A Dynamic Algorithm for Weighted Submodular Cover Problem

Kiarash Banihashem [*1] **Samira Goudarzi** [*1] **MohammadTaghi Hajiaghayi** [*1] **Peyman Jabbarzade** [*1]
Morteza Monemizadeh [*2]

Abstract

We initiate the study of the submodular cover problem in dynamic setting where the elements of the ground set are inserted and deleted. In the classical submodular cover problem, we are given a monotone submodular function $f : 2^V \to \mathbb{R}^{\geq 0}$ and the goal is to obtain a set $S \subseteq V$ that minimizes the cost subject to the constraint $f(S) = f(V)$. This is a classical problem in computer science and generalizes the Set Cover problem, 2-Set Cover, and dominating set problem among others. We consider this problem in a dynamic setting where there are updates to our set V, in the form of insertions and deletions of elements from a ground set $\mathcal{V}$, and the goal is to maintain an approximately optimal solution with low query complexity per update. For this problem, we propose a randomized algorithm that, in expectation, obtains a $(1 - O(\epsilon), O(\epsilon^{-1}))$-bicriteria approximation using polylogarithmic query complexity per update.

1. Introduction

Submodular optimization is a classical problem in computer science and machine learning with applications spanning various domains such as data summarization, active learning, network inference, video analysis, and facility location (see (Krause, 2013) for a survey).

The submodular cover problem, initially introduced by (Wolsey, 1982), is a well-studied classical variant of the problem where the objective is to minimize the sum of the weight of selected elements chosen from a set subject to a submodular function constraint. Specifically, given a set of elements V, a monotone submodular function $f : 2^V \to \mathbb{R}^{\geq 0}$, and a weight function $w : V \to \mathbb{R}^{\geq 0}$, we seek to pick a set S minimizing $\sum_{v \in S} w(v)$ that satisfies $f(S) = f(V)$.

This problem generalizes various noteworthy problems such as the set cover problem, 2-set cover, dominating set, and others. It can also be seen as a dual of the submodular maximization problem, in which the goal is to maximize $f(S)$ subject to the constraint $|S| \leq k$ for some parameter k.

While the submodular cover problem has been extensively studied (see (Bar-Ilan et al., 2001) for a survey), the majority of the algorithms in the literature predominantly depend on having access to the entire ground set throughout their execution, which is not a valid assumption in numerous real-world applications dealing with ever-changing data and makes them impractical.

Given the mentioned limitation, there has recently been a surge of interest in reexamining classical problems under a variety of massive data models such as streaming, distributed, dynamic, and online settings. For submodular maximization, the problem has been considered in the streaming (Badanidiyuru et al., 2014; Chakrabarti & Kale, 2015; Mirzasoleiman et al., 2018; Kazemi et al., 2019), distributed (Mirrokni & Zadimoghaddam, 2015; Liu & Vondrák, 2018), and dynamic (Monemizadeh, 2020; Lattanzi et al., 2020; Chen & Peng, 2022; Duetting et al., 2023; Banihashem et al., 2023a;b; 2024) settings. Similarly for submodular cover, recent works have studied the problem in the distributed, streaming, and scalable settings (Mirzasoleiman et al., 2015; Norouzi-Fard et al., 2016; Chen & Crawford, 2023; Crawford, 2023).

Motivated by these advances, we consider the submodular cover in a dynamic setting where the elements of the ground set are inserted and deleted, and the goal is to always

[*]Equal contribution [1]Department of Computer Science, University of Maryland, MD, USA [2]Department of Mathematics and Computer Science, TU Eindhoven, the Netherlands. Correspondence to: Samira Goudarzi <samirag@umd.edu>, Kiarash Banihashem <kiarash@umd.edu>.

Proceedings of the 41^{st} International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

maintain an approximately optimal solution. While this can easily be done by re-running an offline algorithm after each update, the goal is to do this with small update time per query. We formally define the dynamic submodular cover problem as follows.

Definition 1.1 (Dynamic Submodular Cover problem)**.** We assume that $f : 2^{\mathcal{V}} \to \mathbb{R}^+$ is a monotone, non-negative submodular function on the ground set $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$, and each element v in the $\mathcal{V}$ has a weight denoted by $w(v)$. For any subset $S \subseteq \mathcal{V}$, $\textsc{Cost}(S)$ is defined to be $\sum_{v \in S} w(v)$. At each time t, the objective of the problem is to choose a subset $S \subseteq V_t$ of minimum cost whose submodular value is equal to $f(V_t)$, i.e.,

$$S_{opt_t} = \arg\min_{S \subseteq V_t} \{\textsc{Cost}(S) : f(S) = f(V_t)\} \ ,$$

where V_t denotes the set of the currently present elements after the first t updates. $\text{OPT}_{\text{cost}_t}$ is defined to be $\textsc{Cost}(S_{opt_t})$, and $f(S_{opt_t}) = f(V_t)$.

Note that, throughout the paper, as we consider a fixed point of time, we drop the subscript t for simplicity.

We note that while (Gupta & Levin, 2020) also considered the submodular cover problem in a dynamic setting, their model is different as it assumes that the submodular function f is changing dynamically, whereas we assume that the ground set undergoes updates. To illustrate the difference, consider the special case of set cover where the elements of V correspond to sets in a set system and we define $f(\mathcal{S}) := |\cup_{S \in \mathcal{S}} S|$ for any $\mathcal{S} \subseteq V$. In this case, the model in (Gupta & Levin, 2020) assumes that the elements of the set system are inserted and deleted, while our model assumes that sets of the set system are inserted and deleted. Our model is consistent with the models considered for the streaming version of the problem (Norouzi-Fard et al., 2016) where the elements are inserted one by one (but the elements are never deleted in their setting) and dynamic setting considered for the submodular maximization problem(Monemizadeh, 2020; Lattanzi et al., 2020; Chen & Peng, 2022; Duetting et al., 2023; Banihashem et al., 2023a;b; 2024).

In this paper, we assume that the updates are specified by an *oblivious adversary*, that is an adversary who knows are algorithm but does not have access to the random bits we use. This is equivalent to assuming that all of the updates are specified before the algorithm is run and as such are not adapted to the algorithm's output.

1.1. Our Contribution

In this paper, we design an algorithm for the dynamic submodular cover problem that maintains an approximately optimal solution using polylogarithmic update time. As is standard for the submodular cover problem (Norouzi-Fard et al., 2016; Chen & Crawford, 2023), our approximation guarantees are bicriteria given the two objectives of the problem. A set S is called a $(1-\epsilon, c)$-bicriteria approximate solution if it satisfies

$$f(S) \geq (1-\epsilon) f(V), \quad \text{and} \quad \textsc{Cost}(S) \leq c\, \textsc{Cost}(S_{opt}),$$

where S_{opt} denotes the optimal solution. We say a (random) set S is *expected* $(1-\epsilon, c)$-bicriteria, if the first guarantee holds in expectation, i.e., $\mathbb{E}[\, f(S)\,] \geq (1-\epsilon) f(V)$. Our main result is stated in the following theorem.

Theorem 1.2. *Define the* weight ratio *of* $\mathcal{V}$ *as* $\rho := \frac{\max_{v \in \mathcal{V}} w(v)}{\min_{v \in \mathcal{V}} w(v)}$, *and set* $n := |\mathcal{V}|$. *For any* $\epsilon > 0$, *there is an algorithm for the dynamic submodular cover problem that maintains an expected* $(1 - O(\epsilon), O(\epsilon^{-1}))$*-bicriteria approximate solution with expected amortized* $\text{poly}(\log(n), \log(\rho), \varepsilon^{-1})$ *update time query complexity.*

In terms of techniques, we build on and generalize the recent advances for dynamic submodular maximization (Monemizadeh, 2020; Lattanzi et al., 2020; Banihashem et al., 2023a), in particular the multi-level construction proposed by (Banihashem et al., 2023a), but require important changes given the "two-dimensional" nature of the problem involving both submodular value $f(.)$ and the weights $w(.)$. Indeed, the underlying "static" algorithm in our approach can be seen as a generalization of the algorithm in (Norouzi-Fard et al., 2016) that can support arbitrary weights (as opposed to the uniform weight setting of (Norouzi-Fard et al., 2016)). Our bucketing structure is two-dimensional in order to handle the effect of deletions, unlike (Banihashem et al., 2023a). We handle our parallel runs, and solution retrieval differently. Additionally, to simplify the analysis of the approximation guarantee, we check the marginal density of each element in the solution at the time of forming our solution sets, as opposed to (Banihashem et al., 2023a) who add their elements in bulk.[1] As such, we do not need the complicated potential function analysis used in (Banihashem et al., 2023a), which is crucial for simplifying the analysis given our more involved two-dimensional setting.

[1] We note that the same idea is used in the corrected version of (Lattanzi et al., 2020).

2. Related Work

Submodular cover The offline version of submodular cover has been extensively studied and it is well-known that the greedy algorithm by (Wolsey, 1982) obtains a logarithmic approximation ratio for the problem (see (Bar-Ilan et al., 2001) for a survey of developments and applications). Recently, given the emergence of Big Data algorithms, there has been an interest in considering the problem in the streaming setting. In particular, (Norouzi-Fard et al., 2016) obtain a $(1-\epsilon, O(\epsilon^{-1}))$-bicriteria approximation algorithm for the problem in the special case where the weights are uniform, i.e., each element has weight 1. Here, the elements of the ground set arrive in a stream and the goal is to build an approximately optimal solution with low-memory. As mentioned earlier, our underlying static algorithm can be seen as a generalization of this approach for handling arbitrary weights. Subsequent works have considered the non-monotone objective functions and designed scalable algorithms for the problem (Crawford, 2023; Chen & Crawford, 2023)

For the dynamic setting, (Gupta & Levin, 2020) consider a different variant of the problem in which the submodular function f changes overtime and obtain a fully dynamic algorithm with bounded recourse. In contrast, our approach assumes that the underlying function is fixed and the ground set changes. This is aligned with the models considered in the streaming setting (Norouzi-Fard et al., 2016; Crawford, 2023) as well the models considered for streaming and dynamic settings for the submodular maximization problem (Monemizadeh, 2020; Lattanzi et al., 2020).

Submodular maximization A closely related problem to submodular cover is submodular maximization. In the classical version of this problem, we are given a ground set of elements V, a submodular function $f : 2^V \to \mathbb{R}^{\geq 0}$ and a parameter k, and the goal is to maximize the function f over all sets S of size at most k. For the offline version of the problem, it is well-known that a standard greedy algorithm that iteratively chooses a remaining element with maximum marginal gain obtains an approximation ratio of $1 - 1/e$ (Nemhauser et al., 1978). The approximation ratio cannot be improved efficiently under complexity assumptions as shown by (Feige, 1998) via a reduction from set cover.

In the streaming setting, submodular maximization was first studied by (Badanidiyuru et al., 2014) who obtained a $1/2 - \epsilon$-approximation algorithm. The $1/2$ bound was later shown to by (Norouzi-Fard et al., 2018). The study of the dynamic version of the problem was first initiated independently by (Monemizadeh, 2020) and (Lattanzi et al., 2020) who obtain $(1/2 - \epsilon)$-approximation algorithms with $O(k^2 \log^2(n)\epsilon^{-3})$ and $O(\log^8(n)\epsilon^{-6})$ update times respectively. [2] The $1/2$ approximation is essentially tight as shown by (Chen & Peng, 2022) using a lower bound construction based on the streaming version of the problem. Recent works have generalized the dynamic results for non-monotone objectives (Banihashem et al., 2023b), as well as matroid constraints (Duetting et al., 2023; Banihashem et al., 2024).

Deletion robust algorithms A closely related but distinct area to dynamic submodular optimization is the robust submodular optimization (Mirzasoleiman et al., 2017; Kazemi et al., 2018; Duetting et al., 2022) in which the goal is to obtain a set that is robust to deletions performed by the adversary. The number of deletions is known upfront, is bounded. In contrast, the dynamic model assumes that insertions and deletions are performed arbitrarily and the goal is to always maintain a good solution.

3. Preliminaries

Notation: For a natural number, the set $\{1, 2, \ldots, x\}$ is denoted as $[x]$. Bold letters represent random variables, while their non-bold counterparts denote specific values. For instance, a random variable is denoted as $\mathbf{X}$ and its value as X. Probability and expectation of a random variable $\mathbf{X}$ are represented by $\Pr[\mathbf{X}]$ and $\mathbb{E}[\mathbf{X}]$ respectively. The notation $\Pr[A|B]$ denotes the conditional probability of event A given event B. For an event A with nonzero probability and a discrete random variable $\mathbf{X}$, the conditional expectation of $\mathbf{X}$ given A is denoted as $\mathbb{E}[\mathbf{X}|A] = \sum_x x \cdot \Pr[\mathbf{X} = x|A]$. Likewise, for discrete random variables $\mathbf{X}$ and $\mathbf{Y}$, the conditional expectation of $\mathbf{X}$ given $\mathbf{Y}$ is denoted as $\mathbb{E}[\mathbf{X}|\mathbf{Y} = y]$. The *indicator function* of an event E is denoted by $\mathbb{1}\{E\}$, where $\mathbb{1}\{E\}$ is assigned one if E occurs and zero otherwise.

Submodular functions: Consider a *non-negative* utility function $f : 2^{\mathcal{V}} \to \mathbb{R}^+$ defined on the given ground set $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$. For any element $v \in \mathcal{V}$ and a set $A \subseteq \mathcal{V}$, $\Delta(v|A)$ is called marginal gain of element v with respect to S and it is defined as $\Delta(v|A) := f(A \cup \{v\}) - f(A)$. Similarly, for any sets $A, B \subseteq \mathcal{V}$, $\Delta(B|A)$ is defined as $f(A \cup B) - f(A)$. Function f is called *submodular* when

[2] The original version of (Lattanzi et al., 2020) had correctness issues in the proof pointed by out (Banihashem et al., 2023a) who provided an alternative algorithm with polylogarithmic update time. The issues were subsequently fixed by (Lattanzi et al., 2020).

for any $A, B \subseteq \mathcal{V}$, we have $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$ or equivalently when for any $A \subseteq B \subseteq \mathcal{V}$ and element v, we have $\Delta(v|A) \geq \Delta(v|B)$. This function f is called *monotone* when for any $A \subseteq B \subseteq \mathcal{V}$, we have $f(A) \leq f(B)$, and it is called *normalized* when $f(\emptyset) = 0$.

Density: Given a submodular function $f : 2^{\mathcal{V}} \to \mathbb{R}^+$ defined on the given ground set $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$, and a weight function $w : \mathcal{V} \to [1, \rho]$, we define density of an element $v \in \mathcal{V}$ as $d(v) := \frac{f(v)}{w(v)}$. Similarly, for any set $A \subseteq \mathcal{V}$ we define marginal density of element v with respect to A as $d(v|A) := \frac{\Delta(v|A)}{w(v)}$.

Update time: We assume access to the monotone submodular function $f : 2^{\mathcal{V}} \to \mathbb{R}^+$ through an *oracle*. This oracle supports *set queries*, allowing one to inquire about the value $f(A)$ for any subset $A \subseteq \mathcal{V}$. In this paper, we measure running time based on the total number of oracle calls, a common practice in submodular optimization, as the processing time of oracle calls typically dominates the running time of other parts of the algorithm (Duetting et al., 2023; Banihashem et al., 2024). We refer to the amortized number of query calls as query complexity and update time.

4. Dynamic Algorithm

4.1. Setting

In the dynamic version of the problem, a sequence of updates, comprising insertions and deletions of elements from the ground set $\mathcal{V}$, alters the set of the current elements denoted by V. Each element may undergo multiple insertions and deletions. At each time frame, the set $V \subseteq \mathcal{V}$ encompasses all inserted elements that have not been deleted since their last insertion. The algorithm's objective is to maintain a solution after each update, with its performance being assessed by its query complexity for each update. It is assumed that ϵ and ϵ_{del} are small enough parameters satisfying $\epsilon < 1/10$ and $\epsilon_{\text{del}} < \epsilon/16$. And lastly, it is also assumed that the weight of each element in $\mathcal{V}$ is between 1 and the parameter ρ.

4.2. Overview of the Algorithms

Our algorithm operates through multiple runs, each assigned a specific threshold parameter, denoted as τ, given to them as their input parameter. This threshold is a critical input used to assess the usefulness of the elements and is employed as a measure to distinguish between valuable and insignificant elements that are no longer relevant. In each time frame, the run with threshold τ that meets some specific criteria will have the appropriate solution for that time frame.

A pivotal aspect of our algorithm is the use of a data structure in each run for keeping the elements, from which we can easily retrieve its solution, and it can efficiently be updated.

The fundamental idea behind this data structure is its hierarchical structure. This structure comprises different levels, where each level ℓ includes the sets L_ℓ and G_ℓ. The family of sets G_ℓ is used to retrieve the solution. Each of them stores the elements that have been selected by the algorithm up to that level, and these sets form a cumulative hierarchy. The set L_ℓ encompasses the elements with a marginal density of at least τ with respect to $G_{\ell-1}$.

Notably, the reconstruction of the entire data structure is a significant operation with potential query complexity implications and one of the key design features of a leveled data structure is its partial reconstruction capability. To further explain, throughout the execution of the algorithm, we can partially reconstruct the data structure starting from the level of our choosing without affecting the previous levels. This feature enables us to handle the insertion or deletion of an element with minimal changes to most levels of our data structure.

Note that even partial reconstructions are heavy operations, and it is in our best interest to avoid them as long as possible, which is why we utilize partial reconstruction only after reaching a level that is heavily affected by the updates up to that point in time and its reconstruction is necessary. To achieve this, in addition to the sets L_ℓ and G_ℓ, we also maintain a set D and extended sets $\overline{L}_\ell$ to keep track of the inserted and deleted elements, triggering reconstruction when deemed necessary.

To further clarify, when an element is deleted instead of removing that element from the sets L_ℓ and G_ℓ, we just add it to the set D, and when an element is inserted we add it to the set $\overline{L}_\ell$ without changing the sets L_ℓ. While iterating through the levels to make these changes, the sets of each level get inspected and a reconstruction starting from that level is triggered if certain criteria are met. It should be noted that a set L_ℓ gets updated whenever and only when its level is being reconstructed.

During the formation of levels, or to be more exact when elements from L_ℓ are being selected for inclusion in G_ℓ, the elements of L_ℓ are grouped into different buckets, so the elements from the same bucket are approximately similar in aspects of their marginal gain, weight, and marginal density.

Then, the largest bucket, denoted as B_ℓ, is chosen, and a suitable number m_ℓ for the sample size is determined based on the chosen bucket. Consequently, a uniformly random subset of size m_ℓ gets chosen from the previously mentioned largest bucket B_ℓ to form the samples S_ℓ. We then form G_ℓ by adding elements of S_ℓ to $G_{\ell-1}$ one by one if they meet our marginal density requirement. We then remove all elements e with $d(e|G_\ell) \leq \tau$ from L_ℓ to form $L_{\ell+1}$.

Now, we proceed to explain more about what we meant by a suitable number for sample size.

Choosing a smaller sample size ensures a larger fraction of elements from S_ℓ appear in G_ℓ, reducing the impact of deletions of elements of B_ℓ on the marginal gain of level ℓ, which leads to less need in invoking reconstruction starting from level ℓ. Conversely, a larger sample size may lead to more substantial removals in the filtering step, impacting the number of the levels of the data structure leading to less query complexity for each reconstruction. This is why we use simulation to obtain a sample size to strike a balance and end up with a low overall query complexity.

4.3. Parallel Runs

We keep parallel runs and designate the threshold $\tau = (1+\epsilon)^i$ to the run i.

In the section B, we guarantee that at each point of time, the output of the run with threshold τ, where $\tau \leq \frac{f(V)\cdot\epsilon}{\text{OPT}_{\text{cost}}} < (1+\epsilon)\tau$ is an appropriate bicriteria approximation of the solution in that time frame.

We know that $\tau \leq \frac{f(V)\cdot\epsilon}{\text{OPT}_{\text{cost}}} < (1+\epsilon)\tau$ is equivalent to $\log_{1+\epsilon}(\tau) \leq \log_{1+\epsilon}(\frac{f(V)\cdot\epsilon}{\text{OPT}_{\text{cost}}}) < 1 + \log_{1+\epsilon}(\tau)$. Therefore, it is guaranteed that the output of the instance with index $\left\lfloor \log_{1+\epsilon}(\frac{f(V)\cdot\epsilon}{\text{OPT}_{\text{cost}}}) \right\rfloor$, has an appropriate solution.

We know that $1 \leq \text{OPT}_{\text{cost}} \leq |V|\rho$, so we have $\frac{f(V)\cdot\epsilon}{|V|\rho} \leq \frac{f(V)\cdot\epsilon}{\text{OPT}_{\text{cost}}} \leq f(V)\cdot\epsilon$, which implies $\log_{1+\epsilon}(\frac{f(V)\cdot\epsilon}{|V|\rho}) \leq \log_{1+\epsilon}(\frac{f(V)\cdot\epsilon}{\text{OPT}_{\text{cost}}}) \leq \log_{1+\epsilon}(f(V)\cdot\epsilon)$. Therefore, at any time we only need to search through the instances with index in $[\left\lfloor \log_{1+\epsilon}(\frac{f(V)\cdot\epsilon}{|V|\rho}) \right\rfloor, \lfloor \log_{1+\epsilon}(f(V)\cdot\epsilon) \rfloor]$ to find a valid solution.

It can be observed in our last argument that the guarantee of a proper solution in the run i with $\tau = (1+\epsilon)^i$ when $\tau \leq \frac{f(V)\cdot\epsilon}{\text{OPT}_{\text{cost}}} < (1+\epsilon)\tau$ is sufficient for the correctness of our algorithm. This means that for each run i with $\tau = (1+\epsilon)^i$ we only need to guarantee its correctness when $\tau \leq \frac{f(V)\cdot\epsilon}{\text{OPT}_{\text{cost}}} < (1+\epsilon)\tau$.

Because of the monotonocity of the function f, we know that for any $e \in V$, $f(e) \leq f(V)$. We also know that $d(e) \leq f(e)$, which implies $d(e) \leq f(V)$. We also know that $\text{OPT}_{\text{cost}} \leq n\rho$. Therefore, for any $e \in V$, we have $\frac{d(e)\cdot\epsilon}{n\rho} \leq \frac{f(V)\cdot\epsilon}{\text{OPT}_{\text{cost}}}$. Hence, $\frac{f(V)\cdot\epsilon}{\text{OPT}_{\text{cost}}} < (1+\epsilon)\tau$ only holds when for any $e \in V$, $\frac{d(e)\cdot\epsilon}{n\rho} < (1+\epsilon)\tau$, which is equivalent to $\log_{1+\epsilon}(\frac{d(e)\cdot\epsilon}{n\rho}) < i+1$. This is why an element e can only be considered in runs with $i \geq \log_{1+\epsilon}(\frac{d(e)\cdot\epsilon}{n\rho})$.

It should also be noted that an element e with $d(e) < \tau$ will be automatically ignored by the algorithm. So we also can consider an element e only in the runs with $\log_{1+\epsilon}(d(e)) \geq i$. Therefore, to handle the update of any element e, we only need to invoke UPDATE in instances within the specified range.

Algorithm 1 Parallel Runs

for $i \in \mathbb{Z}$ **do**
  Let $\mathcal{I}_i$ be the instance of our dynamic algorithm, for which $\tau = (1+\epsilon)^i$.
function GLOBALUPDATE(e)
  update(V)
  for each $\log_{1+\epsilon}\left(\frac{d(e)\cdot\epsilon}{n\rho(1+\epsilon)}\right) \leq i \leq \log_{1+\epsilon} d(e)$ **do**
    Invoke UPDATE(e) for instance $\mathcal{I}_i$.
function SOLUTIONRETRIEVAL()
  Let $i^* \in [\left\lfloor \log_{1+\epsilon}(\frac{f(V)\cdot\epsilon}{|V|\rho}) \right\rfloor, \lfloor \log_{1+\epsilon}(f(V)\cdot\epsilon) \rfloor]$ be the index of the instance whose corresponding G_T meets the criteria $f(G_T) \geq (1-O(\epsilon))f(V)$ and its $G_T \backslash D$ has the lowest cost.
  return $G_T \backslash D$ of $\mathcal{I}_{i^*}$.

4.4. Data Structure Construction

The RECONSTRUCT(i) function iteratively constructs a leveled data structure built upon levels $\ell < i$. It starts by updating sets L_i to include the elements inserted since its last update that have pre-approved marginal density with respect to G_{i-1} and to exclude the elements that have been since deleted. It also updates $\overline{L}_i$ based on the current L_i.

Then, a process begins, where in each step, the set G of the current level gets selected, and then the elements get filtered based on their marginal density with respect to the selected G to form the set L of the subsequent level.

This process terminates when there are no elements left in a level's set L, which is when the algorithms sets T to the index of the last nonempty level.

To select each G_i, the elements in L_i get processed, and

each of them gets assigned to a bucket in a two-dimensional array based on their weight and their marginal density. Then the largest bucket gets selected and will be named B_i. The algorithm determines a specific threshold τ_i for level i and calculates a suitable sample size m_i using the CALCSAMPLESIZE function. It then selects a uniform subset S_i from B_i and adds them to G_i one at a time if they still meet the marginal density condition.

Algorithm 2 Data Structure Construction

function INIT(V)
  $L_0 \leftarrow V, \quad G_0 \leftarrow \emptyset, \quad D \leftarrow \emptyset, \quad \overline{L}_0 \leftarrow L_0$
  $L_1 \leftarrow \{e \in L_0 : d(e|G_0) \geq \tau\}, \quad \overline{L}_1 \leftarrow L_1$
  RECONSTRUCT(1)
function RECONSTRUCT(i)
  $L_i \leftarrow \overline{L}_i \backslash D, \quad \overline{L}_i \leftarrow L_i$
  while $L_i \neq \emptyset$ **do**
    for $e \in L_i$ **do**
      $j \leftarrow \left\lfloor \log_{1+\epsilon}(\frac{d(e|G_{i-1})}{\tau}) \right\rfloor$
      $k \leftarrow \left\lfloor \log_{1+\epsilon}(w(e)) \right\rfloor$
      $\text{buck}_{j,k} \leftarrow \text{buck}_{j,k} \cup \{e\}$
    Let $b_{i,1}$ and $b_{i,2}$ be the indices of the largest buck
    $B_i \leftarrow \text{buck}_{b_{i,1},b_{i,2}}, \quad \tau_i \leftarrow (1+\epsilon)^{b_{i,1}} \cdot \tau$
    $m_i \leftarrow \text{CALCSAMPLESIZE}(B_i, G_{i-1}, \tau_i)$
    $S_i = [e_{i,1}, \ldots, e_{i,m_i}] \leftarrow$ Uniform subset of size m_i from B_i
    $G_i \leftarrow G_{i-1}$
    for $e \in S_i$ **do**
      if $d(e|G_i) \geq \tau_i$ **then**
        $G_i \leftarrow G_i \cup e$
    $L_{i+1} \leftarrow \{e \in L_i : d(e|G_i) \geq \tau\}$
    $\overline{L}_{i+1} \leftarrow L_{i+1}$
    $i \leftarrow i + 1$
  $T \leftarrow i - 1$

4.5. Insertion

In the INSERT Function, we manage the insertion of an element e, into our dynamic data structure. First, we remove e from the set of deleted elements D, indicating its active status. Next, we add e to the extended set $\overline{L}_0$. Then, we iterate over the levels, starting from level 1 up to $T + 1$, where T represents the index of the last nonempty level. At each level, we check if the density of e with respect to G_{i-1} is greater than the threshold τ. If so, e is added to the extended set $\overline{L}_i$. Otherwise, e would not be added to the extended set $\overline{L}_i$, and we also no longer need to check the subsequent levels, so we terminate the loop. We also monitor the size of $\overline{L}_i$, and if $|\overline{L}_i|$ exceeds $\frac{3}{2}|L_i|$, we reconstruct the levels starting from Level i and terminate the loop.

4.6. Deletion

The DELETE function handles the removals of the elements. When an element e is deleted, we begin by adding e to the set D, which keeps track of the deleted elements. Then, we iterate over all nonempty levels. In each level, we check if the proportion of deleted elements from the bucket B_i used for sampling S_i exceeds the threshold ϵ. If this condition holds, we trigger a reconstruction of the data structure starting from the current level i using the RECONSTRUCT function and then terminate the loop.

Algorithm 3 Insertion

function INSERT(e)
  $D \leftarrow D \backslash \{e\}$
  $\overline{L}_0 \leftarrow \overline{L}_0 \cup \{e\}$
  for $i \leftarrow 1, \ldots, T + 1$ **do**
    if $d(e|G_{i-1}) < \tau$ **then**
      break
    $\overline{L}_i \leftarrow \overline{L}_i \cup \{e\}$
    if $i = T + 1$ or $|\overline{L}_i| \geq \frac{3}{2} \cdot |L_i|$ **then**
      RECONSTRUCT(i)
      break

Algorithm 4 Deletion

function DELETE(e)
  $D \leftarrow D \cup e$
  for $i \leftarrow 1, \ldots, T$ **do**
    if $|D \cap B_i| \geq \epsilon_{\text{del}} \cdot |B_i|$ **then**
      RECONSTRUCT(i)
      break

4.7. Choice of Sample Size

We know that as we add elements of S_i to G_i, and G_i grows larger, the remaining elements in S_i are less likely to satisfy the marginal density requirement for being added to the G_i. Thus, intuitively, choosing a smaller sample size ensures that a larger fraction of the elements in S_i appear in G_i. Therefore, deleting an ϵ-fraction of S_i would not drastically affect the value of $\Delta(G_i|G_{i-1})$ as opposed to the case where only a few elements of S_i appear in G_i and the deletion of those few elements has a significant impact on the marginal value $\Delta(G_i|G_{i-1})$. Therefore, having a smaller sample size leads to less invocation of RECONSTRUCT function.

On the other hand, choosing a larger m_i ensures that a

larger number of elements will be removed in the filtering step. This will reduce the number of levels of our data structure, which decreases the query complexity of the RECONSTRUCT function.

To balance this trade-off, we first try to find out if we process all the elements in B_i one by one in a random order, for any $j \in [1, |B_i|]$, what is the probability of the j^{th} element being added to G_i and denote such probability by $X^*(j)$. Then we choose the largest integer m_i^* such that $X^*(j) \geq 1 - \epsilon$ for all $j \leq m_i^*$. This choice ensures that

1. In expectation, $(1 - \epsilon)$-fraction of the elements of S_i are added to G_i
2. In expectation, at least ϵ-fraction of the elements in $|B_i|$ have their marginal gain decreased sufficiently at the end of this level. Formally, these elements either do not appear in L_{i+1}, or the index of their bucket decreases.

Yet, since we cannot calculate the exact values of $X(i)$, we estimate these probabilities by simulating, and we obtain a sample size that satisfies properties similar to properties of m_i^* with a high probability.

Here we provide a formal definition for the notion of *suitable sample size* for each level. It can be verified that the definition is chosen such that a single run of CALCSAMPLESIZE(L', G', τ') provides us with a suitable sample size with respect to L', G', and τ', with a high probability. For a proofs we refer to Lemmas D.8 and D.7 in Appendix D, which were used in both query complexity guarantee and approximation guarantee of the algorithm.

Definition 4.1 (Suitable sample size)**.** Given the values L', G', and τ', consider a run of APPLYANDREVERT on these values and let $\mathbf{X}$ be the $|L'| + 1$ dimensional random output. A number $m^* \leq |L'|$ is called a *suitable sample size with respect to* L', G', *and* τ' if:

$$\mathbb{E}\left[\,\mathbf{X}(r)\,\right] \geq 1 - 2\epsilon \text{ for all } r \in [1, m^*] \text{ and}$$
$$\mathbb{E}\left[\,\mathbf{X}(m^* + 1)\,\right] \leq 1 - \frac{\epsilon}{2}.$$

We use M_i^* to denote the set of suitable sample sizes for (B_i, G_{i-1}, τ_i).

5. Theoretical Analysis

In this section, we state our main theoretical results.

Theorem 5.1. *We have provided an algorithm for dynamic submodular cover problem, where weight of each item is guaranteed to be in the range* $[1, \rho]$*, that maintains an expected* $(1 - O(\epsilon), O(\epsilon)^{-1})$*-bicriteria approximate solution with an expected* $\text{poly}(\log(n), \log(\rho), \varepsilon^{-1})$ *amortized oracle queries per update.*

Algorithm 5 CALCSAMPLESIZE

```
function CalcSampleSize(L', G', τ')
    t ← ⌈ 4 (1/ε²) log(n¹²/ε) ⌉
    for j ∈ [1, t] do
        X_j ← ApplyAndRevert(L', G', τ')
    Let m' be the smallest index i ∈ [1, |L'| + 1] for which (1/t)(Σ_{j=1}^{t} X_j(i)) < 1 − ε
    Return m' − 1
function ApplyAndRevert(L', G', τ')
    Let [e_1, ..., e_{|L'|}] be a random permutation of L'
    Let X be a |L'| + 1 dimensional vector initialized to 0.
    G'' ← G'
    for i = 1 to |L'| do
        if d(e_i | G'') ≥ τ' then
            X(i) ← 1
            G'' ← G'' ∪ {e_i}
        else
            continue
    return X
```

Note that as mentioned in the statement of the theorem, our algorithm and our proof use the assumption that weight of each element is in the range $[1, \rho]$. However, it can easily be verified that this theorem proves Theorem 1.2 as dividing the weight of all element by $\min_{v \in \mathcal{V}} w(v)$ guarantees the assumption while it would not change the ratio between the cost of our proposed solution and the optimal solution.

Proof. We prove this theorem using the following theorems 5.2 and 5.3 regarding the approximation guarantees and query complexity guarantee of the dynamic algorithm that we proposed in section 4, respectively. □

Complete and detailed proof of the next theorem and alongside its references can be found in Appendix B.

Theorem 5.2. *Our algorithm maintains an expected* $(1 - O(\epsilon), O(\epsilon)^{-1})$*-bicriteria approximation of the solution.*

Proof. Consider the run whose assigned threshold parameter satisfies $\tau \leq \frac{f(V) \cdot \epsilon}{\text{OPT}_{\text{cost}}} < (1 + \epsilon)\tau$. Lemma B.1 guarantees that output of this run is an expected $(1 - O(\epsilon), O(\epsilon)^{-1})$-bicriteria approximate solution. As explained in 4.3 this

run is included in the instances with an index between $\left\lfloor \log_{1+\epsilon}\left(\frac{f(V)\cdot\epsilon}{|V|\rho}\right)\right\rfloor$ and $\left\lfloor \log_{1+\epsilon}\left(f(V)\cdot\epsilon\right)\right\rfloor$, and Lemma B.4 ensures that G_T of this run satisfies the condition of $f(G_T) \geq (1-\epsilon)f(V)$. Additionally Lemma B.5 ensures that $f(G_T \backslash D)$ of any run with $f(G_T) \geq (1-\epsilon)f(V)$ is an expected $(1-\epsilon)$ approximate of $f(V)$. Therefore, by checking all the instances in the specified range with $f(G_T) \geq (1-\epsilon)f(V)$, and choosing the one with lowest $\textsc{Cost}(G_T \backslash D)$, we are guaranteed to find an expected $(1-O(\epsilon), O(\epsilon)^{-1})$-bicriteria approximate solution for the problem. □

A complete proof of the following theorem and its references can be found in detail in Appendix C.

Theorem 5.3. *The expected amortized query complexity of our algorithm is at most* $\text{poly}(\log(\eta), \frac{1}{\epsilon})$ *per update, where* $\eta := \frac{1+\epsilon}{\epsilon} n\rho$.

Proof. We start by bounding the count of "direct" queries, which come from insertions and deletions. Here, we don't count queries made indirectly through RECONSTRUCT. Each insertion or deletion can result in at most $O(\mathbf{T})$ queries, where $\mathbf{T}$ is the number of levels during the update. According to Lemma C.8, this is capped at $\text{poly}(\log(n), \log(\eta), \frac{1}{\epsilon})$ because $|\widehat{L}_1| \leq n$.

Moving on to "indirect" queries made by RECONSTRUCT, we charge the cost of each $\textsc{Reconstruct}(i)$ call to the updates causing it. If $\textsc{Reconstruct}(i)$ is triggered by an insertion, its cost is charged to $\overline{L}_i \backslash L_i$, and if by a deletion, it's charged to $B_i \cap D$. Each time $\textsc{Reconstruct}(i)$ is called for some i, the expected number of queries is $|\widehat{L}_i|\text{poly}(\log(|\widehat{L}_i|), \log(\eta), \frac{1}{\epsilon})$. However, this cost is spread across at least $\frac{|\widehat{L}_i|}{\text{poly}(\log(\rho), \log(\eta), \frac{1}{\varepsilon})}$ updates due to the reconstruction conditions (The lower bound is chosen considering the reconstruction condition of deletion and size of B_i and will clearly also hold if $\textsc{Reconstruct}(i)$ is triggered by an insertion, because in that case $|\overline{L}_i \backslash L_i|$ is at least $\frac{1}{3}|\overline{L}_i| \geq \frac{1}{3}|\widehat{L}_i|$). Hence, the cost of each charge is at most $\text{poly}(\log(\eta), \frac{1}{\varepsilon})$ (note that $|\widehat{L}_i| \leq n$, and η has both ρ and n as factors). Now, since each update can be charged by $\textsc{Reconstruct}(i)$ only once and only if when the update happens $T > i$ and level i gets affected, we can say each update is charged at most $\text{poly}(\log(\eta), \frac{1}{\varepsilon})$ for each of the levels it affects. And as the expected number of levels during the update is at most $\text{poly}(\log(\eta), \frac{1}{\varepsilon})$ by Lemma C.8, the claim follows. It's important to note that the random bits used to limit the expectation of $\mathbf{T}$ and the ones used to limit the queries for each reconstruction are separate. Since the value $\mathbf{T}$ is known at the update time, it relies on the random bits used before the update. In contrast, the number of queries for each RECONSTRUCT depends on random bits used after (or at the time of) the update. □

Note that the proofs of the Theorems 5.2 and 5.3 use the invariants introduced in Appendix A and proved in Appendix D.

6. Conclusion

In this paper, we explored the dynamic setting of the monotone submodular cover problem. Specifically, we introduced a $(1-O(\epsilon), O(\epsilon^{-1}))$-bicriteria approximation algorithm with polylogarithmic query complexity.

For future research directions, a promising avenue is to refine the query complexity to $\text{poly}(\log(k), \epsilon)$ while making it independent of n.

Moreover, the exploration of the non-monotone version of the submodular cover problem in the dynamic setting remains an open challenge.

Acknowledgements

Partially supported by DARPA QuICC, ONR MURI 2024 award on Algorithms, Learning, and Game Theory, Army-Research Laboratory (ARL) grant W911NF2410052, NSF AF:Small grants 2218678, 2114269, 2347322

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. It contributes to the literature on the weighed submodular cover problem with applications in various fields and ML tasks like data summarization and active set selection. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

References

Badanidiyuru, A., Mirzasoleiman, B., Karbasi, A., and Krause, A. Streaming submodular maximization: Massive data summarization on the fly. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 671–680, 2014.

Banihashem, K., Biabani, L., Goudarzi, S., Hajiaghayi, M., Jabbarzade, P., and Monemizadeh, M. Dynamic con-

strained submodular optimization with polylogarithmic update time. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023a.

Banihashem, K., Biabani, L., Goudarzi, S., Hajiaghayi, M., Jabbarzade, P., and Monemizadeh, M. Dynamic non-monotone submodular maximization. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023b.

Banihashem, K., Biabani, L., Goudarzi, S., Hajiaghayi, M., Jabbarzade, P., and Monemizadeh, M. Dynamic algorithms for matroid submodular maximization. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 3485–3533, 2024. doi: 10.1137/1.9781611977912.125. URL https://epubs.siam.org/doi/abs/10.1137/1.9781611977912.125.

Bar-Ilan, J., Kortsarz, G., and Peleg, D. Generalized submodular cover problems and applications. *Theoretical Computer Science*, 250(1-2):179–200, 2001.

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

Chen, W. and Crawford, V. G. Bicriteria approximation algorithms for the submodular cover problem. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Chen, X. and Peng, B. On the complexity of dynamic submodular maximization. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pp. 1685–1698, 2022.

Crawford, V. Scalable bicriteria algorithms for non-monotone submodular cover. In *International Conference on Artificial Intelligence and Statistics*, pp. 9517–9537. PMLR, 2023.

Duetting, P., Fusco, F., Lattanzi, S., Norouzi-Fard, A., and Zadimoghaddam, M. Deletion robust submodular maximization over matroids. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvári, C., Niu, G., and Sabato, S. (eds.), *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pp. 5671–5693. PMLR, 2022. URL https://proceedings.mlr.press/v162/duetting22a.html.

Duetting, P., Fusco, F., Lattanzi, S., Norouzi-Fard, A., and Zadimoghaddam, M. Fully dynamic submodular maximization over matroids. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J. (eds.), *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 8821–8835. PMLR, 2023. URL https://proceedings.mlr.press/v202/duetting23a.html.

Feige, U. A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998.

Gupta, A. and Levin, R. Fully-dynamic submodular cover with bounded recourse. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 1147–1157. IEEE, 2020.

Kazemi, E., Zadimoghaddam, M., and Karbasi, A. Scalable deletion-robust submodular maximization: Data summarization with privacy and fairness constraints. In Dy, J. G. and Krause, A. (eds.), *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*, pp. 2549–2558. PMLR, 2018. URL http://proceedings.mlr.press/v80/kazemi18a.html.

Kazemi, E., Mitrovic, M., Zadimoghaddam, M., Lattanzi, S., and Karbasi, A. Submodular streaming in all its glory: Tight approximation, minimum memory and low adaptive complexity. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *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*, pp. 3311–3320. PMLR, 2019. URL http://proceedings.mlr.press/v97/kazemi19a.html.

Krause, A. Submodularity in machine learning and vision. In Burghardt, T., Damen, D., Mayol-Cuevas, W. W., and Mirmehdi, M. (eds.), *British Machine Vision Conference, BMVC 2013, Bristol, UK, September 9-13, 2013*. BMVA Press, 2013. doi: 10.5244/C.27.2. URL https://doi.org/10.5244/C.27.2.

Lattanzi, S., Mitrovic, S., Norouzi-Fard, A., Tarnawski, J., and Zadimoghaddam, M. Fully dynamic algorithm for constrained submodular optimization. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems 33:*

Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual, 2020.

Liu, P. and Vondrák, J. Submodular optimization in the mapreduce model. *arXiv preprint arXiv:1810.01489*, 2018.

Mirrokni, V. S. and Zadimoghaddam, M. Randomized composable core-sets for distributed submodular maximization. In Servedio, R. A. and Rubinfeld, R. (eds.), *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pp. 153–162. ACM, 2015. doi: 10.1145/2746539.2746624. URL https://doi.org/10.1145/2746539.2746624.

Mirzasoleiman, B., Karbasi, A., Badanidiyuru, A., and Krause, A. Distributed submodular cover: Succinctly summarizing massive data. *Advances in Neural Information Processing Systems*, 28, 2015.

Mirzasoleiman, B., Karbasi, A., and Krause, A. Deletion-robust submodular maximization: Data summarization with "the right to be forgotten". In Precup, D. and Teh, Y. W. (eds.), *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*, pp. 2449–2458. PMLR, 2017. URL http://proceedings.mlr.press/v70/mirzasoleiman17a.html.

Mirzasoleiman, B., Jegelka, S., and Krause, A. Streaming non-monotone submodular maximization: Personalized video summarization on the fly. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Monemizadeh, M. Dynamic submodular maximization. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/6fbd841e2e4b2938351a4f9b68f12e6b-Abstract.html.

Nemhauser, G. L., Wolsey, L. A., and Fisher, M. L. An analysis of approximations for maximizing submodular set functions—i. *Mathematical programming*, 14:265–294, 1978.

Norouzi-Fard, A., Bazzi, A., Bogunovic, I., El Halabi, M., Hsieh, Y.-P., and Cevher, V. An efficient streaming algorithm for the submodular cover problem. *Advances in Neural Information Processing Systems*, 29, 2016.

Norouzi-Fard, A., Tarnawski, J., Mitrovic, S., Zandieh, A., Mousavifar, A., and Svensson, O. Beyond 1/2-approximation for submodular maximization on massive data streams. In Dy, J. G. and Krause, A. (eds.), *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*, pp. 3826–3835. PMLR, 2018. URL http://proceedings.mlr.press/v80/norouzi-fard18a.html.

Wolsey, L. A. An analysis of the greedy algorithm for the submodular set covering problem. *Combinatorica*, 2(4): 385–393, 1982.

A. Invariants

In this section, we are going to introduce the invariants, which will be used in our analysis. These invariants are guaranteed to hold after each insertion or deletion throughout the execution of the algorithm.

We define the function FILTER as follows:

$$\textsc{Filter}(L, G, \tau) := \{\, e \in L : d(e|G) \geq \tau \,\}.$$

Note that we use $\hat{L}_i$ to denote $\overline{L}_i \backslash D$. Also recall that we use bold letters to denote random variables while non-bold letters are used to denote the value of variables during the execution.

Level invariants

- **Filter invariant**: $\hat{L}_i = \textsc{Filter}(\hat{L}_{i-1}, G_{i-1}, \tau)$ for all $i \in [T+1]$.
- **Subset invariant**: $\overline{L}_i \subseteq \overline{L}_{i-1}$ for all $i \in [T+1]$.
- **Deviation invariant**: $|B_i \cap D| \leq \epsilon_{\text{del}}|B_i|$ and $|\overline{L}_i| \leq \frac{3}{2}|L_i|$ for all $i \in [T]$.
- **Stopping invariant**: $\hat{L}_{T+1} = \overline{L}_{T+1} = L_{T+1} = \emptyset$ and $\hat{L}_i, \overline{L}_i, L_i \neq \emptyset$ for any $i \in [T]$.

Before continuing with the rest of the invariants, we provide a few more definitions.

Definition A.1. We define the pre-sample history of Level i as

$$H_i := (\overline{L}_0, \overline{L}_1, \ldots, \overline{L}_i, L_0, L_1, \ldots, L_i, G_1, \ldots, G_{i-1}, m_i). \tag{1}$$

Intuitively, H_i captures the state, or "history", of Algorithm 2 before $\mathbf{S}_i$ is sampled.

Similarly, we define the pre-size history of Level i to be its pre-sample history minus m_i, i.e.,

$$H_i^{\text{pre}} = (\overline{L}_0, \overline{L}_1, \ldots, \overline{L}_i, L_0, L_1, \ldots, L_i, G_1, \ldots, G_{i-1}). \tag{2}$$

We analogously use $\mathbf{H}_\ell$ and $\mathbf{H}_\ell^{\text{pre}}$ to denote the random variables corresponding to these quantities.

Now we introduce the random invariants of our algorithm.

Sampling invariants

- **Sample uniformity invariant**: Conditioned on the pre-sample history of Level i, the sample set $\mathbf{S}_i$ is a uniformly random subset of size m_i from B_i.

  Formally, for any $i \geq 1$, and any H_i such that $\Pr[\, \mathbf{T} \geq i, \mathbf{H}_i = H_i \,] > 0$,

  $$\Pr[\, \mathbf{S}_i = S | \mathbf{T} \geq i, \mathbf{H}_i = H_i \,] = \frac{1}{|X_i|} \mathbb{1}\{S \in X_i\}, \tag{3}$$

  where $|X_i|$ denotes all sequences of length m_i in B_i.

- **Sample size invariant**: Conditioned on the pre-size history of Level i, m_i is a suitable sample size with a high probability. Formally, for all H_i^{pre} such that $\Pr\left[\, \mathbf{T} \geq i, \mathbf{H}_i^{\text{pre}} = H_i^{\text{pre}} \,\right] > 0$,

  $$\Pr\left[\, \mathbf{m}_i \in M_i^* | \mathbf{T} \geq i, \mathbf{H}_i^{\text{pre}} = H_i^{\text{pre}} \,\right] \geq 1 - \frac{\epsilon}{n^{10}}.$$

We note that while the above two invariants are intuitively evident for a single execution of Algorithm 2 (see Claim D.6), the dynamic algorithm has the potential to modify both S_i and H_i during update processing. Therefore, the result is not immediately clear and needs to be formally proved. Indeed, the proof heavily relies on the fact that the decision to invoke RECONSTRUCT(i) in INSERT and DELETE procedures are based only on H_i^{pre} and not on S_i.

We refer to Appendix D for proofs of these invariants.

B. Approximation guarantee

Lemma B.1. *Assuming that $\tau \leq \frac{f(V)\cdot\epsilon}{\text{OPT}_{cost}} < (1+\epsilon)\tau$, the output (i.e., $G_T \backslash D$) is an expected $(1-O(\epsilon), O(\epsilon)^{-1})$-bicriteria approximation of the solution.*

Proof. We prove this lemma by the combination of the following Lemmas B.2 and B.3. □

Lemma B.2. *Assuming that $\tau \leq \frac{f(V)\cdot\epsilon}{\text{OPT}_{cost}} < (1+\epsilon)\tau$, the following holds:*

$$\textsc{Cost}(G_T \backslash D) < (\frac{1+\epsilon}{\epsilon})\text{OPT}_{cost}.$$

Proof. Let's use $e_1, e_2, \ldots, e_{|G_T|}$ to denote the elements in G_T based on the order they were added to the solution sets.

For any level $1 \leq \ell \leq T$, we know that $\tau_\ell \geq \tau$, and the marginal density of any element in G_T must have been greater than some τ_ℓ at the time of it was added to the solution sets. Therefore, for any $1 \leq i \leq |G_T|$, we know that $d(e_i|\{e_1, \ldots, e_{i-1}\}) = \frac{\Delta(e_i|\{e_1,\ldots,e_{i-1}\})}{w(e_i)} \geq \tau$, or equivalently $w(e_i) \leq \frac{\Delta(e_i|\{e_1,\ldots,e_{i-1}\})}{\tau}$. The submodularity of the function f ensures that $\Delta(e_i|\{e_1, \ldots, e_{i-1}\}) \leq \Delta(e_i|\{e_1, \ldots, e_{i-1}\}\backslash D)$. Hence, for any $1 \leq i \leq |G_T|$, we have $w(e_i) \leq \frac{\Delta(e_i|\{e_1,\ldots,e_{i-1}\}\backslash D)}{\tau}$. Therefore, $\textsc{Cost}(G_T \backslash D) = \sum_{e\in G_T\backslash D} w(e) \leq \frac{f(G_T\backslash D)}{\tau}$. We have assumed that $\frac{1}{(1+\epsilon)\tau} < \frac{\text{OPT}_{\text{cost}}}{f(V)\cdot\epsilon}$, which implies $\frac{1}{\tau} < \frac{(1+\epsilon)\text{OPT}_{\text{cost}}}{f(V)\cdot\epsilon}$. Therefore, we have $\textsc{Cost}(G_T \backslash D) < \frac{(1+\epsilon)\text{OPT}_{\text{cost}}}{\epsilon}(\frac{f(G_T\backslash D))}{f(V)}) \leq \frac{(1+\epsilon)\text{OPT}_{\text{cost}}}{\epsilon}$, where the second inequality follows from the monotonocity of the function f and the fact that $(G_T \backslash D) \subseteq V$.

□

Lemma B.3. *Assuming that $\tau \leq \frac{f(V)\cdot\epsilon}{\text{OPT}_{cost}} < (1+\epsilon)\tau$, the following holds:*

$$\mathbb{E}\,[\, f(\mathbf{G_T}\backslash D)\,] \geq (1-O(\epsilon))f(V).$$

Proof. To prove this theorem, we ignore the deletions at the begining, and investigate the approximations of G_T even though it might include some deleted elements. We provide an upper bound on $f(G_T)$ in Lemma B.4, and then we factor in the removal of deleted elements by bounding the effect of the deleted elements on the value of the function in Lemma B.5. □

Lemma B.4. *Assuming that $\tau \leq \frac{f(V)\cdot\epsilon}{\text{OPT}_{cost}} < (1+\epsilon)\tau$, we have $f(G_T) \geq (1-\epsilon)f(V)$.*

Proof. The statement of the Lemma trivially holds if $f(G_T) \geq f(V)$. Thus, we assume that $f(G_T) < f(V)$. Recall that $f(V) = f(S_{opt})$. By the monotonicity property of function f, we have $f(S_{opt} \cup G_T) \geq f(V)$. Let's denote $S_{opt}\backslash G_T$ by the set $\{u_1, u_2, \ldots, u_d\}$ for some $0 < d \leq |V|$. For each $u \in S_{opt}\backslash G_T$, we know that $u \in V = \hat{L}_0$ and $u \notin \hat{L}_{T+1} = \emptyset$ by stopping invariant (Lemma D.4). Therefore, there exists a level ℓ such that $e \in \hat{L}_\ell$ but $e \notin \hat{L}_{\ell+1}$. By the filter invariant (Lemma D.1) we know that $e \notin filter(\hat{L}_\ell, G_\ell)$. Therefore, we know that $d(e|G_\ell) < \tau$. The former inequality and the submodularity of the function f alongside with the fact that $G_\ell \subseteq G_T$ implies that $d(e|G_T) < \tau$. Hence, for any $i \in [d]$, we have $\frac{\Delta(u_i|G_T)}{w(u_i)} < \tau$. By the submodular property of f, we have:

$$f(S_{opt} \cup G_T) - f(G_T) \leq \sum_{i=1}^{d} \Delta(u_i|G_T).$$

Therefore, we have:

$$f(S_{opt} \cup G_T) - f(G_T) \leq \sum_{i=1}^{d} \tau \cdot w(u_i) = \tau \sum_{i=1}^{d} w(u_i) = \tau \cdot \textsc{Cost}(S_{opt}\backslash G_T).$$

We know that $\textsc{Cost}(S_{opt}\backslash G_T) \leq \textsc{Cost}(S_{opt}) = \text{OPT}_{\text{cost}}$. So

$$f(S_{opt} \cup G_T) - f(G_T) \leq \tau \cdot \text{OPT}_{\text{cost}} \leq \frac{f(V)\cdot\epsilon}{\text{OPT}_{\text{cost}}} \cdot \text{OPT}_{\text{cost}} = f(V)\cdot\epsilon,$$

where the second inequality holds because of the assumption of the Lemma. Previous equation yields that $f(G_T) \geq f(S_{opt} \cup G_T) - (f(V) \cdot \epsilon) \geq f(V) - (f(V) \cdot \epsilon) = (1-\epsilon)f(V)$. □

Now, in the following lemma, we give an upper bound on the effect of the deleted elements and thus prove an expected lower bound for $f(\mathbf{G_T} \backslash D)$.

Lemma B.5. $\mathbb{E}[\, f(\mathbf{G_T} \backslash D)\,] \geq (1 - O(\epsilon))\mathbb{E}[\, f(\mathbf{G_T})\,]$

Proof. Define the variables g_ℓ and l_ℓ to denote the gain and loss of each level $\ell \leq T$ as:

$$g_\ell := \Delta((G_\ell \backslash G_{\ell-1}) | G_{\ell-1}), \quad l_\ell := \Delta(((G_\ell \backslash G_{\ell-1}) \cap D) | G_{\ell-1}).$$

For $\ell \geq T+1$, define $g_\ell = l_\ell = 0$.

we need to show that

$$\mathbb{E}\left[\sum_{\ell=1}^{\infty} (\mathbf{g}_\ell - \mathbf{l}_\ell)\right] \geq (1 - O(\epsilon))\mathbb{E}\left[\sum_{\ell=1}^{\infty} \mathbf{g}_\ell\right].$$

Note that the summation is over all $\ell \geq 1$ and $g_\ell = l_\ell = 0$ for $\ell > T$. The above inequality is equivalent to

$$\mathbb{E}\left[\sum_{\ell=\mathbf{1}}^{\infty} \mathbf{l}_\ell\right] \leq O(\epsilon)\mathbb{E}\left[\sum_{\ell=\mathbf{1}}^{\infty} \mathbf{g}_\ell\right] \tag{4}$$

We will show that $\mathbb{E}[\, \mathbf{l}_\ell\,] \leq O(\epsilon)\mathbb{E}[\, \mathbf{g}_\ell\,]$ holds for all $\ell \geq 1$. Summing over ℓ, we obtain the equation (4) by linearity of expectation.

By the law of total expectation, it suffices to prove

$$\mathbb{E}\left[\, \mathbf{l}_\ell | \mathbf{H}_\ell^{\text{pre}} = H_\ell^{\text{pre}}\,\right] \leq O(\epsilon)\mathbb{E}\left[\, \mathbf{g}_\ell | \mathbf{H}_\ell^{\text{pre}} = H_\ell^{\text{pre}}\,\right], \tag{5}$$

for all H_ℓ^{pre} such that $\Pr\left[\, \mathbf{H}_\ell^{\text{pre}} = H_\ell^{\text{pre}}\,\right] > 0$. To do this, we will first show that the claim holds if $\mathbf{m}_\ell \in M_\ell^*$. Given the Sample size invariant (Lemma D.8), $\mathbf{m}_\ell \in M_\ell^*$ holds with high probability, which will later allow us to prove Equation (5).

Consider any value $m_\ell \in M_\ell^*$ and define H_ℓ as $(H_\ell^{\text{pre}}, m_\ell)$. Assuming that H_ℓ is such that $\Pr[\, \mathbf{T} \geq \ell, \mathbf{H}_\ell = H_\ell\,] > 0$, we can claim that

$$\mathbb{E}[\, \mathbf{l}_\ell | \mathbf{H}_\ell = H_\ell\,] \leq O(\epsilon)\mathbb{E}[\, \mathbf{g}_\ell | \mathbf{H}_\ell = H_\ell\,]. \tag{6}$$

To prove this, we first note that we are taking m_ℓ samples from B_ℓ, and the weight of all the elements in B_ℓ is in range $[(1+\epsilon)^k, (1+\epsilon)^{k+1})$ for some integer $0 \leq k \leq \lfloor \log_{1+\epsilon}(\rho) \rfloor$.

To prove this, we first note that we are taking m_ℓ samples from B_ℓ, and the weight of all the elements in B_ℓ is in range $[(1+\epsilon)^k, (1+\epsilon)^{k+1})$ for some integer $0 \leq k \leq \lfloor \log_{1+\epsilon}(\rho) \rfloor$.

Lemma B.6 implies that

$$\mathbb{E}[\, \mathbf{g}_\ell | \mathbf{H}_\ell = H_\ell\,] \geq (1 - 2\epsilon) \cdot m_\ell \cdot \tau_\ell \cdot (1+\epsilon)^k. \tag{7}$$

Furthermore, considering deviation invariant (Lemma D.3), $|D \cap B_\ell| \leq \epsilon \cdot |B_\ell|$, and given the sample uniformity invariant (Lemma D.5), which states $\mathbf{S}_\ell$ is a uniformly random subset of $|B_\ell|$ of size m_ℓ, we have:

$$\mathbb{E}[\, |D \cap \mathbf{S}_\ell| \mid \mathbf{H}_\ell = H_\ell\,] \leq \epsilon \cdot m_\ell.$$

Since $G_\ell \backslash G_{\ell-1}$ is a subset of S_ℓ, we also have $\mathbb{E}[\, |D \cap (G_\ell \backslash G_{\ell-1})|\,] \leq \epsilon \cdot m_\ell$. Since marginal density of each element in $G_\ell \backslash G_{\ell-1}$ is at most $\tau_\ell \cdot (1+\epsilon)$, which implies marginal value of each element is at most $\tau_\ell \cdot (1+\epsilon) \cdot (1+\epsilon)^{k+1}$, we have

$$\mathbb{E}[\, \mathbf{l}_\ell | \mathbf{H}_\ell = H_\ell\,] \leq \epsilon(1+\epsilon) \cdot m_\ell \cdot \tau_\ell \cdot (1+\epsilon)^{k+1},$$

which together with Equation (7), proves Equation (6). Equation (6) and law of total expectation imply that

$$\mathbb{E}\left[\,\mathbf{l}_\ell|\mathbf{H}_\ell^{\text{pre}}=H_\ell^{\text{pre}},\mathbf{m}\in M_\ell^*\,\right]\leq O(\epsilon)\mathbb{E}\left[\,\mathbf{g}_\ell|\mathbf{H}_\ell^{\text{pre}}=H_\ell^{\text{pre}},\mathbf{m}\in M_\ell^*\,\right]. \tag{8}$$

We have:

$$\begin{aligned}\mathbb{E}\left[\,\mathbf{g}_\ell|\mathbf{H}_\ell^{\text{pre}}=H_\ell^{\text{pre}}\,\right]&\geq \Pr\left[\,\mathbf{m}_\ell\in M_\ell^*|\mathbf{H}_\ell^{\text{pre}}=H_\ell^{\text{pre}}\,\right]\cdot\mathbb{E}\left[\,\mathbf{g}_\ell|\mathbf{H}_\ell^{\text{pre}}=H_\ell^{\text{pre}},\mathbf{m}_\ell\in M_\ell^*\,\right] &(9)\\ &\geq(1-O(\epsilon))\cdot\mathbb{E}\left[\,\mathbf{g}_\ell|\mathbf{H}_\ell^{\text{pre}}=H_\ell^{\text{pre}},\mathbf{m}_\ell\in M_\ell^*\,\right] &(10)\\ &\geq\frac{1}{2}\mathbb{E}\left[\,\mathbf{g}_\ell|\mathbf{H}_\ell^{\text{pre}}=H_\ell^{\text{pre}},\mathbf{m}_\ell\in M_\ell^*\,\right] &(11)\end{aligned}$$

On the other hand, we have:

$$\begin{aligned}\mathbb{E}\left[\,\mathbf{l}_\ell|\mathbf{H}_\ell^{\text{pre}}=H_\ell^{\text{pre}}\,\right]&=\Pr\left[\,\mathbf{m}_\ell\in M_\ell^*|\mathbf{H}_\ell^{\text{pre}}=H_\ell^{\text{pre}}\,\right]\cdot\mathbb{E}\left[\,\mathbf{l}_\ell|\mathbf{H}_\ell^{\text{pre}}=H_\ell^{\text{pre}},\mathbf{m}_\ell\in M_\ell^*\,\right]\\ &\quad+\Pr\left[\,\mathbf{m}_\ell\notin M_\ell^*|\mathbf{H}_\ell^{\text{pre}}=H_\ell^{\text{pre}}\,\right]\cdot\mathbb{E}\left[\,\mathbf{l}_\ell|\mathbf{H}_\ell^{\text{pre}}=H_\ell^{\text{pre}},\mathbf{m}_\ell\notin M_\ell^*\,\right]\\ &\leq 1\cdot\mathbb{E}\left[\,\mathbf{l}_\ell|\mathbf{H}_\ell^{\text{pre}}=H_\ell^{\text{pre}},\mathbf{m}_\ell\in M_\ell^*\,\right]+O(\frac{\epsilon}{n^{10}})\cdot\mathbb{E}\left[\,\mathbf{l}_\ell|\mathbf{H}_\ell^{\text{pre}}=H_\ell^{\text{pre}},\mathbf{m}_\ell\notin M_\ell^*\,\right]\\ &\leq O(\epsilon)\mathbb{E}\left[\,\mathbf{g}_\ell|\mathbf{H}_\ell^{\text{pre}}=H_\ell^{\text{pre}},\mathbf{m}_\ell\in M_\ell^*\,\right]+O(\frac{\epsilon}{n^{10}})\cdot\mathbb{E}\left[\,\mathbf{l}_\ell|\mathbf{H}_\ell^{\text{pre}}=H_\ell^{\text{pre}},\mathbf{m}_\ell\notin M_\ell^*\,\right]\\ &\leq O(\epsilon)\mathbb{E}\left[\,\mathbf{g}_\ell|\mathbf{H}_\ell^{\text{pre}}=H_\ell^{\text{pre}},\mathbf{m}_\ell\in M_\ell^*\,\right]+O(\frac{\epsilon}{n^{10}})\mathbb{E}\left[\,\mathbf{g}_\ell|\mathbf{H}_\ell^{\text{pre}}=H_\ell^{\text{pre}},\mathbf{m}_\ell\notin M_\ell^*\,\right],\end{aligned}$$

where the first inequality comes from sample size invariant, the second equality comes Equation (8), and the third one comes from the fact that $\mathbf{l}_\ell\leq\mathbf{g}_\ell$.

The previous equation and the Equation (11) imply that

$$\mathbb{E}\left[\,\mathbf{l}_\ell|\mathbf{H}_\ell^{\text{pre}}=H_\ell^{\text{pre}}\,\right]\leq O(\epsilon)\mathbb{E}\left[\,\mathbf{g}_\ell|\mathbf{H}_\ell^{\text{pre}}=H_\ell^{\text{pre}}\,\right]+O(\frac{\epsilon}{n^{10}})\mathbb{E}\left[\,\mathbf{g}_\ell|\mathbf{H}_\ell^{\text{pre}}=H_\ell^{\text{pre}},\mathbf{m}_\ell\notin M_\ell^*\,\right]. \tag{12}$$

From the definition, we know that $\mathbb{E}\left[\,\mathbf{g}_\ell|\mathbf{H}_\ell^{\text{pre}}=H_\ell^{\text{pre}},\mathbf{m}_\ell\notin M_\ell^*\,\right]\leq\mathbf{m}_\ell\cdot\max_{e\in B_\ell}\{\Delta(e|G_{\ell-1})\}$. Also, $\mathbf{m}_\ell$ clearly is not greater than n. Hence, $\mathbb{E}\left[\,\mathbf{g}_\ell|\mathbf{H}_\ell^{\text{pre}}=H_\ell^{\text{pre}},\mathbf{m}_\ell\notin M_\ell^*\,\right]\leq n\cdot\max_{e\in B_\ell}\{\Delta(e|G_{\ell-1})\}$.

Additionally, we know that the first sampled element from B_ℓ always gets added to G_ℓ as it definitely meets the threshold requirement. Therefore, we know that $\mathbb{E}\left[\,\mathbf{g}_\ell|\mathbf{H}_\ell^{\text{pre}}=H_\ell^{\text{pre}}\,\right]$ is at least equal to the expected marginal gain of a random element in B_ℓ, which is definitely at least $\frac{1}{|B_\ell|}(\max_{e\in B_\ell}\{\Delta(e|G_{\ell-1})\})$. $|B_\ell|$ is also at most n, so $\max_{e\in B_\ell}\{\Delta(e|G_{\ell-1})\}\leq n\cdot\mathbb{E}\left[\,\mathbf{g}_\ell|\mathbf{H}_\ell^{\text{pre}}=H_\ell^{\text{pre}}\,\right]$. Therefore, we have $\mathbb{E}\left[\,\mathbf{g}_\ell|\mathbf{H}_\ell^{\text{pre}}=H_\ell^{\text{pre}},\mathbf{m}_\ell\notin M_\ell^*\,\right]\leq n^2\cdot\mathbb{E}\left[\,\mathbf{g}_\ell|\mathbf{H}_\ell^{\text{pre}}=H_\ell^{\text{pre}}\,\right]$. This last inequality and Equation (12) prove Equation (5) and our proof is complete.

□

Lemma B.6. *Consider a value of H_ℓ for the history up to cache ℓ such that $\Pr[\,\mathbf{H}_\ell=H_\ell\,]>0$, and define $g(A)$ as $f(A|G_{\ell-1})$.Assume that the weights of all the elements in B_ℓ are in range $[(1+\epsilon)^k,(1+\epsilon)^{k+1})$. If $m_\ell\in\mathbf{M}_\ell^*$, then*

$$\mathbb{E}\left[\,\mathbf{g}(\mathbf{G}_\ell\backslash G_{\ell-1})|\mathbf{H}_\ell=H_\ell\,\right]\geq(1-2\epsilon)\cdot\tau_\ell\cdot m_\ell.$$

Proof. By definition of suitable sample size, we know that for all $i\in[m_\ell]$, $\mathbb{E}\left[\,\mathbf{X}(i)\,\right]\geq 1-2\epsilon$. So for each $i\in[m_\ell]$, the i^{th} element of S_ℓ is added to the G_ℓ with a probability of at least $1-2\epsilon$. We also know that each element is added only if its marginal density is at least τ_ℓ, which means its marginal value is at least $\tau_\ell(1+\epsilon)^k$, and increases the value of $f(G_\ell)$ by at least $\tau_\ell(1+\epsilon)^k$. Therefore,

$$\mathbb{E}\left[\,\mathbf{g}(\mathbf{G}_\ell\backslash G_{\ell-1})|\mathbf{H}_\ell=H_\ell\,\right]\geq(1-2\epsilon)\cdot\tau_\ell\cdot m_\ell\cdot(1+\epsilon)^k$$

as claimed. □

B.1. Proof of Theorem 5.2

Consider the run whose assigned threshold parameter satisfies $\tau \leq \frac{f(V)\cdot\epsilon}{\text{OPT}_{\text{cost}}} < (1+\epsilon)\tau$. Lemma B.1 guarantees that output of this run is an expected $(1-O(\epsilon), O(\epsilon)^{-1})$-bicriteria approximate solution. As explained in 4.3 this run is included in the instances with an index between $\left\lfloor \log_{1+\epsilon}(\frac{f(V)\cdot\epsilon}{|V|\rho}) \right\rfloor$ and $\left\lfloor \log_{1+\epsilon}(f(V)\cdot\epsilon) \right\rfloor$, and Lemma B.4 ensures that G_T of this run satisfies the condition of $f(G_T) \geq (1-\epsilon)f(V)$. Additionally Lemma B.5 ensures that $f(G_T \backslash D)$ of any run with $f(G_T) \geq (1-\epsilon)f(V)$ is an expected $(1-\epsilon)$ approximate of $f(V)$. Therefore, by checking all the instances in the specified range with $f(G_T) \geq (1-\epsilon)f(V)$, and choosing the one with lowest $\textsc{Cost}(G_T \backslash D)$, we are guaranteed to find an expected $(1-O(\epsilon), O(\epsilon)^{-1})$-bicriteria approximate solution for the problem.

C. Query complexity

In this section, we provide an analysis of the query complexity for our algorithm. The proofs in this section follow the framework of (Banihashem et al., 2023a) and are provided for completeness.

Throughout the section, we set $\eta := \frac{1+\epsilon}{\epsilon} n\rho$. We also use the shorthand σ_i and σ_i^{pre} to denote the events $(\, \mathbf{T} \geq i \wedge \mathbf{H}_i = H_i \,)$ and $\left(\, \mathbf{T} \geq i \wedge \mathbf{H}_i^{\text{pre}} = H_i^{\text{pre}} \,\right)$ respectively.

We begin with the following definition.

Definition C.1 (Potential)**.** For any $i \geq 1$ and any element $e \in \widehat{L}_i$, *potential of* e *in level* i, is defined as the number $P(e,i)$ satisfying $\frac{d(e|G_{i-1})}{\tau} \in [(1+\epsilon)^{P(e,i)-1}, (1+\epsilon)^{P(e,i)})$. For elements $e \notin \widehat{L}_i$, we define $P(e,i) = 0$. We define the *potential of level* i as $P_i := \sum_{e\in V} P(e,i) = \sum_{e\in \widehat{L}_i} P(e,i)$ for $i \leq T+1$ and $P_i = 0$ for $i > T+1$.

Lemma C.2. *For any* e *and* i *the following hold:*

1. $P(e,i) \in [1, O(\log_{1+\epsilon}(\eta))]$.
2. $|\hat{L}_i| \leq P_i \leq O(\log_{1+\epsilon}(\eta))|\hat{L}_i|$
3. $P_{T+1} = 0$ *and* $P_i > 0$ *for* $i \leq T$.

Proof. For the first result, note that since $e \in \widehat{L}_i$, given Lemma D.1, we have $e \in \textsc{Filter}(\widehat{L}_{i-1}, G_{i-1}, \tau)$, which implies $d(e|G_{i-1}) \geq \tau$. Therefore, $P(e,i) \geq 1$. On the other hand, based on Line 5, we only add elements that satisfy $d(e) \leq \eta\tau$. By submodularity, this implies that $d(e|G_{i-1}) \leq \eta\tau$, thus proving the claim. The second result follows from the first one since $P(e,i) = 0$ for $e \notin \widehat{L}_i$ and the third result follows from the second one and Lemma D.4. □

Lemma C.3. *For any* $i \in [1,T]$ *and any* $e \in V$*, the following inequalities hold:* $P(e,i) \geq P(e,i+1)$ *and, consequently,* $P_i \geq P_{i+1}$.

Proof. If $e \notin \widehat{L}_{i+1}$, the claim holds trivially because the right-hand side equals zero. Otherwise, since $\widehat{L}_{i+1} \subseteq \widehat{L}_i$ by Lemma D.1, and $e \in \widehat{L}_i$, the claim follows from the fact that $G_i \subseteq G_{i+1}$. □

Lemma C.4. *Assume we are given sets* L_j, G_{j-1} *satisfying* $\textsc{Filter}(L_j, G_{j-1}, \tau) = L_j$*, and we invoke* $\textsc{Reconstruct}(j)$ *obtaining the (random) values* $\mathbf{T}, \mathbf{S}_j, \ldots \mathbf{S}_{\mathbf{T}}, \mathbf{L}_{j+1}, \ldots \mathbf{L}_{\mathbf{T}+1}$*. Then, for each* $i \geq j$*,*

$$\mathbb{E}\left[\, P_i - \mathbf{P}_{i+1} \middle| \mathbf{T} \geq i, \mathbf{H}_i^{pre} = H_i^{pre} \,\right] \geq \Omega(\epsilon) \cdot |B_i|$$

for all H_i^{pre} *such that* $\Pr[\, \mathbf{T} \geq i, \mathbf{H}_i^{pre} = H_i^{pre} \,] > 0$*, where* H_i^{pre} *is defined as in* (2)*.*

Proof. We first observe that since we are considering these values right after we invoke $\textsc{Reconstruct}(j)$, we have $\widehat{L}_i = L_i$. We first give a sketch of the proof. For each $i \geq j$, by Lemma D.7, $\mathbf{m}_i \in M_i^*$ with probability at least $1 - O(\epsilon/n^{10})$. By definition of M_i^*, $\mathbf{m}_i \in M_i^*$ means that in expectation, at least $\epsilon/2$ fraction of the elements in B_i satisfy $P_{i+1}(b,e) \leq P_i(b,e) - 1$. As for the case $\mathbf{m}_i \notin M_i^*$, since this only happens with low probability, we can handle it using the bound $P_{i+1} \leq P_i$.

Formally, using the shorthand σ_i^{pre} instead of $\mathbf{T} \geq i, \mathbf{H}_i^{\text{pre}} = H_i^{\text{pre}}$,

$$\begin{aligned}
&\mathbb{E}\left[P_i - \mathbf{P}_{i+1} \middle| \sigma_i^{\text{pre}} \right] \\
&= \Pr\left[\mathbf{m}_i \in M_i^* | \sigma_i^{\text{pre}} \right] \cdot \mathbb{E}\left[P_i - \mathbf{P}_{i+1} \middle| \sigma_i^{\text{pre}}, \mathbf{m}_i \in M_i^* \right] + \Pr\left[\mathbf{m}_i \notin M_i^* | \sigma_i^{\text{pre}} \right] \cdot \mathbb{E}\left[P_i - \mathbf{P}_{i+1} \middle| \sigma_i^{\text{pre}}, \mathbf{m}_i \notin M_i^* \right] \\
&\geq \Pr\left[\mathbf{m}_i \in M_i^* | \sigma_i^{\text{pre}} \right] \cdot \mathbb{E}\left[P_i - \mathbf{P}_{i+1} \middle| \sigma_i^{\text{pre}}, \mathbf{m}_i \in M_i^* \right]
\end{aligned}$$

where the inequality follows from the fact that $P_i - P_{i+1}$ is always non-negative given Lemma C.3. By Lemma D.7, we can further bound this as

$$\begin{aligned}
\mathbb{E}\left[P_i - \mathbf{P}_{i+1} \middle| \sigma_i^{\text{pre}} \right] &\geq \Pr\left[\mathbf{m}_i \in M_i^* | \sigma_i^{\text{pre}} \right] \cdot \mathbb{E}\left[P_i - \mathbf{P}_{i+1} \middle| \sigma_i^{\text{pre}}, \mathbf{m}_i \in M_i^* \right] \\
&\geq (1 - O(\frac{\epsilon}{n^{10}})) \cdot \mathbb{E}\left[P_i - \mathbf{P}_{i+1} \middle| \sigma_i^{\text{pre}}, \mathbf{m}_i \in M_i^* \right] \\
&\geq \frac{1}{2} \cdot \mathbb{E}\left[P_i - \mathbf{P}_{i+1} \middle| \sigma_i^{\text{pre}}, \mathbf{m}_i \in M_i^* \right] \\
&= \frac{1}{2} \cdot \mathbb{E}_{m_i \sim \mathbf{m}_i | \sigma_i^{\text{pre}}, \mathbf{m}_i \in M_i^*} \left[\mathbb{E}\left[P_i - \mathbf{P}_{i+1} \middle| \sigma_i^{\text{pre}}, \mathbf{m}_i = m_i \right] \right] \qquad (13)
\end{aligned}$$

Finally, as $\widehat{L}_i = L_i$, we observe that

$$\begin{aligned}
P_i - P_{i+1} = \sum_{e \in L_i} P(e,i) - \sum_{e \in L_{i+1}} P(e,i+1) &= \sum_{e \in L_i} P(e,i) - \sum_{e \in L_i} P(e,i+1) \\
&= \sum_{e \in B_i} \left(P(e,i) - P(e,i+1) \right) + \sum_{e \in L_i \setminus B_i} \left(P(e,i) - P(e,i+1) \right) \\
&\overset{(a)}{\geq} \sum_{e \in B_i} \left(P(e,i) - P(e,i+1) \right) \\
&\geq \sum_{e \in B_i} \mathbb{1}\left\{ P(e,i) - P(e,i+1) \geq 1 \right\} \\
&\overset{(b)}{=} \sum_{e \in B_i} \mathbb{1}\left\{ d(e|G_i) < \tau_i \right\} \\
&\overset{(c)}{=} |B_i| - |\textsc{Filter}(B_i, G_i, \tau_i)|
\end{aligned}$$

In the above derivation, inequality (a) follows from Lemma C.3, (b) follows from the fact that if $d(e|G_i) < \tau_i$ then $P(e,i+1) < P(e,i)$ will hold for $e \in B_i$ since $d(e|G_{i-1}) \geq \tau_i$ for these elements. Finally, (c) follows from the definition of FILTER.

Therefore,

$$\begin{aligned}
\mathbb{E}\left[P_i - \mathbf{P}_{i+1} \middle| \sigma_i^{\text{pre}} \right] &\geq \frac{1}{2} \cdot \mathbb{E}_{m_i \sim \mathbf{m}_i | \sigma_i^{\text{pre}}, \mathbf{m}_i \in M_i^*} \left[\mathbb{E}\left[P_i - \mathbf{P}_{i+1} \middle| \sigma_i^{\text{pre}}, \mathbf{m}_i = m_i \right] \right] \\
&\geq \frac{1}{2} \cdot \frac{\epsilon}{2} |B_i|
\end{aligned}$$

where the first inequality follows from (13) and the final inequality follows from Definition 4.1. □

Next, we state the following inequality and refer to (Banihashem et al., 2024) for a proof.

Lemma C.5 (Lemma 25 in (Banihashem et al., 2024)). *Let $\mathbf{X}_0, \mathbf{X}_1, \ldots, \mathbf{X}_n$ be a sequence of integer positive variables such that $\mathbf{X}_i \leq \mathbf{X}_{i-1}$ and*

$$\mathbb{E}\left[\mathbf{X}_i \mid \mathbf{X}_1 = X_1, \ldots, \mathbf{X}_{i-1} = X_{i-1} \right] \leq (1 - \varepsilon') X_{i-1}.$$

Let T denote the first index i such that $X_T = 0$ and assume that $\mathbf{X}_0 = N$ for some fixed integer N. Then $\mathbb{E}\left[\mathbf{T} \right] \leq \frac{\log(N)+1}{\text{poly}(\varepsilon')}$.

Lemma C.6. *Assume we are given sets L_j, G_{j-1} satisfying* FILTER$(L_j, G_{j-1}, \tau) = L_j$ *and* $L_j = \widehat{L}_j$. *Let* $\mathbf{T}, \mathbf{S}_j, \dots \mathbf{S}_T, \mathbf{L}_{j+1}, \dots \mathbf{L}_{T+1}$ *denote the values after invoking* RECONSTRUCT(j)*, the expected value of* $\mathbf{T} - j$ *is bounded by* $\text{poly}(\log(|L_j|), \log(\eta), \frac{1}{\varepsilon})$.

Proof. By Lemma C.2

$$|L_i| \le P_i \le |L_i| \cdot \log_{1+\epsilon}(\eta)$$

Given Lemma C.4, for $i \ge j$,

$$\begin{aligned}
\mathbb{E}\left[\, P_i - \mathbf{P}_{i+1} | \sigma_i^{\text{pre}} \,\right] \ge \Omega(\epsilon) \cdot |B_i| &\overset{(a)}{\ge} \Omega(\frac{\epsilon}{\log_{1+\epsilon}(\eta)\log_{1+\epsilon}(\rho)}) \cdot |L_i| \\
&\overset{(b)}{\ge} \Omega(\frac{\epsilon}{\log^2_{1+\epsilon}(\eta)\log_{1+\epsilon}(\rho)}) \cdot |P_i| \\
&\overset{(c)}{\ge} \Omega(\frac{\epsilon^4}{\log^3(\eta)}) P_i
\end{aligned}$$

where for (a), we have used the fact that $|B_i|$ was the largest bucket, for (b) we have used Lemma C.2, and for (c), we have used the inequality $\log(1+x) \ge \frac{x}{4}$ for $x < 1$. Therefore, for some $\varepsilon' = \Omega(\frac{\epsilon^4}{\log^3(\eta)})$, we have

$$\mathbb{E}\left[\, \mathbf{P}_{i+1} | \sigma_i^{\text{pre}} \,\right] \le (1 - \varepsilon') \cdot P_i,$$

Since the values $\mathbf{P}_1, \dots, \mathbf{P}_i$ are deterministic conditioned on σ_i^{pre}, this further implies

$$\mathbb{E}\left[\, \mathbf{P}_{i+1} | \mathbf{P}_i = P_i, \dots, \mathbf{P}_1 = P_1 \,\right] \le (1 - \varepsilon') \cdot P_i. \tag{14}$$

Formally, since when $P_i \ne 0$, we have $\mathbf{T} \ge i$ given Lemma C.2, and followed by iterated expectation, we will have

$$\begin{aligned}
\mathbb{E}\left[\, \mathbf{P}_{i+1} | \mathbf{P}_i = P_i, \dots, \mathbf{P}_1 = P_1 \,\right] &= \mathbb{E}\left[\, \mathbf{P}_{i+1} | \mathbf{P}_i = P_i, \dots, \mathbf{P}_1 = P_1, \mathbf{T} \ge i \,\right] \\
&= \mathbb{E}\left[\, \mathbb{E}\left[\, \mathbf{P}_{i+1} | \mathbf{P}_i = P_i, \dots, \mathbf{P}_1 = P_1, \mathbf{T} \ge i, \mathbf{H}_i = H_i \,\right] \right] \\
&= \mathbb{E}\left[\, \mathbb{E}\left[\, \mathbf{P}_{i+1} | \mathbf{T} \ge i, \mathbf{H}_i = H_i \,\right] \right] \\
&\le (1 - \varepsilon') P_i
\end{aligned}$$

where the third equality follows from the fact that $\mathbf{P}_1, \dots, \mathbf{P}_i$ is deterministic conditioned on $\{\, \mathbf{T} \ge i, \mathbf{H}_i = H_i \,\}$. If $P_i = 0$, then (14) holds trivially because $P_{i+1} \le P_i = 0$. The claim now follows from Lemma C.5 and the fact that $P_j \le |L_j| O(\log(\eta)/\epsilon)$ □

Lemma C.7. *The expected number of queries made by calling* RECONSTRUCT*(i) is* $|\widehat{L}_i| \cdot \text{poly}(\log(|\widehat{L}_i|), \log(\eta), \frac{1}{\varepsilon})$*, where* $|\widehat{L}_i|$ *refers to the size of* $\widehat{L}_i$ *after the update.*

Proof. By Lemma D.7, and the fact that FILTER and bucketing make at most $O(|\widehat{L}_i|)$ queries, each iteration the while-loop in algorithm 2 makes at most $O((|\widehat{L}_i|) \cdot \text{poly}(\log(\eta), \frac{1}{\epsilon}))$ queries. By Lemma C.6, in expectation, the while-loop is executed at most $\text{poly}(\log(|\widehat{L}_i|), \log(\eta), \frac{1}{\varepsilon})$ times, which proves the claim. □

Lemma C.8. *At any point in the stream, the expected number of levels* $\mathbb{E}\left[\, \mathbf{T} \,\right]$ *is at most* $\text{poly}(\log(n), \log(\eta), \frac{1}{\varepsilon})$.

Proof. We note that this lemma is different from Lemma C.6 because we are claiming that the number of levels is always bounded in expectation, while Lemma C.6 can only bound $\mathbb{E}\left[\, \mathbf{T} - j \,\right]$ right after a call to RECONSTRUCT(j). In particular, this also means that we can no longer assume $\widehat{L}_i = L_i$. The proof follows using a similar technique as Lemma C.6. We first claim that a variant of Lemma C.4 still holds. More formally, we claim that

$$\mathbb{E}\left[\, P_i - \mathbf{P}_{i+1} \middle| \mathbf{T} \ge i, \mathbf{H}_i^{\text{pre}} = H_i^{\text{pre}} \,\right] \ge \Omega(\epsilon) \cdot |B_i| \tag{15}$$

for any $i \geq 1$. The proof follows with the exact same logic as the proof of Lemma C.4. Using the shorthand σ_i^{pre} to denote $\mathbf{T} \geq i, \mathbf{H}_i^{\text{pre}} = H_i^{\text{pre}}$, for all $i \geq 1$,

$$\begin{aligned}
\mathbb{E}\left[P_i - \mathbf{P}_{i+1} \middle| \sigma_i^{\text{pre}} \right] &\geq \Pr\left[\mathbf{m}_i \in M_i^* | \sigma_i^{\text{pre}} \right] \cdot \mathbb{E}\left[P_i - \mathbf{P}_{i+1} \middle| \sigma_i^{\text{pre}}, \mathbf{m}_i \in M_i^* \right] \\
&\overset{(a)}{\geq} (1 - O(\frac{\epsilon}{n^{10}})) \cdot \mathbb{E}\left[P_i - \mathbf{P}_{i+1} \middle| \sigma_i^{\text{pre}}, \mathbf{m}_i \in M_i^* \right] \\
&\geq \frac{1}{2} \cdot \mathbb{E}\left[P_i - \mathbf{P}_{i+1} \middle| \sigma_i^{\text{pre}}, \mathbf{m}_i \in M_i^* \right] \\
&= \frac{1}{2} \cdot \mathbb{E}_{m_i \sim \mathbf{m}_i | \sigma_i^{\text{pre}}, \mathbf{m}_i \in M_i^*} \left[\mathbb{E}\left[P_i - \mathbf{P}_{i+1} \middle| \sigma_i^{\text{pre}}, \mathbf{m}_i = m_i \right] \right] \qquad (16)
\end{aligned}$$

where for (a) we have now used Lemma D.8 instead of D.7.

Next, we observe that

$$\begin{aligned}
P_i - P_{i+1} &= \sum_{e \in \widehat{L}_i} (P(e,i) - P(e,i+1)) \\
&\geq \sum_{e \in B_i \cap \widehat{L}_i} \mathbb{1}\left\{ P(e,i) - P(e,i+1) \geq 1 \right\} \\
&\geq \sum_{e \in B_i \cap \widehat{L}_i} \mathbb{1}\left\{ d(e|G_i) < \tau_i \right\} \\
&= \sum_{e \in B_i} \mathbb{1}\left\{ d(e|G_i) < \tau_i \right\} - \sum_{e \in B_i \setminus \widehat{L}_i} \mathbb{1}\left\{ d(e|G_i) < \tau_i \right\} \\
&\overset{(a)}{\geq} \sum_{e \in B_i} \mathbb{1}\left\{ d(e|G_i) < \tau_i \right\} - \sum_{e \in B_i \cap D} \mathbb{1}\left\{ d(e|G_i) < \tau_i \right\} \\
&\geq \sum_{e \in B_i} \mathbb{1}\left\{ d(e|G_i) < \tau_i \right\} - |B_i \cap D| \\
&= |B_i| - |\textsc{Filter}(B_i, G_i, \tau_i)| - |B_i \cap D| \\
&\overset{(b)}{\geq} |B_i| - |\textsc{Filter}(B_i, G_i, \tau_i)| - \epsilon_{\text{del}} |B_i|
\end{aligned}$$

where for (a), we have used the fact that $L_i \setminus \widehat{L}_i \subseteq D$, and for (b), we have used Lemma D.3. Therefore,

$$\begin{aligned}
\mathbb{E}\left[P_i - \mathbf{P}_{i+1} \middle| \sigma_i^{\text{pre}} \right] + \frac{\epsilon_{\text{del}}}{2} |B_i| &\geq \frac{1}{2} \mathbb{E}_{m_i \sim \mathbf{m}_i | \sigma_i^{\text{pre}}, \mathbf{m}_i \in M_i^*} \left[\mathbb{E}\left[P_i - \mathbf{P}_{i+1} \middle| \sigma_i^{\text{pre}}, \mathbf{m}_i = m_i \right] \right] + \frac{\epsilon_{\text{del}}}{2} |B_i| \\
&\geq \frac{1}{2} \mathbb{E}_{m_i \sim \mathbf{m}_i | \sigma_i^{\text{pre}}, \mathbf{m}_i \in M_i^*} \left[|B_i| - |\textsc{Filter}(B_i, G_i, \tau_i)| \,\middle|\, \sigma_i^{-m}, \mathbf{m}_i = m_i \right] \\
&\geq \frac{1}{2} \cdot \frac{\epsilon}{2} |B_i|
\end{aligned}$$

where the first inequality follows from (16), and the final inequality follows from Lemma D.5, together with Definition 4.1. Since $\epsilon_{\text{del}} \leq \epsilon/16$, it follows that

$$\mathbb{E}\left[P_i - \mathbf{P}_{i+1} \middle| \sigma_i^{\text{pre}} \right] \geq \Omega(\epsilon) \cdot |B_i|.$$

We can conclude from Lemma D.3 that $|\widehat{L}_i| \leq |\overline{L}_i| \leq \frac{3}{2}|L_i| \leq 2|L_i|$. Therefore,

$$\begin{aligned}\mathbb{E}\left[\, P_i - \mathbf{P}_{i+1} | \sigma_i^{\text{pre}} \,\right] &\geq \Omega(\epsilon) \cdot |B_i| \geq \Omega(\frac{\epsilon}{\log_{1+\epsilon}(\eta)\log_{1+\epsilon}(\rho)}) \cdot |L_i| \\ &\overset{(a)}{\geq} \Omega(\frac{\epsilon}{\log_{1+\epsilon}(\eta)\log_{1+\epsilon}(\rho)}) \cdot |\widehat{L}_i| \\ &\overset{(b)}{\geq} \Omega(\frac{\epsilon}{\log^2_{1+\epsilon}(\eta)\log_{1+\epsilon}(\rho)}) \cdot |P_i| \\ &\overset{(c)}{\geq} \Omega(\frac{\epsilon^4}{\log^3(\eta)}) P_i\end{aligned}$$

where for (a), we have used $|\widehat{L}_i| \leq 2|L_i|$, for (b) we have used Lemma C.2, and for (c) we have used the inequality $\log(1+x) \geq \frac{x}{4}$ for $x < 1$ and the assumption $\epsilon \leq 1/10$.

Therefore, for some $\varepsilon' = \Omega(\frac{\epsilon^4}{\log^3(\eta)})$, we have

$$\mathbb{E}\left[\, \mathbf{P}_{i+1} | \sigma_i^{\text{pre}} \,\right] \leq (1-\varepsilon') \cdot P_i,$$

As before, this further implies

$$\mathbb{E}\left[\, \mathbf{P}_{i+1} | \mathbf{P}_i = P_i, \ldots, \mathbf{P}_1 = P_1 \,\right] \leq (1-\varepsilon') \cdot P_i.$$

Formally, if $P_i \neq 0$, we have $\mathbf{T} \geq i$ given Lemma C.2. Therefore, by iterated expectation,

$$\begin{aligned}\mathbb{E}\left[\, \mathbf{P}_{i+1} | \mathbf{P}_i = P_i, \ldots, \mathbf{P}_1 = P_1 \,\right] &= \mathbb{E}\left[\, \mathbf{P}_{i+1} | \mathbf{P}_i = P_i, \ldots, \mathbf{P}_1 = P_1, \mathbf{T} \geq i \,\right] \\ &= \mathbb{E}\left[\, \mathbb{E}\left[\, \mathbf{P}_{i+1} | \mathbf{P}_i = P_i, \ldots, \mathbf{P}_1 = P_1, \mathbf{T} \geq i, \mathbf{H}_i = H_i \,\right]\right] \\ &= \mathbb{E}\left[\, \mathbb{E}\left[\, \mathbf{P}_{i+1} | \mathbf{T} \geq i, \mathbf{H}_i = H_i \,\right]\right] \\ &\leq (1-\varepsilon') P_i\end{aligned}$$

as claimed. If $P_i = 0$, then (14) holds trivially because $P_{i+1} \leq P_i = 0$. Note however that

$$\frac{1}{\varepsilon'} = \text{poly}(\log(\eta), \frac{1}{\epsilon})$$

and

$$P_1 \leq O(\log_{1+\epsilon}(\eta))|\widehat{R}_1|.$$

The claim now follows from Lemma C.5. □

C.1. Proof of Theorem 5.3

We start by bounding the count of "direct" queries, which come from insertions and deletions. Here, we don't count queries made indirectly through RECONSTRUCT. Each insertion or deletion can result in at most $O(\mathbf{T})$ queries, where $\mathbf{T}$ is the number of levels during the update. According to Lemma C.8, this is capped at $\text{poly}(\log(n), \log(\eta), \frac{1}{\epsilon})$ because $|\widehat{L}_1| \leq n$.

Moving on to "indirect" queries made by RECONSTRUCT, we charge the cost of each RECONSTRUCT(i) call to the updates causing it. If RECONSTRUCT(i) is triggered by an insertion, its cost is charged to $\overline{L}_i \backslash L_i$, and if by a deletion, it's charged to $B_i \cap D$. Each time RECONSTRUCT(i) is called for some i, the expected number of queries is $|\widehat{L}_i|\text{poly}(\log(|\widehat{L}_i|), \log(\eta), \frac{1}{\epsilon})$. However, this cost is spread across at least $\frac{|\widehat{L}_i|}{\text{poly}(\log(\rho), \log(\eta), \frac{1}{\varepsilon})}$ updates due to the reconstruction conditions (The lower bound is chosen considering the reconstruction condition of deletion and size of B_i and will clearly also hold if RECONSTRUCT(i) is triggered by an insertion, because in that case $|\overline{L}_i \backslash L_i|$ is at least $\frac{1}{3}|\overline{L}_i| \geq \frac{1}{3}|\widehat{L}_i|$). Hence, the cost of each charge is at most $\text{poly}(\log(\eta), \frac{1}{\varepsilon})$ (note that $|\widehat{L}_i| \leq n$, and η has both ρ and n as factors). Now, since each update can be charged

by RECONSTRUCT(i) only once and only if when the update happens $T > i$ and level i gets affected, we can say each update is charged at most poly$(\log(\eta), \frac{1}{\varepsilon})$ for each of the levels it affects. And as the expected number of levels during the update is at most poly$(\log(\eta), \frac{1}{\varepsilon})$ by Lemma C.8, the claim follows. It's important to note that the random bits used to limit the expectation of $\mathbf{T}$ and the ones used to limit the queries for each reconstruction are separate. Since the value $\mathbf{T}$ is known at the update time, it relies on the random bits used before the update. In contrast, the number of queries for each RECONSTRUCT depends on random bits used after (or at the time of) the update.

D. Invariant proofs

The following two lemmas holds given the conditions in the algorithm for insertion and deletion.

Lemma D.1. *For all* $i \in [T+1]$, $\hat{L}_i = \text{FILTER}(\hat{L}_{i-1}, G_i, \tau)$.

Proof. The lemma holds by construction of L_i in RECONSTRUCT and is preserved by insertion and deletion. given the conditions for adding an element to L_i. □

Lemma D.2. *For all* $i \in [T+1]$, $\overline{L}_i \subseteq \overline{L}_{i-1}$.

Proof. The lemma holds by construction of L_i in RECONSTRUCT. Additionally, it is preserved by insertion and deletion because if L_i is reconstructed, then L_{i+1} is reconstructed as well. □

Lemma D.3. *For all* $i \in [T]$, $|B_i \cap D| \leq \epsilon_{\text{del}}|B_i|$ *and* $|\overline{L}_i| \leq \frac{3}{2}|L_i|$

Proof. The lemma holds by the reconstruction condition for insertion and deletion. □

Lemma D.4. $\hat{L}_{T+1} = \overline{L}_{T+1} = L_{T+1} = \emptyset$ *and* $\hat{L}_i, L_i, \overline{L}_i \neq \emptyset$ *for* $i \in [1, T]$.

Proof. The lemma holds after invoking RECONSTRUCT by definition of T. Additionally, it is preserved by insertion and deletion because if $\hat{L}_i$ for $i \in [T]$ becomes empty or $\overline{L}_{T+1}$ becomes non-empty then RECONSTRUCT is invoked, ensuring that the statement holds again. □

Lemma D.5. *For any* $i \geq 1$*, and any* H_i *such that* $\Pr[\, \mathbf{T} \geq i, \mathbf{H}_i = H_i \,] > 0$*,*

$$\Pr[\, \mathbf{S}_i = S \mid \mathbf{T} \geq i, \mathbf{H}_i = H_i \,] = \frac{1}{|X_i|} \mathbb{1}\{S \in X_i\} \tag{17}$$

where $|X_i|$ *denotes all sequences of length* m_i *in* L_i*.*

Proof. We first prove that (3) holds immediately after a call to RECONSTRUCT.

Claim D.6. *Assume we call* RECONSTRUCT(j) *for some* $j \leq i$ *in a data structure with values* $T^-, L_0^-, \ldots$*, satisfying* $T^- \geq i$*, obtaining the new (random) values* $\mathbf{T}, \mathbf{L}_0, \ldots$*. Then for all sets* S*,*

$$\Pr[\, \mathbf{S}_i = S | \mathbf{T} \geq i, \mathbf{H}_i = H_i \,] = \frac{1}{|X_i|} \cdot \mathbb{1}\{S \in X_i\}.$$

Proof. We observe that before S_i is sampled in the RECONSTRUCT algorithm, all of the values compromising H_i are already determined and will not change after S_i is sampled. Therefore, since the claim holds when S_i is sampled (by construction of S_i), it holds afterwards as well.

□

Next, we prove Lemma D.5 by showing that (3) is preserved after insertions and deletions. At the beginning of the stream, the lemma holds trivially since the event $\mathbf{T} \geq i$ is impossible because of $T = 0 < 1 \leq i$. Assuming the lemma holds before some update, we will show that it holds after the update as well.

Formally, let's consider an update in the form of either inserting or deleting an element v. We denote the values of the data structure before the operation as $\mathbf{T}^-, \mathbf{L}_0^-, \ldots$ and the values afterward as $\mathbf{T}, \mathbf{L}_0, \ldots$. For a fixed i, we will prove that (3) holds.

The proof revolves around analyzing two cases based on whether RECONSTRUCT(i) was triggered by the insertion or deletion operation. In the first case, result follows from Claim D.6. In the second case, we rely on the induction hypothesis which asserted that (3) held for the values $\mathbf{T}^-, \mathbf{L}_0^-, \ldots$.

We will now proceed with a formal proof.

Let $\mathbf{Reset}_i$ be a random variable that takes the value 1 if RECONSTRUCT(i) was called because of the update and 0 otherwise. Define the event σ_i as $\mathbf{T} \geq i \wedge \mathbf{H}_i = H_i$. We need to prove that $\Pr[\,\mathbf{S}_i = S|\sigma_i\,] = \frac{1}{|X_i|} \cdot \mathbb{1}\{S \in X_i\}$. By conditioning on $\mathbf{Reset}_i$, we can express $\Pr[\,\mathbf{S}_i = S|\sigma_i\,]$ as follows:

$$\Pr[\,\mathbf{S}_i = S|\sigma_i\,] = \mathbb{E}_{\text{Reset}_i \sim \mathbf{Reset}_i|\sigma_i}\left[\Pr[\,\mathbf{S}_i = S|\sigma_i, \mathbf{Reset}_i = \text{Reset}_i\,]\right]. \tag{18}$$

Thus, it suffices to prove

$$\Pr[\,\mathbf{S}_i = S|\sigma_i, \mathbf{Reset}_i = \text{Reset}_i\,] = \frac{1}{|X_i|} \cdot \mathbb{1}\{S \in X_i\}, \tag{19}$$

for both $\text{Reset}_i = 0$ and $\text{Reset}_i = 1$. For $\text{Reset}_i = 1$, the claim holds by Claim D.6 since, by definition, $\text{Reset}_i = 1$ implies that RECONSTRUCT(j) was called for some $j \leq i$.

Therefore, we focus on the case of $\text{Reset}_i = 0$. We begin by observing that $\sigma_i, \mathbf{Reset}_i = 0$ implies $\mathbf{T}^- \geq i$. This is because $\mathbf{Reset}_i = 0$ indicates that RECONSTRUCT(j) for any $j \leq i$ was not called. Consequently, if $\mathbf{T}^-$ were strictly less than i, then $\mathbf{T}$ would equal $\mathbf{T}^-$ (as RECONSTRUCT can only be called for values up to $T + 1$ and if RECONSTRUCT is not called, then T does not change). However, this is not possible since $\mathbf{T} \geq i$.

Given that $\mathbf{T}^- \geq i$, we can condition on the value of the *previous history* of level i. More formally, we define the random variable $\mathbf{H}_i^-$ as:

$$\mathbf{H}_i^- := (\overline{\mathbf{L}}_0^-, \overline{\mathbf{L}}_1^-, \ldots, \overline{\mathbf{L}}_i^-, \mathbf{L}_0^-, \mathbf{L}_1^-, \ldots, \mathbf{L}_i^-, \mathbf{S}_1^-, \ldots, \mathbf{S}_{i-1}^-, \mathbf{m}_i^-).$$

By the law of iterated expectation, we express:

$$\begin{aligned}\Pr[\,\mathbf{S}_i = S|\sigma_i, \mathbf{Reset}_i = 0\,] &= \Pr\left[\,\mathbf{S}_i = S|\sigma_i, \mathbf{Reset}_i = 0, \mathbf{T}^- \leq i\,\right] \\ &= \mathbb{E}_{H_i^- \sim \mathbf{H}_i^-|\sigma_i, \mathbf{Reset}_i = 0, \mathbf{T}^- \leq i}\left[\Pr\left[\,\mathbf{S}_i = S|\sigma_i, \mathbf{Reset}_i = 0, \mathbf{T}^- \leq i, \mathbf{H}_i^- = H_i^-\,\right]\right] \end{aligned} \tag{20}$$

where the expectation is taken over all H_i^- with positive probability.

We now observe that conditioned on $\mathbf{T}^- \leq i, \mathbf{H}_i^- = H_i^-$, the value of $\mathbf{Reset}_i$ always equals 0. This is because Reset_i is a function of $(\overline{L}_0, \ldots, \overline{L}_i, L_0, \ldots, L_i, D)$, which is determined by H_i. Note that D is deterministic since it contains all the deleted elements in the stream and is independent of our algorithm. Therefore, since we only consider H_i^- with positive probability, we can drop the conditioning on $\mathbf{Reset}_i = 0$ in the $\Pr\left[\,\mathbf{S}_i = S|\sigma_i, \mathbf{Reset}_i = 0, \mathbf{T}^- \leq i, \mathbf{H}_i^- = H_i^-\,\right]$ term of (20) since it is redundant.

We can similarly drop σ_i. This is because, as $\mathbf{Reset}_i = 0$, the value of $\mathbf{H}_i$ is deterministic conditioned on $\mathbf{H}_i^- = H_i^-$. Notably, the values of $\mathbf{L}_1, \ldots, \mathbf{L}_i$ are going to be $L_1^-, \ldots, L_i^-$. The same can be said for $\mathbf{S}_1, \ldots, \mathbf{S}_{i-1}$ and $\mathbf{m}_i$. As for

$\overline{\mathbf{L}}_1, \ldots, \overline{\mathbf{L}}_i$, even though their value may be different from $\overline{L_1^-}, \ldots, \overline{L_i^-}$, it is *still deterministic* conditioned on $\mathbf{H}_i^- = H_i^-$ as the decision to add elements in Line 7 is based on the values in H_i^- only.

Given the above observation, we can rewrite $\Pr[\mathbf{S}_i = S|\sigma_i, \mathbf{Reset}_i = 0]$ as:

$$\Pr[\mathbf{S}_i = S|\sigma_i, \mathbf{Reset}_i = 0] = \mathbb{E}_{H_i^- \sim \mathbf{H}_i^-|\sigma_i, \mathbf{Reset}_i = 0}\left[\Pr\left[\mathbf{S}_i = S|\mathbf{T}^- \leq i, \mathbf{H}_i^- = H_i^-\right]\right]$$

We can further replace $\mathbf{S}_i = S$ with $\mathbf{S}_i^- = S$ as $\mathbf{Reset}_i = 0$ implies $\mathbf{S}_i = \mathbf{S}_i^-$.

It follows that

$$\begin{aligned}
\Pr[\mathbf{S}_i = S|\sigma_i, \mathbf{Reset}_i = 0] &= \mathbb{E}_{H_i^- \sim \mathbf{H}_i^-|\sigma_i, \mathbf{Reset}_i = 0}\left[\Pr\left[\mathbf{S}_i^- = S|\mathbf{T}^- \leq i, \mathbf{H}_i^- = H_i^-\right]\right] \\
&\overset{(a)}{=} \mathbb{E}_{H_i^- \sim \mathbf{H}_i^-|\sigma_i, \mathbf{Reset}_i = 0}\left[\frac{1}{|X_i^-|} \cdot \mathbb{1}\left\{S \in X_i^-\right\}\right] \\
&\overset{(b)}{=} \mathbb{E}_{H_i^- \sim \mathbf{H}_i^-|\sigma_i, \mathbf{Reset}_i = 0}\left[\frac{1}{|X_i|} \cdot \mathbb{1}\left\{S \in X_i\right\}\right] \\
&= \frac{1}{|X_i|} \cdot \mathbb{1}\left\{S \in X_i\right\}
\end{aligned}$$

where for (a), we have used the induction assumption, and for (b) we have used the fact that $X_i^- = X_i$ because of $\mathbf{Reset}_i = 0$. We have therefore proved (19) for both $\text{Reset}_i = 0$ and $\text{Reset}_i = 1$, which completes the proof given (18). □

Lemma D.7. *Consider a call to* CALCSAMPLESIZE(L', G', τ') *with values satisfying* FILTER$(L', G', \tau') = L'$ *and* $L' \neq \emptyset$. *The number of queries made by* CALCSAMPLESIZE *is bounded by* $O\left(|L'| \cdot \frac{\log(n)}{\epsilon^3}\right)$. *Furthermore, the output is a suitable sample size (Definition 4.1) with probability at least* $1 - O(\frac{\epsilon}{n^{10}})$.

Proof. To bound the number of queries, note that there will be $4\frac{1}{\epsilon^2}\log(\frac{n^{12}}{\epsilon})$ calls to APPLYANDREVERT, and each call makes $|L'|$ queries, implying the first part of the lemma's statement.

Focus on proving that the output is suitable with a probability of at least $1 - O(\frac{\epsilon}{n^{10}})$.

For a number r, define the value $\mathbf{X}(r)$ as explained in Definition 4.1. For the number m' defined in CALCSAMPLESIZE, we want to prove that $\mathbb{E}[\mathbf{X}(r)] \geq 1 - 2\epsilon$ for all $r \in [1, m' - 1]$ and $\mathbb{E}[\mathbf{X}(m')] \leq 1 - \frac{\epsilon}{2}$ with probability at least $1 - O(\frac{\epsilon}{n^{10}})$.

Hoeffding's inequality implies that for any $r \in [1, m']$

$$\Pr\left[|\frac{\sum_{i=1}^t X_i(r)}{t} - \mathbb{E}[\mathbf{X}(r)]| \geq \frac{\epsilon}{2}\right] \leq 2e^{-\frac{t\epsilon^2}{4}} \leq \frac{\epsilon}{n^{12}},$$

where the second inequality follows from the assumption $t = \left\lceil 4\frac{1}{\epsilon^2}\log(\frac{n^{12}}{\epsilon})\right\rceil$. Applying union bound over r implies that

$$\Pr\left[\forall r \in [1, m'] : |\frac{\sum_{i=1}^t X_i(r)}{t} - \mathbb{E}[\mathbf{X}(r)]| \geq \frac{\epsilon}{2}\right] \leq \frac{\epsilon}{n^{10}}. \tag{21}$$

According to Line 5, $\frac{\sum_{i=1}^t X_i(r)}{t} \geq 1 - \epsilon$ for all $r < m'$ and $\frac{\sum_{i=1}^t X_i(r)}{t} \leq 1 - \epsilon$ for $r = m'$. Together with Equation (21), this implies that with probability at least $1 - \frac{\epsilon}{n^{10}}$,

$$\mathbb{E}[\mathbf{X}(r)] \geq 1 - 2\epsilon \text{ for all } j \in [1, m' - 1] \text{ and } \mathbb{E}[\mathbf{X}(m')] \leq 1 - \frac{\epsilon}{2}.$$

This proves that $m' - 1$ returned by CALCSAMPLESIZE is a suitable sample size with probability at least $1 - O(\frac{\epsilon}{n^{10}})$.

□

Lemma D.8. *For any $i \geq 1$ and at any point in the stream, the following inequality holds:*

$$\Pr[\, \mathbf{m}_i \in M_i^* | \mathbf{T} \geq i, \mathbf{H}_i^{pre} = H_i^{pre} \,] \geq 1 - O(\frac{\epsilon}{n^{10}})$$

Proof. We establish the claim through induction on the update stream. Initially, the claim trivially holds as $\mathbf{T} \geq i$ is impossible.

Assuming the lemma's statement holds before an update, we demonstrate that it holds for the new values as well. The superscript $^-$ denotes values before the insertion, e.g., T^- signifies the number of levels before insertion, and T denotes the number of levels after insertion.

Let $\mathbf{Reset}_i$ be a random variable set to 1 if RECONSTRUCT(j) is called for some $j \leq i$ and 0 otherwise. We show that

$$\Pr\left[\, \mathbf{m}_i \in M_i^* | \mathbf{T} \geq i, \mathbf{H}_i^{\text{pre}} = H_i^{\text{pre}}, \mathbf{Reset}_i = \text{Reset}_i \,\right] \geq 1 - O(\frac{\epsilon}{n^{10}})$$

holds for both $\text{Reset}_i = 0$ and $\text{Reset}_i = 1$. For $\text{Reset}_i = 1$, the claim follows from Lemma D.7. As for $\text{Reset}_i = 0$, it implies $\mathbf{T}^- \geq i$ because if $\mathbf{T}^- < i$, then $\mathbf{T}$ would have been equal to $\mathbf{T}^-$ due to $\mathbf{Reset}_i = 0$, signifying that RECONSTRUCT(j) was not invoked for any $j \leq i$.

Let $(H_i^{\text{pre}})^-$ denote the value of pre-count history before the update. By the law of iterated expectation, it suffices to show

$$\Pr\left[\, \mathbf{m}_i \in M_i^* | \mathbf{T} \geq i, \mathbf{T}^- \geq i, \mathbf{H}_i^{\text{pre}} = H_i^{\text{pre}}, (\mathbf{H}_i^{\text{pre}})^- = (H_i^{\text{pre}})^-, \mathbf{Reset}_i = 0 \,\right] \geq 1 - O(\frac{\epsilon}{n^{10}}) \tag{22}$$

for all $(H_i^{\text{pre}})^-$ with positive probability conditioned on $\mathbf{T} \geq i, \mathbf{H}_i^{\text{pre}} = H_i^{\text{pre}}, \mathbf{Reset}_i = 0$. We can drop the $\mathbf{T} \geq i, \mathbf{H}_i^{\text{pre}} = H_i^{\text{pre}}, \mathbf{Reset}_i = 0$ from the condition since they are implied by $(\mathbf{H}_i^{\text{pre}})^- = (H_i^{\text{pre}})^-$. Furthermore, we can replace $\mathbf{m}_i \in M_i^*$ with $\mathbf{m}_i^- \in (M_i^*)^-$, where $(M_i^*)^-$ is the set of suitable sample counts for level i before the update, as determined by $(H_i^{\text{pre}})^-$. This is because both $\mathbf{m}_i$ and M_i^* remain unchanged through the update (note that M_i^* is a function of L_i, G_{i-1}, and $(L_i, G_{i-1}) = (L_i^-, G_{i-1}^-)$). This transformation reduces (22) to the induction hypothesis, concluding the proof. □