{Promote}(I_{i-1}, I'_{i-1}, e_i, , w[I_{i-1}])$ holds for any $i \in [1, i^*)$. By the assumption of this lemma, $I_i^- = I_{i-1}^- + e_i^- - \textsc{Promote}(I_{i-1}^-, I'^-_{i-1}, e_i^-, w^-[I_{i-1}^-])$ holds. Besides, $I_i = I_i^-$ for any any $i \in [1, i^*)$ according to Fact A.9. Therefore,

$$I_i = I_i^- = I_{i-1}^- + e_i^- - \textsc{Promote}(I_{i-1}^-, I'^-_{i-1}, e_i^-, w^-[I_{i-1}^-]) \ .$$

Recall that $w[X]$ means w restricted to domain X. Again same as Lemma 3.4, we show that $w^-[I_{i-1}^-] = w[I_{i-1}]$ holds for any $i \in [1, i^*)$. By Fact A.9, $I_{i-1} = I_{i-1}^-$, which means the domain of $w^-[I_{i-1}^-]$ and $w[I_{i-1}]$ are equal. Moreover, since the independent invariant holds before the deletion of v by the assumption of this lemma, we have $I_{i-1}^- \subseteq \{e_1^-, \cdots, e_{i-1}^-\}$. Besides, for any $j \leq i-1 < i^*$, Fact A.7 implies that $e_j^- = e_j$, which results in $I_{i-1}^- \subseteq \{e_1, \cdots, e_{i-1}\}$. We also have $w(e_j) = w^-(e_j)$ for any $j \leq i-1 < i^*$ using Fact A.8. It completes the proof of $w^-[I_{i-1}^-] = w[I_{i-1}]$. Hence,

$$I_i = I_{i-1}^- + e_i^- - \textsc{Promote}(I_{i-1}^-, I'^-_{i-1}, e_i^-, w[I_{i-1}]) \ .$$

Adding it to $I_{i-1}^- = I_{i-1}$ by Fact A.9, $e_i^- = e_i$ by Fact A.7, and $I'^-_{i-1} = I'_{i-1}$ by Fact A.10, for any $i \in [1, i^*)$ we have

$$I_i = I_{i-1} + e_i - \textsc{Promote}(I_{i-1}, I'_{i-1}, e_i, w[I_{i-1}]) \ .$$

Next, we show $I'_i = \cup_{j \leq i} I_j$ holds for any $i \in [1, i^*)$. By Fact A.10, $I'_i = I'^-_i$ holds for any $i \in [1, i^*)$. Also, $I'^-_i = \cup_{j \leq i} I_j^-$ is a result of the assumption of this lemma. Moreover, Fact A.9 implies $I_j^- = I_j$ for any $j \leq i < i^*$. Therefore,

$$I'_i = I'^-_i = \cup_{j \leq i} I_j^- = \cup_{j \leq i} I_j \ .$$

Survivor invariant (partially). We next show that the survivor invariant partially holds for the first i^* levels. To do this, we prove $R_i = \{e \in R_{i-1} - e_{i-1} : \text{PROMOTE}(I_{i-1}, I'_{i-1}, e, w[I_{i-1}]) \neq \text{FAIL}\}$ holds for any $i \in [1, i^*]$.

Using Fact A.11, we have $R_i = R_i^- \backslash \{v\}$ for each $i \in [1, i^*]$. Also, the assumption of this lemma implies that $R_i^- = \{e \in R_{i-1}^- - e_{i-1}^- : \text{PROMOTE}(I_{i-1}^-, I'^-_{i-1}, e, w^-[I_{i-1}^-]) \neq \text{FAIL}\}$. Thus,

$$\begin{aligned} R_i &= R_i^- \backslash \{v\} \\ &= \{e \in R_{i-1}^- - e_{i-1}^- : \text{PROMOTE}(I_{i-1}^-, I'^-_{i-1}, e, w^-[I_{i-1}^-]) \neq \text{FAIL}\} \backslash \{v\} \\ &= \{e \in R_{i-1}^- \backslash \{v\} - e_{i-1}^- : \text{PROMOTE}(I_{i-1}^-, I'^-_{i-1}, e, w^-[I_{i-1}^-]) \neq \text{FAIL}\} \ . \end{aligned}$$

As we stated in the previous, we know that $w^-[I_{i-1}^-] = w[I_{i-1}]$ holds when $i - 1 < i^*$. Using this fact alongside $R_{i-1}^- = R_{i-1}$ by Fact A.11, $e_{i-1}^- = e_{i-1}$ by Fact A.7, $I_{i-1}^- = I_{i-1}$ by Fact A.9, and $I'^-_{i-1} = I'_{i-1}$ by Fact A.10, for each $i \in [1, i^*]$ we have

$$R_i = \{e \in R_{i-1} - e_{i-1} : \text{PROMOTE}(I_{i-1}, I'_{i-1}, e, w[I_{i-1}]) \neq \text{FAIL}\} \ .$$

Completing the proof. Above, we show that the starter, survivor, weight, and independent invariants hold for the first i^* levels. To complete the proof of this lemma, we show all level invariants hold. If $i^* \neq T + 1$, which means that we have invoked MATROIDCONSTRUCTLEVEL(i^*), our proof is complete by Theorem 3.1. Otherwise, if $i^* = T + 1$, we just need to show the terminator invariant holds to complete our proof, which means $R_{T+1} = \emptyset$. Note that in this case, $T = T^-$. Fact A.11 implies that $R_{T+1} = R_{T+1}^-$, and the assumption of lemma implies that $R_{T^-+1}^- = \emptyset$. Hence,

$$R_{T+1} = R_{T+1}^- = R_{T^-+1}^- = \emptyset \ ,$$

which completes the proof.

B Analysis of dynamic algorithm for maximum submodular under cardinality constraint

In this section, we prove the correctness of CONSTRUCTLEVEL, DELETE, and INSERT algorithms. We will also compute the query complexity of each one of them. Except for the approximation guarantee proof and some parts of the query complexity section, most of the theorems, lemmas, and their proof in this section are similar to Section 3 where we analyze our dynamic algorithm for matroid constraint. First, we define the following random variables.

Random Variables:

- We define the random variable $\mathbf{e}_i$ for the promoting element e_i that we pick at level L_i.
- We denote by $\mathbf{R}_i$ the random variable that corresponds to the set R_i of elements at the level L_i and its value is denoted by R_i which is the set of elements that are in the set R_i.
- We define the random variable $\mathbf{T}$ for the index of the last level that our algorithm creates. Indeed, for a level L_i to be created entirely, the value of the random variable $\mathbf{T}$ should be $\mathbf{T} \geq i$.
- We define $H_i = (e_1, \ldots, e_{i-1}, R_0, \ldots, R_i)$ as *the partial configuration up to level i*. Note that R_i is included in this definition, while e_i is not. We let $\mathbf{H}_i := (\mathbf{e}_1, \ldots, \mathbf{e}_{i-1}, \mathbf{R}_0, \mathbf{R}_1, \ldots, \mathbf{R}_i)$ be a random variable that corresponds to H_i.

We break the analysis of our algorithms into a few steps.

Step 1: Analysis of binary search. In the first step, we prove that the binary search that we use to speed up the process of finding the right levels for non-promoting elements works. Indeed, we prove that if $e \in V$ is a promoting element for a level L_{z-1}, it is promoting for all levels $L_{r \leq z-1}$ and if e is not promoting for the level L_z, it is not promoting for all levels $L_{r \geq z}$. Therefore, because of this monotonicity property, we can do a binary search to find the smallest $z \in [i, \ell - 1]$ so that e is promoting for the level L_{z-1}, but it is not promoting for the level L_z.

Step 2: Maintaining invariants. We define five invariants, and we show that these invariants *hold* when INIT is run, and our whole data structure gets built, and *are preserved* after every insertion and deletion of an element.

Invariants:

1. **Level invariants.**
   1.1 **Starter.** $R_0 = V$ and $I_0 = \emptyset$
   1.2 **Survivor.** For $1 \le i \le T+1$, $R_i = \{e \in R_{i-1} - e_{i-1} : \text{PROMOTE}(I_{i-1}, e) = True\}$
   1.3 **Cardinality.** For $1 \le i \le T$, $I_i = I_{i-1} + e_i$ where $\text{PROMOTE}(I_{i-1}, e) = True$
   1.4 **Terminator.** $R_{T+1} = \emptyset$
2. **Uniform invariant.** For all $i \ge 1$, condition on the random variables $\mathbf{H}_i$, the element e_i is chosen uniformly at random from the set R_i. That is, $\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \ge i \text{ and } \mathbf{H}_i = H_i] = \frac{1}{|R_i|} \cdot \mathbb{1}[e \in R_i]$.

The survivors invariant says that all elements that are added to R_i at a level L_i are promoting elements for that level. In another words, those elements of the set $R_{i-1} - e_{i-1}$ that are not promoting will be filtered out and not be seen in R_i. The terminator invariant shows that the recursive construction of levels stops when the survivor set becomes empty. The cardinality invariant shows that the sets I_i are constructed by adding a new element that promotes the previous set. That means the new element adds at least τ to the submodular value of I_{i-1}, and the size of I_i remains at most k. Intuitively, these level invariants provide us the approximation guarantee.

The uniform invariant asserts that for every level $L_{i \in [T]}$, condition on $\mathbf{T} \ge i$ and $\mathbf{H}_i = H_i$, the element e_i is chosen uniformly at random from the set R_i. That is, $\mathbb{P}[e_i = e | \mathbf{T} \ge i \text{ and } \mathbf{H}_i = H_i] = \frac{1}{|R_i|} \cdot \mathbb{1}[e \in R_i]$. Intuitively, this invariant provides us with the randomness that we need to fool the adversary in the (fully) dynamic model which in turn helps us to develop a dynamic algorithm for the submodular maximization under the cardinality constraint.

Step 3: Query complexity. In the third part of the proof, we bound the worst-case expected query complexity of the leveling algorithm, and later, we show that if the **uniform** invariant holds, we can bound the worst-case expected query complexity of the insertion and deletion operations.

Step 4: Approximation guarantee. Finally, in the last step of the proof, we show that if the **level** invariants is fulfilled, we can report a set I_T with $|I_T| \le k$ whose submodular value is an $(2 + \epsilon)$-approximation of the optimal value.

B.1 Monotone property of promotable levels and binary search argument Recall that we defined the function $\text{PROMOTE}(I_j, e)$ for an element $e \in V$ with respect to the level L_j which

- returns $True$ if $f(I_j + e) - f(I_j) \ge \tau$ and $|I_j| < k$;
- returns $False$ otherwise.

LEMMA B.1. *Let L_j be an arbitrary level of the Algorithm* CARDINALITYCONSTRAINTLEVELING*, where $1 \le j \le T$. Let $e \in V$ be an arbitrary element of the ground set. If* $\text{PROMOTE}(I_{j-1}, e)$ *returns False, then* $\text{PROMOTE}(I_j, e)$ *returns False.*

Proof. Recall that in Line 10, we set $I_j = I_{j-1} + e_j$. Suppose that $\text{PROMOTE}(I_{j-1}, e)$ returns $False$. It means that either $f(I_{j-1} + e) - f(I_{j-1}) \ge \tau$ or $|I_{j-1}| < k$ not hold. If $|I_{j-1}| >= k$, since $|I_j| = |I_{j-1}| + 1$, we can conclude that $|I_j| > k$ which means $\text{PROMOTE}(I_j, e) = False$. Moreover, if $f(I_{j-1} + e) - f(I_{j-1}) < \tau$, since $I_{j-1} \subset I_j$ and f is submodular, we have $f(I_j + e) - f(I_j) \le f(I_{j-1} + e) - f(I_{j-1}) < \tau$, which means that $\text{PROMOTE}(I_j, e) = False$. □

Using Lemma B.1, and by applying a simple induction, we can show the function $\text{PROMOTE}(I_j, e)$ is monotone. Thus, for every arbitrary element e, it is possible to perform a binary search on an interval $[i, \ell - 1]$ to find the smallest $z \in [i, \ell - 1]$ such that $\text{PROMOTE}(I_{z-1}, e) = True$ and $\text{PROMOTE}(I_z, e) = False$.

B.2 Correctness of invariants after CONSTRUCTLEVEL is called In this section, we show that the invariants that we defined above will hold at the end of the algorithm CONSTRUCTLEVEL(j). There are many similarities between this section and Section 3.2. However, in Section 3.2, where we prove the correctness of invariants after calling MATROIDCONSTRUCTLEVEL, we have independent and weight invariants instead of the cardinality invariant.

We first explain what we mean by stating that level invariants partially hold.

DEFINITION B.1. *For $j \ge 1$, we say that* the level invariants partially hold for the first j levels *if the followings hold.*

1. ***Starter.*** *$R_0 = V$ and $I_0 = \emptyset$*
2. ***Survivor.*** *For $1 \le i \le j$, $R_i = \{e \in R_{i-1} - e_{i-1} : \text{PROMOTE}(I_{i-1}, e) = True\}$*
3. ***Cardinality.*** *For $1 \le i \le j-1$, $I_i = I_{i-1} + e_i$ where $\text{PROMOTE}(I_{i-1}, e_i) = True$*

We want to prove the following theorem that says after invoking CONSTRUCTLEVEL(j), all level invariants hold. We break the proof of this theorem into four parts where we separately prove the survivor, cardinality, and terminator invariant hold in

Lemmas B.2, B.3, andB.4, respectively. Also, note that the starter invariant holds by the assumption of our theorem.

THEOREM B.1. *If before calling* CONSTRUCTLEVEL(j)*, the level invariants partially hold for the first j levels, then after the execution of* CONSTRUCTLEVEL(j)*, level invariants fully hold.*

LEMMA B.2. (SURVIVOR INVARIANT) *If before calling* CONSTRUCTLEVEL(j)*, the level invariants partially hold for the first j levels, then after its execution, the survivor invariant fully holds.*

Proof. First of all, we assume that $R_j \neq \emptyset$, otherwise $T = j - 1$ and we are done. As we have in Algorithm CONSTRUCTLEVEL, let P be a random permutation of the set R_j. Let us fix an arbitrary element $e \in P$ and suppose that at the time when we see $e \in P$, the current level is L_ℓ for $\ell \geq j$. We have two cases. Either e is a promoting element for the level $L_{\ell-1}$ or it is not promoting for the level $L_{\ell-1}$.

First, assume that e is a promoting element for the level $L_{\ell-1}$. We then let e_ℓ be e, perform a set of computations, and then start the new level. In particular, the element e is not added to $R_{\ell+1}$ and so, it will not appear in any set $R_{z>\ell}$. Recall that Lemma B.1 proves if e is not a promoting element with respect to a level L_z, it will not be a promoting element for the next level L_x where $z \leq x \leq T$. On the other hand, since e is a promoting element for the level $L_{\ell-1}$, we add e to all previous sets $R_{j+1}, \cdots, R_\ell$.

Next, we consider the latter case where e is not a promoting element for the level $L_{\ell-1}$. That is, PROMOTE($I_{\ell-1}, e$) is $False$. This essentially means that if we inductively apply the argument of Lemma B.1, there exists an integer $z \in [j, \ell)$ for which PROMOTE(I_{z-1}, e) is $True$, but PROMOTE(I_z, e) is $False$. This means e is a promoting element for all levels $L_j, \cdots, L_{z-1}$ and it is not promoting for levels $L_z, \cdots, L_T$. According to function CONSTRUCTLEVEL, we insert the element e into sets $R_{j+1}, \cdots, R_z$. Hence, after the execution of CONSTRUCTLEVEL(j), the survivors invariant holds. ☐

LEMMA B.3. (CARDINALITY INVARIANT) *If before calling* CONSTRUCTLEVEL(j)*, the level invariants partially hold for the first j levels, then after its execution, the cardinality invariant fully holds.*

Proof. In the execution of CONSTRUCTLEVEL(j), the variable ℓ is set to $j, j+1, \cdots, T, T+1$. Therefore, for each $\ell \in [j, T]$, we set $I_\ell = I_{\ell-1} + e_\ell$ in Line 10, where according to Line 9, PROMOTE($I_{\ell-1}, e_\ell$) = $True$. ☐

LEMMA B.4. (TERMINATOR INVARIANT) *If before calling* CONSTRUCTLEVEL(j)*, the level invariants partially hold for the first j levels, then after its execution, the terminator invariant fully holds.*

Proof. According to Line 15 and the variable z, if we add an element e to R_r at some point of time, then $r \leq z \leq \ell - 1$ holds at that moment. Since the variable ℓ never decreases during the execution of CONSTRUCTLEVEL(j) and we return $\ell - 1$ as T at the end, we can conduct that no element has been added to R_{T+1}, and then $R_{T+1} = \emptyset$, which means the terminator invariant holds. ☐

LEMMA B.5. (UNIFORM INVARIANT) *If* CONSTRUCTLEVEL(j) *is invoked and the level invariants are going to hold after its execution, then for any $j \geq i$ we have $\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i$ and $\mathbf{H}_i = H_i] = \frac{1}{|R_i|} \cdot \mathbb{1}[e \in R_i]$.*

Proof. In the beginning of CONSTRUCTLEVEL(j), we take a random permutation of elements in R_j. Making a random permutation is equal to sampling all elements without replacement. In other words, instead of fixing a random permutation P of R_j and iterating through P in Line 8, we repeatedly sample a random element e from the unseen elements of R_j until we have seen all of the elements. Thus, for proving this lemma, we are assuming that our algorithm uses sampling without replacement.

Given this view, we make the following claims.

Observation 1. e_i is the first element of R_i seen in the permutation.

This is because before e_i is seen, the value of ℓ is at most i. It is also clear from the algorithm that when an element e is considered, it can only be added to sets R_x for $x \leq \ell$, both when PROMOTE($I_{\ell-1}, e$) = $Ture$ and when PROMOTE($I_{\ell-1}, e$) = $False$. Furthermore, e can only be added to R_ℓ if $e = e_\ell$. Therefore, no element can be added to R_i before e_i is seen.

Observation 2. Once $e_1, \ldots, e_{i-1}$ have been seen, the set R_i is uniquely determined.

Note that R_i is uniquely determined *even though the algorithm has not observed its elements yet*. This is because regardless of the randomness of CONSTRUCTLEVEL(j), the level invariants will hold after its execution. This implies that the content of the set R_i only depends on the value of $(\mathbf{e}_1, \ldots, \mathbf{e}_{i-1})$, which is not going to change after it is set to be equal to $(e_1, \ldots, e_{i-1})$.

Let the random variable $\mathbf{M}_i$ denote the sequence of elements that our algorithm observes until setting $\mathbf{e}_{i-1}$ to be e_{i-1}, including e_{i-1} itself. In other words, if e_{i-1} is the x-th element of the permutation P, M_i is the first x elements of P.

Based on the above facts, conditioned on $\mathbf{M}_i = M_i$, **(a)** the value of $\mathbf{R}_i$, or in other words R_i is uniquely determined. **(b)** e_i is going to be the first element of R_i that the algorithm observes. Therefore, since we assumed that the algorithm uses sampling without replacement, $\mathbf{e}_i$ is going to have a uniform distribution over R_i, i.e.,

$$\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{M}_i = M_i] = \frac{1}{|R_i|} \mathbb{1}[e \in R_i] \quad .$$

By the law of total probability, we have

$$\mathbb{P}[\mathbf{e}_i = e_i | \mathbf{T} \geq i, \mathbf{H}_i = H_i] = \mathbb{E}_{M_i}[\mathbb{P}[\mathbf{e}_i = e_i | \mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{M}_i = M_i]] \quad ,$$

where the expectation is taken over all M_i with positive probability.

Also, note that knowing that $\mathbf{M}_i = M_i$ uniquely determines the value of $\mathbf{H}_i$ as well. This is because M_i includes $(e_1, \ldots, e_{i-1})$ and, with similar reasoning to what we used for Observation 2, we can say that $R_1, \ldots, R_i$ are uniquely determined by $(e_1, \ldots, e_{i-1})$.

Since we are only considering M_i with positive probability, and $\mathbf{H}_i$ is a function of $\mathbf{M}_i$ given the discussion above, all the forms of M_i that we consider in our expectation are the ones that imply $\mathbf{H}_i = H_i$. Therefore, we can drop the condition $\mathbf{H}_i = H_i$ from the condition $\mathbf{H}_i = H_i, \mathbf{M}_i = M_i$, which implies

$$\mathbb{P}[\mathbf{e}_i = e_i | \mathbf{T} \geq i, \mathbf{H}_i = H_i] = \mathbb{E}_{M_i}[\mathbb{P}[\mathbf{e}_i = e_i | \mathbf{T} \geq i, \mathbf{M}_i = M_i]] = \mathbb{E}_{M_i}\left[\frac{1}{|R_i|}\mathbb{1}[e_i \in R_i]\right] = \frac{1}{|R_i|}\mathbb{1}[e_i \in R_i] \quad ,$$

as claimed. □

B.3 Correctness of invariants after an update In our dynamic model, we consider a sequence $\mathcal{S}$ of updates of elements of an underlying ground set V where at time t of the sequence $\mathcal{S}$, we observe an update which can be the deletion of an element $e \in V$ or insertion of an element $e \in V$. We assume that an element e can be deleted at time t, if after the first time e is inserted, it is not deleted until time t. In this section, similar to Section 3.3, we prove that all invariants hold after each update. While most of this section is identical to Section 3.3, we have the cardinality invariant instead of independent and weight invariants in this section.

We use several random variables for our analysis, including $\mathbf{e}_i$, $\mathbf{R}_i$, $\mathbf{T}$, and $\mathbf{H}_i$. Upon observing an update at time t, we should distinguish between each of these random variables and their corresponding values before and after the update. To do so, we use the notations $\mathbf{Y}^-$ and Y^- to denote a random variable and its value before time t when e is either deleted or inserted, and we keep using $\mathbf{Y}$ and Y to denote them at the current time after the execution of update. As an example, $\mathbf{H}_i^- := (\mathbf{e}_1^-, \ldots, \mathbf{e}_{i-1}^-, \mathbf{R}_0^-, \mathbf{R}_1^-, \ldots, \mathbf{R}_i^-)$ is the random variable that corresponds to the partial configuration $H_i^- = (e_1^-, \ldots, e_{i-1}^-, R_0^-, \ldots, R_i^-)$.

B.3.1 Correctness of invariants after every insertion We first consider the case when the update at time t of the sequence $\mathcal{S}$ is an insertion of an element v. In this section, we prove the following theorem.

Theorem B.2. *If before the insertion of an element v, the level invariants and uniform invariant hold, then they also hold after the execution of* Insert(v).

We break the proof of this theorem into Lemmas B.6 and B.7.

Lemma B.6. (Level invariants) *If before the insertion of an element v the level invariants (i.e., starter, survivor, cardinality, and terminator) hold, then they also hold after the execution of* Insert(v).

Proof. To prove the lemma, we first mention some useful facts and then show that the starter, cardinality, and survivor invariants partially hold. Finally, we prove that all level invariants hold.

We begin with defining variables i^* and j^* as follows.

- i^***:** If during the execution of Insert(v) there is $i \in [T]$ such that e_i has been set to be v, which also implies that we have invoked ConstructLevel($i + 1$), then we set i^* to be i. Otherwise, we set i^* to be $T + 1$.
- j^***:** Let j^* be the largest $i \in [0, T^- + 1]$ such that we have added v to R_i^-.

We consider these two cases in this proof.

- Case 1: $i^* \leq T$, which means $e_{i^*} = v$ and therefore $j^* = i^*$. It also means that we have invoked ConstructLevel($i^* + 1$).
- Case 2: $i^* = T + 1$, which means ConstructLevel has never been invoked during the insertion of v. Note that in this case, $T = T^-$ and therefore, $j^* < T^- + 1 = T + 1 = i^*$.

Considering our algorithm in Insert(v), it is clear that for any $i < i^*$, we have not made any kind of change in e_i^- or I_i^- at least until ConstructLevel is invoked, if it ever gets invoked. Additionally, according to ConstructLevel, we know that if we have invoked ConstructLevel($i^* + 1$), there has not been any alteration to the variables regarding previous levels. Hence, we can conduct the following facts.

Fact B.1. *For any $i \in [1, i^*)$, we have $e_i = e_i^-$.*

Fact B.2. *For any $i \in [0, i^*)$, we have $I_i = I_i^-$.*

By the definition of j^*, we have added the element v to the set R_i^-, for each $i \in [0, j^*]$. Recall that $j^* \leq i^*$, and by invoking constLevel($i^* + 1$), there has not been any alteration to the variables regarding previous levels. It leads to the following fact.

Fact B.3. *For any* $i \in [0, j^*]$*, we have* $R_i = R_i^- + v$.

We know that if Case 2 holds, which means ConstructLevel has never been invoked during the insertion of v, we have $R_i = R_i^-$ for any $i \in [j^* + 1, T + 1]$. Recall that in Case 2, $i^* = T + 1$, and therefore $[j^* + 1, T + 1] = [j^* + 1, i^*]$. Also if Case 1 holds, $j^* = i^*$, so $[j^* + 1, i^*] = \emptyset$. Thus, independent of the case, we can have the following fact.

Fact B.4. *For any* $i \in [j^* + 1, i^*]$*, we have* $R_i = R_i^-$.

In the following, we first prove that the starter invariant holds after executing Insert(v). We next show that the cardinality and survivor invariants partially hold for the first $i^* + 1$ levels. Finally, we complete the proof by proving that all the level invariants hold.

Starter invariant. To show that the starter invariant holds after Insert(v), we need to prove $R_0 = V$ and $I_0 = \emptyset$. By the assumption of this lemma, we have $R_0^- = V^-$, and Fact B.3 results that $R_0 = R_0^- + v$. Thus $R_0 = R_0^- + v = V^- + v = V$.

Again by the assumption of this lemma, $I_0^- = \emptyset$. Due to Fact B.2, we have $I_0 = I_0^-$, and therefore, it is clear that $I_0 = I_0^- = \emptyset$.

Cardinality invariant (partially). Now we show that the cardinality invariant partially holds up to the level L_{i^*} after Insert(v). To do this, we prove $I_i = I_{i-1} + e_i$ and Promote$(I_{i-1}, e_i) = True$ holds for all $i \in [1, i^*)$.

Using Fact B.2, we have $I_i = I_i^-$ for any $i \in [1, i^*)$. Also, $I_i^- = I_{i-1}^- + e_i^-$ by the assumption of this lemma. Then,

$$I_i = I_i^- = I_{i-1}^- + e_i^- \ .$$

For any $i \in [1, i^*)$, we have $I_{i-1} = I_{i-1}^-$ by Fact B.2 and $e_i = e_i^-$ by Fact B.1. Then we can further conclude that:

$$I_i = I_{i-1} + e_i \ .$$

Finally, we prove Promote$(I_{i-1}, e_i) = True$ for any $i \in [1, i^*)$. By the assumption of this lemma, we know that Promote$(I_{i-1}^-, e_i^-) = True$. In addition, for any $i \in [1, i^*)$, we have $I_{i-1} = I_{i-1}^-$ by Fact B.2 and $e_i = e_i^-$ by Fact B.1. Thus, Promote$(I_{i-1}, e_i) = True$ for any $i \in [1, i^*)$.

Survivor invariant (partially). Next, we show that the survivor invariant partially holds for the first i^* levels by proving that $R_i = \{e \in R_{i-1} - e_{i-1} : \text{Promote}(I_{i-1}, e) = True\}$ holds for any $i \in [1, i^*]$. In the following, we first consider $i \in [1, j^*]$ and then $i \in [j^* + 1, i^*]$. Recall that j^* is the largest j that we added v to R_j in Insert(v).

First we study $i \in [1, j^*]$. Using Fact B.3, $R_i = R_i^- + v$ holds for each $i \in [1, j^*]$, and $R_i^- = \{e \in R_{i-1}^- - e_{i-1}^- : \text{Promote}(I_{i-1}^-, e) = True\}$ according to the assumption of this lemma. Besides, by the definition of j^* and according to the break condition in Line 13, we can conduct that Promote$(I_{i-1}^-, v) = True$. Putting everything together we have,

$$\begin{aligned} R_i &= R_i^- + v \\ &= \{e \in R_{i-1}^- - e_{i-1}^- : \text{Promote}(I_{i-1}^-, e) = True\} + v \\ &= \{e \in R_{i-1}^- + v - e_{i-1}^- : \text{Promote}(I_{i-1}^-, e) = True\} \ . \end{aligned}$$

Using $R_{i-1} = R_{i-1}^- + v$ by Fact B.3, $e_{i-1} = e_{i-1}^-$ by Fact B.1, and $I_{i-1} = I_{i-1}^-$ by Fact B.2, we have:

$$R_i = \{e \in R_{i-1} - e_{i-1} : \text{Promote}(I_{i-1}, e) = True\} \ .$$

Recall that if case 1 holds, $i^* = j^*$, and then the survivor invariant partially holds for the first i^* levels. Otherwise, if Case 2 holds, it remains to study $i \in [j^* + 1, i^*] = [j^* + 1, T + 1]$.

It holds that $R_i = R_i^-$ for any $i \in [j^*, i^*]$ according to Fact B.4. Adding it to the assumption of this lemma, we have:

$$R_i = R_i^- = \{e \in R_{i-1}^- - e_{i-1}^- : \text{Promote}(I_{i-1}^-, e) = True\} \ .$$

Besides, $e_{i-1} = e_{i-1}^-$ by Fact B.1 and $I_{i-1} = I_{i-1}^-$ by Fact B.2. Thus,

$$R_i = \{e \in R_{i-1}^- - e_{i-1} : \text{Promote}(I_{i-1}, e) = True\} \ .$$

We have either $i = j^* + 1$ or $i \in (j^* + 1, i^*]$. If $i \in (j^* + 1, i^*]$, then $R_{i-1} = R_{i-1}^-$ by Fact B.4. Hence,

$$R_i = \{e \in R_{i-1} - e_{i-1} : \text{Promote}(I_{i-1}, e) = True\} \ .$$

Now consider $i = j^* + 1$. According to Fact B.3, $R_{j^*} = R_{j^*}^- + v$, and then $R_{j^*}^- = R_{j^*} \setminus \{v\}$. Due to the definition of j^*, we know v is not added to R_{j^*+1}. Hence, according to the break condition in Line 13, we can conduct that Promote$(I_{j^*}, e) = False$.

Putting everything we have

$$\begin{aligned} R_{j^*+1} &= \{e \in R^-_{j^*} - e_{j^*} : \textsc{Promote}(I_{j^*}, e) = True\} \\ &= \{e \in R_{j^*} \backslash \{v\} - e_{j^*} : \textsc{Promote}(I_{j^*}, e) = True\} \\ &= \{e \in R_{j^*} - e_{j^*} : \textsc{Promote}(I_{j^*}, e) = True\} \ , \end{aligned}$$

which finishes the proof.

Completing the proof. Now having everything in hand, we can complete the proof of this lemma. Above, we show that the starter, survivor, and cardinality invariants partially hold for the first i^* levels.

If Case 1 holds, it means we set $e_{i^*} = v$ and $I_{i^*} = I_{i^*-1} + e_{i^*}$ in Line 18. Then in the Line 19, we set $R_{i^*+1} = \{e' \in R_{i^*} : \textsc{Promote}(I_{i^*}, e') = True\}$. It means that the invariants partially hold for the first $i^* + 1$ levels. Next, we invoke $\textsc{ConstructLevel}(i^* + 1)$, and then all the invariants hold by Theorem B.1.

Otherwise, if Case 2 holds, $i^* = T + 1 = T^- + 1$. It means the starter, survivor, and cardinality invariants hold and it remains to show the terminator invariant to complete the proof. Recall that in this case, $j^* < T^- + 1$, which implies $(T^- + 1) \in [j^* + 1, i^*]$ and then $R_{T^-+1} = R^-_{T^-+1}$ by Fact B.3. Also, the assumption of this lemma implies $R^-_{T^-+1} = \emptyset$. Therefore, $R_{T+1} = R_{T^-+1} = R^-_{T^-+1} = \emptyset$, which means the terminator invariant holds and completes the proof.
□

Lemma B.7. (Uniform invariant) *If before the insertion of an element v the level and uniform invariants hold, then the uniform invariant also holds after the execution of* $\textsc{Insert}(v)$.

Proof. By the assumption that the uniform invariant holds before the insertion of the element v, we mean that for any arbitrary i and any arbitrary element e, the following holds:

$$\mathbb{P}\left[\mathbf{e}^-_i = e | \mathbf{T}^- \geq i, \mathbf{H}^-_i = H^-_i\right] = \frac{1}{|R^-_i|} \cdot \mathbb{1}\left[e \in R^-_i\right] \ .$$

We aim to prove that given our assumptions, after the execution of $\textsc{Insert}(v)$, for each arbitrary i and each arbitrary element e, we have

$$\mathbb{P}\left[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i\right] = \frac{1}{|R_i|} \cdot \mathbb{1}\left[e \in R_i\right] \ .$$

Note that $\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i]$, is only defined when $\mathbb{P}[\mathbf{T} \geq i, \mathbf{H}_i = H_i] > 0$, which means that given the input and considering the behavior of our algorithm including its random choices, it is possible to reach a state where $\mathbf{T} \geq i$ and $\mathbf{H}_i = H_i$. In this proof, we use $\mathbf{p}_i$ to denote the variable p_i used in the $\textsc{Insert}$ as a random variable.

Fix any arbitrary i and any arbitrary element e. Since $\mathbf{H}^-_i = (\mathbf{e}^-_1, \dots, \mathbf{e}^-_{i-1}, \mathbf{R}^-_0, \mathbf{R}^-_1, \dots, \mathbf{R}^-_i)$ refers to our data structure levels before the insertion of the element v, it is clear that the following facts hold about $\mathbf{H}^-_i$.

Fact B.5. *For any $j < i$, $\mathbf{e}^-_j \neq v$.*

Fact B.6. *For any $j \leq i$, $v \notin \mathbf{R}^-_j$.*

We consider the following cases based on which of the following holds for $H_i = (e_1, \dots, e_{i-1}, R_0, R_1, \dots, R_i)$:

- Case 1: If the $e_j = v$ for some $j < i$.
- Case 2: If $v \notin \{e_1, \dots, e_{i-1}\}$.

By Claims B.1 and B.2, we prove that no matter the case, $\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i]$ is equal to $\frac{1}{|R_i|} \cdot \mathbb{1}[e \in R_i]$, which completes the proof of the Lemma.

Claim B.1. *If H_i is such that there is a $1 \leq j < i$ that $e_j = v$, then $\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i] = \frac{1}{|R_i|} \cdot \mathbb{1}[e \in R_i]$.*

Proof. We know that, $\mathbf{p_j}$ must have been equal to 1, as otherwise, instead of having $\mathbf{e_j} = e_j = v$, we would have had $\mathbf{e}_j = \mathbf{e}^-_j$, which would not have been equal to v as stated in Fact B.5. According to our algorithm, since $\mathbf{p_j}$ has been equal to 1, we have invoked $\textsc{ConstructLevel}(j + 1)$. By Lemma B.6, we know that the level invariants hold at the end of the execution of $\textsc{Insert}$, which is also the end of the execution of $\textsc{ConstructLevel}(j + 1)$. Thus, Lemma B.5, proves that $\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i] = \frac{1}{|R_i|} \cdot \mathbb{1}[e \in R_i]$. □

Claim B.2. *If H_i is such that $e_j \neq v$ for any $1 \leq j < i$, then $\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i] = \frac{1}{|R_i|} \cdot \mathbb{1}[e \in R_i]$.*

Proof. We first define H^-_i based on H_i as $H^-_i := (R_0 \backslash \{v\}, \dots, R_i \backslash \{v\}, e_1, \dots, e_{i-1})$ and prove the following claim.

Claim B.3. *The events $[\mathbf{T} \geq i, \mathbf{H}_i = H_i]$ and $[\mathbf{T}^- \geq i, \mathbf{H}^-_i = H^-_i, \mathbf{p_1} = 0, \dots, \mathbf{p_{i-1}} = 0]$ are equivalent and imply each other, thusly they are interchangeable.*

Proof. First, we show that if $\mathbf{T} \geq i, \mathbf{H}_i = H_i$, then $\mathbf{T}^- \geq i, \mathbf{H}^-_i = H^-_i, \mathbf{p_1} = 0, \dots, \mathbf{p_{i-1}} = 0$. Considering that case 2 holds for H_i, $\mathbf{H}_i = H_i$, means that for any $j < i$, $\mathbf{e}_i \neq v$, which means there is no $j < i$ with $\mathbf{p_j} = 1$. Note that if $\mathbf{p_j} = 1$, then we would have set $\mathbf{e_j}$ to be equal to v, and we would have invoked $\textsc{ConstructLevel}(j + 1)$. Thus, in addition to knowing that for any

$j < i$, $\mathbf{p_j} = 0$, we also know that, we have not invoked ConstructLevel$(j+1)$ for any $j < i$. As for any $j < i$, $\mathbf{p_j} = 0$ and ConstructLevel$(j+1)$ was not invoked, we have the following results:

1. Level i also existed before the insertion of v, i.e. $\mathbf{T}^- \geq i$.
2. We have made no change in the values of $(\mathbf{e_1}, \ldots, \mathbf{e_{i-1}})$, and they still have the values they had before the insertion of v, i.e. for any $j < i$, $\mathbf{e_j} = \mathbf{e_j^-}$, and so $\mathbf{e_j^-} = e_j$.
3. All the change we might have made in our data structure is limited to adding the element v to a subset of $\{\mathbf{R_0^-}, \ldots, \mathbf{R_i^-}\}$. Hence, for any $j \leq i$, whether $\mathbf{R_j}$ is equal to $\mathbf{R_j^-}$ or $\mathbf{R_j^-} \cup \{v\}$, $\mathbf{R_j^-} = \mathbf{R_j} \backslash \{v\} = R_j \backslash \{v\}$.

So far, we have proved that throughout our algorithm, we reach the state, where $\mathbf{T} \geq i, \mathbf{H}_i = H_i$, only if $\mathbf{T}^- \geq i, \mathbf{H}_i^- = H_i^-, \mathbf{p_1} = 0, \ldots, \mathbf{p_{i-1}} = 0$.

We know that in our insertion algorithm, there is not any randomness other than setting the value of $\mathbf{p_j}$ as long as we have not invoked ConstructLevel, which only happens when for a j, $\mathbf{p_j}$ is set to be 1. It means that the value of $\mathbf{H}_i$ can be determined uniquely if we know the value of $\mathbf{H}_i^-$, and we know that $\mathbf{p_1}, \ldots, \mathbf{p_{i-1}}$ are all equal to 0. Since we have assumed that $\mathbf{T} \geq i, \mathbf{H}_i = H_i$ is a valid and reachable state in our algorithm, $\mathbf{T}^- \geq i, \mathbf{H}_i^- = H_i^-$ must have been a reachable state as well. Plus, $\mathbf{T}^- \geq i, \mathbf{H}_i^- = H_i^-, \mathbf{p_1} = 0, \ldots, \mathbf{p_{i-1}} = 0$, should imply that $\mathbf{T} \geq i$ and $\mathbf{H}_i = H_i$. Otherwise, $\mathbf{T} \geq i, \mathbf{H}_i = H_i$ could not be a reachable state, which is in contradiction with our assumption. □

Now, we proceed to calculate $\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i]$. As stated above, considering that Case 2 holds for H_i, we know that $\mathbf{T} \geq i, \mathbf{H}_i = H_i$ implies that ConstructLevel has not been invoked for any $j < i$. Thus, the value of $\mathbf{e}_i$ will be determined based on the random variable $\mathbf{p_i}$. And we have:

$$\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i] = \sum_{p_i \in \{0,1\}} \left(\mathbb{P}[\mathbf{p_i} = p_i | \mathbf{T} \geq i, \mathbf{H}_i = H_i] \cdot \mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{p_i} = p_i]\right)$$

According to the algorithm, if $v \in H_i$, then $\mathbb{P}[\mathbf{p_i} = 1 | \mathbf{T} \geq i, \mathbf{H}_i = H_i]$ is equal to $\frac{1}{|R_i|}$. Otherwise, if $v \notin H_i$, then $\mathbf{p_i}$ would be zero by default, and $\mathbb{P}[\mathbf{p_i} = 1 | \mathbf{T} \geq i, \mathbf{H}_i = H_i] = 0$. Hence, we can say that:

$$\mathbb{P}[\mathbf{p_i} = 1 | \mathbf{T} \geq i, \mathbf{H}_i = H_i] = \frac{1}{|R_i|} \cdot \mathbb{1}[v \in R_i].$$

Additionally, Having $\mathbf{T} \geq i, \mathbf{H}_i = H_i$, if $\mathbf{p_i} = 1$, then $\mathbf{e}_i$ would be v. Otherwise, if $\mathbf{p_i} = 0$, then $\mathbf{e}_i^-$ would remain unchanged, i.e. $\mathbf{e}_i = \mathbf{e}_i^-$. Hence, $\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i]$ is equal to

$$\frac{1}{|R_i|} \cdot \mathbb{1}[v \in R_i] \cdot \mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{p_i} = 1] + (1 - \frac{1}{|R_i|} \cdot \mathbb{1}[v \in R_i]) \cdot \mathbb{P}[\mathbf{e}_i^- = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{p_i} = 0].$$

We consider the following cases based on the value of e:

- Case (i): $e = v$.
  In this case $\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{p_i} = 1] = 1$, and $\mathbb{P}\left[\mathbf{e}_i^- = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{p_i} = 0\right] = 0$. Thus, we have:

$$\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i] = \frac{1}{|R_i|} \cdot \mathbb{1}[v \in R_i] \cdot 1 + (1 - \frac{1}{|R_i|} \cdot \mathbb{1}[v \in R_i]) \cdot 0 = \frac{1}{|R_i|} \cdot \mathbb{1}[v \in R_i] = \frac{1}{|R_i|} \cdot \mathbb{1}[e \in R_i].$$

- Case (ii): $e \neq v$ In this case, $\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{p_i} = 1] = 0$. So we have:

$$\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i] = \frac{1}{|R_i|} \cdot \mathbb{1}[v \in R_i] \cdot 0 + (1 - \frac{1}{|R_i|} \cdot \mathbb{1}[v \in R_i]) \cdot \mathbb{P}[\mathbf{e}_i^- = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{p_i} = 0].$$

  According to the claim that we proved beforehand, $\mathbf{T} \geq i, \mathbf{H}_i = H_i$ and $\mathbf{T}^- \geq i, \mathbf{H}_i^- = H_i^-, \mathbf{p_1} = 0, \ldots, \mathbf{p_{i-1}} = 0$ are interchangeable. So we have:

$$\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i] = (1 - \frac{1}{|R_i|} \cdot \mathbb{1}[v \in R_i]) \cdot \mathbb{P}[\mathbf{e}_i^- = e | \mathbf{T}^- \geq i, \mathbf{H}_i^- = H_i^-, \mathbf{p_1} = 0, \ldots, \mathbf{p_i} = 0].$$

  Since for any $j \leq i$, $\mathbf{e}_i^-$ and $\mathbf{p_i}$ are independent random variables, we have:

$$\begin{aligned}\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i] &= (1 - \frac{1}{|R_i|} \cdot \mathbb{1}[v \in R_i]) \cdot \mathbb{P}[\mathbf{e}_i^- = e | \mathbf{T}^- \geq i, \mathbf{H}_i^- = H_i^-] \\ &= (1 - \frac{1}{|R_i|} \cdot \mathbb{1}[v \in R_i]) \cdot \left(\frac{1}{|R_i^-|} \cdot \mathbb{1}[e \in R_i^-]\right),\end{aligned}$$

where the last equality holds because of the assumption stated in Lemma. From the definition of H_i^-, we have $R_i^- = R_i \backslash \{v\}$. Therefore,

$$\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i] = \frac{|R_i| - \mathbb{1}[v \in R_i]}{|R_i|} \cdot \left(\frac{1}{|R_i| - \mathbb{1}[v \in R_i]} \cdot \mathbb{1}[e \in R_i \backslash \{v\}] \right) .$$

And since, $e \neq v$, we have:

$$\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i] = \frac{1}{|R_i|} \cdot \mathbb{1}[e \in R_i] .$$

□

As stated before, proof of these claims completes the Lemma's proof. □

B.3.2 Correctness of invariants after every deletion Now, we consider the case when the update at time t of the sequence $\mathcal{S}$, is a deletion of an element v, and prove the following theorem.

Theorem B.3. *If before the deletion of an element v, the level invariants and the uniform invariant hold, then they also hold after the execution of* Delete(v).

Similar to Theorem B.2, we break the proof of this theorem into Lemmas B.8 and B.9.

Lemma B.8. (Level invariants) *If before the deletion of an element v the level invariants (i.e., starter, survivor, cardinality, and terminator) hold, then they also hold after the execution of* Delete(v).

Proof. Let i^* be the level for which ConstructLevel is invoked, and if ConstructLevel is never invoked during the execution of Delete(v), we set i^* to be $T + 1$. Considering our algorithm in Delete(v), we know that in levels before i^*, or in other words in each level $i \in [1, i^*)$ we do not make any change in our data structure other than removing the element v from R_i^- if it has this element in it. Hence, we have the following facts about e and I, which are similar to the Facts B.1 and B.2 in the proof of Lemma B.6.

Fact B.7. *For any $i \in [1, i^*)$, it holds that $e_i = e_i^-$.*

Fact B.8. *For any $i \in [0, i^*)$, it holds that $I_i = I_i^-$.*

In Delete(v), we removes the element v from R_i^- for any $i \in [0, \min(i^*, T^-)]$. By the definition of i^*, we have $i^* \leq T^- + 1$, and $i^* = T^- + 1$ happens only if ConstructLevel has never been invoked during the execution of Delete(v). In this case, $R_{T^-+1} = R^-_{T^-+1}$, and since $R^-_{T^-+1} = \emptyset$ according to the assumption of this lemma, we have $R_{T^-+1} = R^-_{T^-+1} = \emptyset$. Therefore, $R_i = R_i^- \backslash \{v\}$ also holds when $i = T^- + 1$, and as $\min(i^*, T^- + 1) = i^*$, we can conduct the following fact.

Fact B.9. *For any $i \in [0, i^*]$, it holds that $R_i = R_i^- \backslash \{v\}$.*

Starter invariant. Similar to Lemma B.6, we prove the starter invariant holds, which means $R_0 = V$ and $I_0 = \emptyset$.

Fact B.9 implies that $R_0 = R_0^- \backslash \{v\}$. Besides, $R_0^- = V^-$ since the starter invariant holds before the deletion of v by the assumption. We also know $V = V^- \backslash \{v\}$. Therefore, $R_0 = R_0^- \backslash \{v\} = V^- \backslash \{v\} = V$.

We have $I_0 = I_0^-$ by Fact B.8. Adding it to $I_0^- = \emptyset$, which is an assumption of this lemma, implies that $I_0 = \emptyset$.

Cardinality invariant (partially). We show that the cardinality invariant partially holds for the first i^* levels, which means $I_i = I_{i-1} + e_i$ and Promote(I_{i-1}, e_i) hold for any $i \in [1, i^*)$.

Same as Lemma B.6, we first prove $I_i = I_{i-1} + e_i$ holds for any $i \in [1, i^*)$. By the assumption of this lemma, $I_i^- = I_{i-1}^- + e_i^-$ holds. Besides, $I_i = I_i^-$ for any any $i \in [1, i^*)$ according to Fact B.8. Therefore,

$$I_i = I_i^- = I_{i-1}^- + e_i^- .$$

Adding it to $I_{i-1}^- = I_{i-1}$ by Fact B.8 and $e_i^- = e_i$ by Fact B.7, for any $i \in [1, i^*)$ we have

$$I_i = I_{i-1} + e_i .$$

Next, we show Promote(I_{i-1}, e_i) $= True$ for any $i \in [1, i^*)$. By the assumption of this lemma, we know that Promote(I_{i-1}^-, e_i^-) $= True$. In addition, for any $i \in [1, i^*)$, we have $I_{i-1} = I_{i-1}^-$ by Fact B.8 and $e_i = e_i^-$ by Fact B.7. Thus, Promote(I_{i-1}, e_i) $= True$ for any $i \in [1, i^*)$.

Survivor invariant (partially). We next show that the survivor invariant partially holds for the first i^* levels. To do this, we prove $R_i = \{e \in R_{i-1} - e_{i-1} : \text{Promote}(I_{i-1}, e) = True\}$ holds for any $i \in [1, i^*]$.

Using Fact B.9, we have $R_i = R_i^- \backslash \{v\}$ for each $i \in [1, i^*]$. Also, the assumption of this lemma implies that

$R_i^- = \{e \in R_{i-1}^- - e_{i-1}^- : \textsc{Promote}(I_{i-1}^-, e) = True\}$. Thus,

$$\begin{aligned} R_i &= R_i^- \backslash \{v\} \\ &= \{e \in R_{i-1}^- - e_{i-1}^- : \textsc{Promote}(I_{i-1}^-, e) = True\} \backslash \{v\} \\ &= \{e \in R_{i-1}^- \backslash \{v\} - e_{i-1}^- : \textsc{Promote}(I_{i-1}^-, e) = True\} \ . \end{aligned}$$

Using $R_{i-1}^- = R_{i-1}$ by Fact B.9, $e_{i-1}^- = e_{i-1}$ by Fact B.7, and $I_{i-1}^- = I_{i-1}$ by Fact B.8, for each $i \in [1, i^*]$ we have

$$R_i = \{e \in R_{i-1} - e_{i-1} : \textsc{Promote}(I_{i-1}, e) = True\} \ .$$

Completing the proof. Above, we show that the starter, survivor, and cardinality invariants hold for the first i^* levels. To complete the proof of this lemma, we show all level invariants hold. If $i^* \neq T + 1$, which means that we have invoked $\textsc{ConstructLevel}(i^*)$, our proof is complete by Theorem B.1. Otherwise, if $i^* = T + 1$, we just need to show the terminator invariant holds to complete our proof, which means $R_{T+1} = \emptyset$. Note that in this case, $T = T^-$. Fact B.9 implies that $R_{T+1} = R_{T+1}^-$, and the assumption of lemma implies that $R_{T^-+1}^- = \emptyset$. Hence,

$$R_{T+1} = R_{T+1}^- = R_{T^-+1}^- = \emptyset \ ,$$

which completes the proof. ∎

Lemma B.9. (Uniform invariant) *If before the deletion of an element v, the level and uniform invariants hold, then the uniform invariant also holds after the execution of* $\textsc{Delete}(v)$.

Proof. In other words, we want to prove that if for any i and any element e

$$\mathbb{P}[\mathbf{e}_i^- = e | \mathbf{T}^- \geq i, \mathbf{H}_i^- = H_i^-] = \frac{1}{|R_i^-|} \cdot \mathbb{1}[e \in R_i^-] \ ,$$

then, after execution $\textsc{Delete}(v)$, for each i and each element e, we have

$$\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i] = \frac{1}{|R_i|} \cdot \mathbb{1}[e \in R_i] \ .$$

Fix any arbitrary i and e. We define a random variable $\mathbf{X_i}$ attaining values from the set $\{0, 1, 2\}$, as follows:

1. If the execution of $\textsc{Delete}(v)$ has terminated after invoking $\textsc{ConstructLevel}(j)$, then we set $\mathbf{X_i}$ to 2.
2. If the execution of $\textsc{Delete}(v)$ has terminated in a level $L_{j \leq i}$ because $v \notin R_j^-$, then we set $\mathbf{X_i}$ to 1.
3. Otherwise, we set $\mathbf{X_i}$ to 0. That is, this case occurs if $v \in R_i^-$ and $\textsc{Delete}(v)$ terminates because in a level $L_{j>i}$, either $e_j = v$ or $v \notin R_j$.

In Claims B.4, B.7, and B.8, we show that for each value $X_i \in \{0, 1, 2\}$, $\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{X_i} = X_i] = \frac{1}{|R_i|} \cdot \mathbb{1}[e \in R_i]$. This would imply the statement of our Lemma and completes the proof since

$$\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i] = \mathbb{E}_{X_i \sim \mathbf{X_i}}[\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{X_i} = X_i]]$$

by the law of total probability.

Claim B.4. $\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{X_i} = 0] = \frac{1}{|R_i|} \cdot \mathbb{1}[e \in R_i]$.

Proof. First, we prove the following claim.

Claim B.5. *If* $\mathbf{X_i} = 0$*, then for every $j < i$, $e_j \neq v$ and $v \notin R_i$.*

Proof. Since $\mathbf{X_i} = 0$, then $\textsc{ConstructLevel}(j)$ has not been invoked for any $j \leq i$. Thus, $\mathbf{e_j^-} = \mathbf{e_j} = e_j$ for any $j < i$. However, if $e_j = v$ for a level index $j < i$, then $\mathbf{e_j^-} = v$ would have held for that $j < i$, which means that $\textsc{ConstructLevel}(j)$ would have been executed for that j. This contradicts the assumption that $\mathbf{X_i} = 0$. Therefore, for all $j < i$, we must have $e_j \neq v$ proving the first part of this claim.

Next, we prove the second part. Since $\mathbf{X_i} = 0$, the algorithm $\textsc{Delete}(v)$ neither has called $\textsc{ConstructLevel}$ nor it terminates its execution until level L_i. Thus, $\mathbf{R_i} = \mathbf{R_i^-} - v$, which implies that $v \notin \mathbf{R_i}$. However, if we had $v \in R_i$, then the event $[\mathbf{H}_i = H_i, \mathbf{X_i} = \mathbf{0}]$ would have been impossible. ∎

Using Claim B.5, we know that $e_j \neq v$ for $j < i$ and $v \notin R_i$. However, we also know that $v \in R_j^-$ for $j \leq i$. Thus, we can define $H_i^- = (e_1^-, \ldots, e_{i-1}^-, R_0^-, \ldots, R_i^-)$ based on $H_i = (e_1, \ldots, e_{i-1}, R_0, \ldots, R_i)$ as follows:

$$H_i^- = (e_1, \ldots, e_{i-1}, R_0 \cup \{v\}, \ldots, R_i \cup \{v\}) \ .$$

Claim B.6. *Two events* $[\mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{X_i} = 0]$ *and* $[\mathbf{T}^- \geq i, \mathbf{H}_i^- = H_i^-, \mathbf{e}_i^- \neq v]$ *are equivalent (i.e., they imply each other).*
Proof. We first prove that the event $[\mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{X_i} = 0]$ implies the event $[\mathbf{T}^- \geq i, \mathbf{H}_i^- = H_i^-, \mathbf{e}_i^- \neq v]$. Indeed, since $\mathbf{X_i} = 0 \neq 2$ we know that the algorithm ConstructLevel(j) was not invoked for any $j \leq i$ and the element v was contained in $\mathbf{R_j^-}$ for all $j \leq i$. In this case, according to the algorithm Delete(v), we conclude that for any $j \leq i$, we have $\mathbf{e_j^-} \neq v$ and $\mathbf{e_j^-} = \mathbf{e_j}$, and $\mathbf{R_j} = \mathbf{R_j^-} - v$. This means that $\mathbf{R_j^-} = \mathbf{R_j} \cup \{v\}$. Therefore, since $\mathbf{H}_i = H_i$, we must have $\mathbf{H}_i^- = H_i^-$, $\mathbf{e}_i^- \neq v$, and $\mathbf{e}_i^- = \mathbf{e}_i$.

Next, we prove the other way around. That is, the event $[\mathbf{T}^- \geq i, \mathbf{H}_i^- = H_i^-, \mathbf{e}_i^- \neq v]$ implies the event $[\mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{X_i} = 0]$. Indeed, since $\mathbf{H}_i^- = H_i^- = (e_1, \ldots, e_{i-1}, R_0 \cup \{v\}, \ldots, R_i \cup \{v\})$, then, for any $j \leq i$, $v \in \mathbf{R_j^-}$ and for any $j < i$, $\mathbf{e_j^-} = e_j$.

Recall from Claim B.5 that for all $j < i$, $e_j \neq v$ and $v \notin R_i$. Thus, for any $j < i$, we know that $\mathbf{e_j^-} \neq v$. However, we also know that $\mathbf{e}_i^- \neq v$. Thus, $\mathbf{e_j^-} \neq v$ for any $j \leq i$. This essentially means that the algorithm Delete(v) neither invokes ConstructLevel nor terminates its execution till the level L_i. This implies that $\mathbf{X_i} = 0$. On the other hand, the algorithm Delete(v) only removes v from R_i^- and does not make any change in $\mathbf{e_1^-}, \ldots, \mathbf{e_i^-}$. Thus, $\mathbf{R_i} = R_i^- - \{v\} = R_i \cup \{v\} - v = R_i$ and $\mathbf{e}_i = \mathbf{e}_i^-$. Therefore, we have $\mathbf{H}_i = H_i$. □

Therefore, we have the following corollary.
Corollary B.1. $\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{X_i} = 0] = \mathbb{P}\left[\mathbf{e}_i^- = e | \mathbf{T}^- \geq i, \mathbf{H}_i^- = H_i^-, \mathbf{e}_i^- \neq v\right]$.

Thus, in order to prove $\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{X_i} = 0] = \frac{1}{|R_i|} \cdot \mathbb{1}[e \in R_i]$, we can prove

$$\mathbb{P}[\mathbf{e}_i^- = e | \mathbf{T}^- \geq i, \mathbf{H}_i^- = H_i^-, \mathbf{e}_i^- \neq v] = \frac{1}{|R_i|} \cdot \mathbb{1}[e \in R_i] \quad .$$

Recall that the assumption of this lemma is $\mathbb{P}\left[\mathbf{e}_i^- = e | \mathbf{T}^- \geq i, \mathbf{H}_i^- = H_i^-\right] = \frac{1}{|R_i^-|} \cdot \mathbb{1}\left[e \in R_i^-\right]$. That is, conditioned on the event $[\mathbf{T}^- \geq i, \mathbf{H}_i^- = H_i^-]$, the random variable $\mathbf{e}_i^- \sim U(R_i^-)$ is a uniform random variable over the set R_i^-. (i.e., the value e_i of the random variable $\mathbf{e}_i^-$ takes ones of the elements of the set R_i^- uniformly at random.) However, since $X_i = 0$ and using Claim B.6, we have $\mathbf{e}_i^- \neq v$. Thus, conditioned on the event $[\mathbf{T}^- \geq i, \mathbf{H}_i^- = H_i^-, \mathbf{e}_i^- \neq v]$, we have that the random variable $\mathbf{e}_i^- \sim U(R_i^- \backslash \{v\}) = U(R_i)$ should be a uniform random variable over the set $R_i^- \backslash \{v\} = R_i$. Indeed, we have

$$\begin{aligned}\mathbb{P}[\mathbf{e}_i^- = e | \mathbf{T}^- \geq i, \mathbf{H}_i^- = H_i^-, \mathbf{e}_i^- \neq v] &= \frac{\mathbb{P}\left[\mathbf{e}_i^- = e, \mathbf{e}_i^- \neq v | \mathbf{T}^- \geq i, \mathbf{H}_i^- = H_i^-\right]}{\mathbb{P}\left[\mathbf{e}_i^- \neq v | \mathbf{T}^- \geq i, \mathbf{H}_i^- = H_i^-\right]} = \frac{\frac{1}{|R_i^-|} \cdot \mathbb{1}\left[e \in R_i^- \backslash \{v\}\right]}{1 - \frac{1}{|R_i^-|}} \\ &= \frac{1}{|R_i^-| - 1} \cdot \mathbb{1}[e \in R_i^- \backslash \{v\}] = \frac{1}{|R_i|} \cdot \mathbb{1}[e \in R_i] \quad ,\end{aligned}$$

where the second equality holds because of our assumption that the uniform invariant holds before the deletion, and the fourth invariant holds because $R_i^- = R_i \cup \{v\}$ and $v \notin R_i$ proving the case $X = 0$. □
Claim B.7. $\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{X_i} = 1] = \frac{1}{|R_i|} \cdot \mathbb{1}[e \in R_i]$.
Proof. We will be conditioning on possible values of $\mathbf{H}_i^-$.

$$\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i, X_i = 1] = \mathbb{E}_{H_i^-}[\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{X_i} = 1, \mathbf{H}_i^- = H_i^-]] \quad ,$$

where the expectation is taken over all H_i for which $\mathbb{P}\left[\mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{X_i} = 1, \mathbf{H}_i^- = H_i^-\right] > 0$. For all such H_i^-, we claim that this can be further rewritten as $\mathbb{P}\left[\mathbf{T} \geq i, \mathbf{H}_i^- = H_i^-\right]$. This is because Delete(v) is executed deterministically if it does not invoke the algorithm ConstructLevel. Furthermore, the value of $\mathbf{X_i}$ is deterministically determined by $\mathbf{H}_i^-$. Therefore, for any value of H_i^-, either $\mathbf{H}_i^- = H_i^-$ implies $\mathbf{X_i} \neq 1$, in which case $\mathbb{P}\left[\mathbf{T} \geq i, \mathbf{H}_i^- = H_i^-, \mathbf{X_i} = 1\right] = 0$, which is in contradiction with our assumption, or $\mathbf{H}_i^- = H_i^-$ imply $\mathbf{X_i} = 1$. Therefore, for all such H_i^- implies $\mathbf{X_i} = 1$, which also means that ConstructLevel never gets invoked, in which case $\mathbf{H}_i$ is uniquely determined. Hence $\mathbf{H}_i^- = H_i^-$ should also imply that $\mathbf{H}_i = H_i$, as otherwise $\mathbb{P}\left[\mathbf{T} \geq i, \mathbf{H}_i^- = H_i^-, \mathbf{H}_i = H_i\right] = 0$. We therefore obtain:

$$\mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{X_i} = 1, \mathbf{H}_i^- = H_i^-] = \mathbb{P}[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i^- = H_i^-]$$

as claimed.

Also, we know that $\mathbf{H}_i = H_i, \mathbf{X_i} = 1$, implies that:

$$\mathbf{T}^- = \mathbf{T}, \ \ \mathbf{R_i^-} = \mathbf{R_i}, \ \ \mathbf{e}_i^- = \mathbf{e}_i,$$

since it means that the execution of DELETE(v) has terminated before level i, thus no change has been made for that level. Therefore, for a H_i^- used in our expectation, we know that $\mathbf{T} \geq i, \mathbf{H}_i^- = H_i^-$ also implies

$$\mathbf{T}^- \geq i, \;\; R_i^- = R_i, \;\; \mathbf{e}_i^- = \mathbf{e}_i,$$

we have:

$$\mathbb{P}\left[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i^- = H_i^-\right] = \mathbb{P}\left[\mathbf{e}_i^- = e | \mathbf{T}^- \geq i, \mathbf{H}_i^- = H_i^-\right] = \frac{1}{|R_i^-|} \cdot \mathbb{1}\left[e \in R_i^-\right] = \frac{1}{|R_i|} \cdot \mathbb{1}\left[e \in R_i\right],$$

where the third equality holds because of our assumption that the uniform invariant holds before the deletion of element v. Therefore, $\mathbb{P}\left[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{X_i} = 1\right] = \frac{1}{|R_i|} \cdot \mathbb{1}\left[e \in R_i\right]$.

□

CLAIM B.8. $\mathbb{P}\left[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{X_i} = 2\right] = \frac{1}{|R_i|} \cdot \mathbb{1}\left[e \in R_i\right]$.

Proof. By Lemma B.8, we know that the level invariants hold at the end of the execution of DELETE, which is also the end of the execution of CONSTRUCTLEVEL(j). Using Lemma B.5, we know that since the level invariants are going to hold after the execution of CONSTRUCTLEVEL(j), for i which is greater than j, we have:

$$\mathbb{P}\left[\mathbf{e}_i = e | \mathbf{T} \geq i, \mathbf{H}_i = H_i, \mathbf{X_i} = 2\right] = \frac{1}{|R_i|} \cdot \mathbb{1}\left[e \in R_i\right] \quad ,$$

which proves this claim. □

□

B.4 Application of Uniform Invariant: Query complexity In terms of the query complexity of this algorithm, it's important to note that verifying whether an element e is promoting for a level L_z requires a single oracle query. The binary search that we perform needs $O(\log T)$ number of such suitability checks for the element e. Thus, if we initiate the leveling algorithm with a set R_i, our algorithm needs $O(|R_i| \cdot \log(T))$ oracle queries to build the levels $L_i, \cdots, L_T$.

LEMMA B.10. *The number of levels T is at most k.*

Proof. Given the starter and the cardinality invariants, we have $|I_0| = 0$ and $|I_i| = |I_{i-1}| + 1$. We can conclude by induction that $|I_i| = i$. Since we have $I_T = I_{T-1} + e_T$ where PROMOTE(I_{T-1}, e_T) $= True$ by cardinality invariant, the element e_T is promoting for level L_{T-1}. Therefore, $T - 1 < k$ which means $T <= k$. □

LEMMA B.11. *The query complexity of calling* CONSTRUCTLEVEL(i) *is at most* $O(\log(k) \cdot |R_i|)$.

Proof. The algorithm CONSTRUCTLEVEL(i) iterates over all elements in R_i. For each element e, it first calls the PROMOTE function, and select e if it is a promoting element, i.e. PROMOTE($I_{\ell-1}, e$) $= True$. In this case, we only need one query call for checking whether the element is promoting or not. However, if e is not a promoting element, it reaches Line 13 and runs the binary search on interval $[i, \ell - 1]$. Based on Lemma B.10, the length of this interval is $O(k)$. Therefore, the number of steps in binary search is at most $O(\log(k))$. In each step of the binary search, the algorithm calls PROMOTE one time, which takes one query call. Thus, for each element, we need $O(\log(k))$, and for all elements, we need $O(\log(k) \cdot |R_i|)$ query calls. □

LEMMA B.12. *For a specified value of OPT, the query complexity of each update operation in* CARDINALITYCONSTRAINTUPDATES *is at most* $O(k \log(k))$.

Proof. Based on uniform invariant, when we insert/delete an element, for each natural number $i \leq T$, we call CONSTRUCTLEVEL(i) with probability $\frac{1}{|R_i|} \cdot \mathbb{1}\left[e \in R_i\right]$ which is at most $\frac{1}{|R_i|}$. Using Lemma B.11, the query complexity for calling CONSTRUCTLEVEL(i) is $O(\log(k) \cdot |R_i|)$. Therefore, the expected number of queries caused by level i is bounded by $\frac{1}{|R_i|} \cdot O(\log(k) \cdot |R_i|) = O(\log(k))$. As the Lemma B.10 bounded the number of levels by $T = O(k)$, we calculate the expected number of query calls for each update by summing the expected number of query calls at each level:

$$\sum_{i=1}^{T} O(\log(k)) \leq O(k \log(k)) \quad .$$

□

In order to obtain an algorithm that works regardless of the value of OPT, we guess OPT up to a factor of $1 + \epsilon$ using parallel runs. Each element is inserted only to $O(\log(k)/\epsilon)$ copies of the algorithm. Therefore, we obtain the total query complexity claimed in Theorem B.4.

THEOREM B.4. *The expected query complexity of each insert/delete for all runs is* $O\left(\frac{k}{\epsilon} \log^2(k)\right)$.

B.5 Application of Level Invariants: Approximation guarantee Recall that we run parallel instances of our algorithm with different guesses of OPT. In this section, we prove that if the level invariants hold, then after each update, there is a run such that the submodular value of the set I_T in this run is a $(2+\epsilon)$-approximation of the optimal value. Formally, we state this claim as follows:

Theorem B.5. *Suppose that the level invariants hold in every run of our algorithm. Let I_T be the selected set of the final level L_T in each run. If $I^* \subseteq V$ is an optimal subset of size at most k that achieves the optimal value, then there is a run such that the set I_T in that run satisfies $(2+\epsilon)\cdot f(I_T) \geq f(I^*)$.*

Proof. First of all, as described earlier, we run parallel instances of our algorithm with different guesses of OPT such that after each update, there is a run in which $f(I^*) \in (\frac{OPT}{1+\epsilon}, OPT]$. Now, we show that in this run, $(2+\epsilon)\cdot f(I_T) \geq f(I^*)$.

Since the terminator invariant holds, $R_{T+1} = \emptyset$, which means no element promotes level L_T. Thus, either $T = k$ or $f(I_T + e) - f(I_T) < \tau$ for every element $e \in V$. If $T = k$, then since the cardinality invariant holds, we have $f(I_T) = \Sigma_{i=1}^{T} f(I_i) - f(I_{i-1}) = \Sigma_{i=1}^{T} f(I_{i-1} + e_i) - f(I_{i-1}) \geq k\tau = \frac{OPT}{2} \geq \frac{f(I^*)}{2}$.

In the other case, $f(I_T + e) - f(I_T) < \tau$ for every element $e \in V$. By the submodularity and monotonicity of the function f, we have

$$f(I^*) \leq f(I^* \cup I_T) \leq f(I_T) + \Sigma_{e\in I^*\setminus I_T} f(I_T + e) - f(I_T) < f(I_T) + k\tau = f(I_T) + \frac{OPT}{2}\ .$$

Given that in this run $\frac{OPT}{1+\epsilon} < f(I^*)$, we have $f(I^*) < f(I_T) + \frac{1+\epsilon}{2}\cdot f(I^*)$. Thus, we can conclude that $\frac{2}{1-\epsilon}\cdot f(I_T) > f(I^*)$. ∎

C Parameterized Lower Bound

In above, we presented our dynamic 0.5-approximation algorithm that has an amortized query complexity of $O(k\log k)$ if we know the optimal value of the sequence $\mathcal{S}$ after every insertion or deletion, and incurs an extra $O(\log(k)/\epsilon)$-factor in the case that we do not know the optimal value. One may ask if we can obtain a dynamic algorithm for this problem that provides better than 0.5-approximation factor having a query complexity that is still linear, or even polynomial in k. Interestingly, we show there is no dynamic algorithm that maintains a $(0.5+\epsilon)$-approximate submodular solution of the sequence $\mathcal{S}$ using a query complexity that is an arbitrary function $g(k)$ of k (e.g., not even doubly exponentially in k). This hardness holds even when we know the optimal value of the sequence after every insertion or deletion. Thus, the approximation ratio of our parameterized dynamic algorithm is tight. We first state the lower bound due to Chen and Peng [52, Theorem 1.1] in the following lemma.

Lemma C.1. (Theorem 1.1 of [52]) *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 $(0.5+\epsilon)$ for dynamic submodular maximization under cardinality constraint k requires amortized query complexity n^{α_ϵ}/k^3, where $\alpha_\epsilon = \tilde{\Omega}(\epsilon)$ and n is the number of elements in V.*

Chen and Peng proved their theorem by considering a sequence that has the optimal value 1 after every insertion or deletion. Building on this lower bound, we next prove the following theorem.

Theorem C.1. *Let $g : \mathbb{N} \to \mathbb{R}^+$ be an arbitrary function. There is no randomized $(0.5+\epsilon)$-approximate algorithm for dynamic submodular maximization under cardinality constraint k with an expected amortized query time of $g(k)$, even if the optimal value is known after every insertion/deletion.*

Proof. Assume for the sake of contradiction, there exists a constant ϵ, a function $g : \mathbb{N} \to \mathbb{R}^+$, and a $(0.5+\epsilon)$-approximation algorithm for dynamic submodular maximization with at most $g(k)$ amortized query per insertion/deletion. According to Lemma C.1, there is a constant $C_\epsilon > 0$ such that for all $k > C_\epsilon$ and $n \geq k^{2/\epsilon}$, any $(0.5+\epsilon)$-approximation algorithm requires at least n^{α_ϵ}/k^3 amortized query. Let k be an arbitrary natural number such that $k > C_\epsilon$, we define $n_0 := \max((g(k)\cdot k^3)^{-\alpha_\epsilon}, k^{2/\epsilon})$. By the definition of n_0, we have $n_0 \geq (g(k)\cdot k^3)^{-\alpha_\epsilon}$, therefore $n_0^{\alpha_\epsilon} \geq g(k)\cdot k^3$, and then $n_0^{\alpha_\epsilon}/k^3 \geq g(k)$. In conclusion, for any $n > n_0$, as $k^2 \leq n^\epsilon$ constraint holds, Lemma C.1 implies that the amortized query complexity is at least $n^{\alpha_\epsilon}/k^3 > n_0^{\alpha_\epsilon}/k^3 \geq g(k)$, even if we know the optimal value. Thus, the algorithm requires more than $g(k)$ amortized query complexity which is in contradiction with the assumption.

∎

As a result of Theorem C.1, for any $\epsilon > 0$, even if k is a constant, the required amortized query complexity to find a $0.5+\epsilon$-approximate solution increases by increasing n. Therefore, even if we know the optimal value, it is not possible to find a parameterized algorithm with an approximation factor better than 0.5 while query complexity is a function of only k; even if the query complexity is a double exponential of k. For example, if we are looking for an algorithm with amortized query complexity of 2^{2^k}, when n goes up enough, it is not possible to get a $(0.5+\epsilon)$-approximate solution for any $\epsilon > 0$. Hence, the best approximation ratio of an algorithm that is parameterized by only k is 0.5, even with the assumption of knowing the optimal value. Surprisingly, the parameterized dynamic 0.5-approximation algorithm that we presented above has expected amortized query complexity of $O(k \log k)$ if we know the optimal value.