Deletion Robust Submodular Maximization over Matroids

Paul Dütting [* 1] **Federico Fusco** [* 2] **Silvio Lattanzi** [* 1] **Ashkan Norouzi-Fard** [* 1] **Morteza Zadimoghaddam** [* 1]

Abstract

Maximizing a monotone submodular function is a fundamental task in machine learning. In this paper we study the deletion robust version of the problem under the classic matroids constraint. Here the goal is to extract a small size summary of the dataset that contains a high value independent set even after an adversary deleted some elements. We present constant-factor approximation algorithms, whose space complexity depends on the rank k of the matroid and the number d of deleted elements. In the centralized setting we present a $(3.582 + O(\varepsilon))$-approximation algorithm with summary size $O(k + \frac{d}{\varepsilon^2} \log \frac{k}{\varepsilon})$. In the streaming setting we provide a $(5.582 + O(\varepsilon))$-approximation algorithm with summary size and memory $O(k + \frac{d}{\varepsilon^2} \log \frac{k}{\varepsilon})$. We complement our theoretical results with an in-depth experimental analysis showing the effectiveness of our algorithms on real-world datasets.

1. Introduction

Submodular maximization is a fundamental problem in machine learning that encompasses a broad range of applications, including active learning (Golovin & Krause, 2011) sparse reconstruction (Bach, 2010; Das & Kempe, 2011; Das et al., 2012), video analysis (Zheng et al., 2014), and data summarization (Lin & Bilmes, 2011; Bairi et al., 2015).

Given a submodular function f, a universe of elements V, and a family $\mathcal{F} \subseteq 2^V$ of feasible subsets of V, the optimization problem consists in finding a set $S \in \mathcal{F}$ that maximizes $f(S)$. A natural choice for $\mathcal{F}$ are capacity constraints (a.k.a. k-uniform matroid constraints) where any subset S of V of size at most k is feasible. Another standard restriction,

which generalizes capacity constraints and naturally comes up in a variety of settings, are matroid constraints. As an example where such more general constraints are needed, consider a movie recommendation application, where, given a large corpus of movies from various genres, we want to come up with a set of recommended videos that contains at most one movie from each genre.

Exact submodular maximization is a NP-hard problem, but efficient algorithms exist that obtain small constant-factor approximation guarantees in both centralized and streaming setting (e.g., Fisher et al., 1978; Călinescu et al., 2011; Chakrabarti & Kale, 2015).

In this work we design algorithms for submodular optimization over matroids that are robust to deletions. A main motivation for considering deletions are privacy and user preferences. For example, users may exert their "right to be forgotten" or may update their preferences and thus exclude some of the data points. For instance, in the earlier movie recommendation example, a user may mark some of the recommended videos as "seen" or "inappropriate," and we may wish to quickly update the list of recommendations.

1.1. The Deletion Robust Approach

Following Mitrovic et al. (2017), we model robustness to deletion as a two phases game against an adversary. In the first phase, the algorithm receives a robustness parameter d and chooses a subset $W \subseteq V$ as summary of the whole dataset V. Concurrently, an adversary selects a subset $D \subseteq V$ with $|D| \leq d$. The adversary may know the algorithm but has no access to its random bits. In the second phase, the adversary reveals D and the algorithm determines a feasible solution from $W \setminus D$. The goal of the algorithm is to be competitive with the optimal solution on $V \setminus D$. Natural performance metrics in this model are the algorithm's approximation guarantee and its space complexity as measured by the size of the set W. We consider this problem in both the centralized and in the streaming setting. The adversary studied is called oblivious, as its choice of the deleted elements D *does not* depend on the realized W, but possibly on the structure of the algorithm (not on its random bits). This is a natural assumption in the applications. Consider the movie recommendation example: the fact that a movie has already been watched by the user or that it is

---

*Equal contribution [1]Google Research, Zürich, Switzerland [2]Department of Computer, Control and Management Engineering "Antonio Ruberti", Sapienza University of Rome, Rome, Italy. Part of this work was done while Federico was an intern at Google Research, hosted by Paul Dütting. Correspondence to: Federico Fusco <fuscof@diag.uniroma1.it>.

deemed inappropriate is independent of the fact that the specific movie has been selected or not in the summary.

In this model, to obtain a constant-factor approximation, the summary size has to be $\Omega(k + d)$, even when f is additive and the constraint is a k-uniform matroid. To see this, consider the case where exactly $k + d$ of the elements have unitary weight and the remaining elements have weight zero. The adversary selects d of the valuable elements to be deleted, but the algorithm does not know which. To be protective against any possible choice of the adversary, the best strategy of the algorithm is to choose W uniformly at random from the elements that carry weight. This leads to an expected weight of the surviving elements of $|W| \cdot k/(k+d)$, while the optimum is k.

Prior work gave deletion robust algorithms for the special case of k-uniform matroids. The state-of-the-art (Kazemi et al., 2018) is a $2 + O(\varepsilon)$ approximation with $O(k + \frac{d \log k}{\varepsilon^2})$ space in the centralized setting, while in the streaming setting the same approximation is achievable at the cost of an extra multiplicative factor of $O(\frac{\log k}{\varepsilon})$ in space complexity.

For general matroids, Mirzasoleiman et al. (2017) give a black-box reduction, which together with the (non-robust) streaming algorithm of Feldman et al. (2021) yields a 3.147-approximation algorithm with space complexity $\tilde{O}(kd)$.[1] The multiplicative factor d is inherent in the construction, and the k is needed in any subroutine, so this approach necessarily yields to a space complexity of $\Omega(dk)$. The main open question from their work is to design a *space-efficient* algorithm for the problem. This question is important in many practical scenarios where datasets are large and space is an important resource.

1.2. Our Results

We present the first constant-factor approximation algorithms for deletion robust submodular maximization subject to general matroid constraints with almost optimal space usage, i.e., our algorithms only use $\tilde{O}(k + d)$ space. More formally, in the centralized setting we present a $(3.582 + O(\varepsilon))$-approximation algorithm with summary size $O(k + \frac{d}{\varepsilon^2} \log \frac{k}{\varepsilon})$. In the streaming setting we provide a $(5.582 + O(\varepsilon))$-approximation algorithm with summary size and memory $O(k + \frac{d}{\varepsilon^2} \log \frac{k}{\varepsilon})$. The constants in the two cases are $2+\beta$ and $4+\beta$, where β is $e/(e-1) \approx 1.582$, i.e., the best-possible approximation guarantee for the standard centralized problem (Călinescu et al., 2011; Feige, 1998). The "price of robustness" is thus just an extra additive 2 or 4 in the approximation factor, depending on the setting. At the same time, up to possibly a logarithmic factor, the memory requirements are tight. Finally, note that the state-of-the-art for (non-robust) streaming submodular maximization with

matroid constraint is a 3.147-approximation (Feldman et al., 2021).

Intuitively, the extra difficulty in obtaining space-efficient robust deletion summaries with general matroid constraints is that the algorithm can only use elements respecting the matroid constraint to replace a good deleted element from a candidate solution. This issue gets amplified when multiple elements need to be replaced. This is in sharp contrast to the special case of k-uniform matroids where all elements can replace any other element.

Both our algorithms start by setting a logarithmic number of value thresholds that span the average contribution of relevant optimum elements, and use these to group together elements with similar marginal value. The candidate solution is constructed using only elements from bundles that are large enough (a factor of $1/\varepsilon$ larger than the number of deletions). Random selection from a large bundle protects/insures the value of the selected solution against adversarial deletions.

In the centralized algorithm, it is possible to sweep through the thresholds in decreasing order. This monotonic iteration helps us design a charging mechanism for high value optimum elements dismissed due to the matroid constraint. We use the matroid structural properties to find an injective mapping from optimum elements rejected by the matroid property to the set of selected elements. Given the monotonic sweeping of thresholds, the marginal value of missed opportunities cannot dominate the values of added elements.

In the streaming setting, elements arrive in an arbitrary order in terms of their membership to various bundles. So we keep adding elements as long as a large enough bundle exists. Addition of elements from lower value bundles might technically prevent us from selecting some high value elements due to the matroid constraint. So when considering a new element, we allow for a swap operation with any of the elements in the solution to maintain feasibility of the matroid constraint. We perform the swap if the marginal value of the new element e is substantially (a constant factor) higher than the marginal value that the element e' we are kicking out of the solution had when it was added to the solution. The constant factor gap between marginal values helps us account for not only e' but also the whole potential chain of elements that e' caused directly or indirectly to be removed in the course of the algorithm.

By testing our algorithms on multiple real-world datasets, we validate that they achieve almost optimal values while storing only a small fraction of the elements. In the settings we tried they typically attain at least $90 - 95\%$ of the value output by state-of-the-art algorithms that know the deletions in advance even though we only keep a few percents of the elements. Our algorithms persevere in achieving high value solutions and maintaining a concise memory footprint even

---

[1] Where the $\tilde{O}$ notation hides logarithmic factors.

in the face of large number of deletions.

Finally, a consideration on the deletion robust parameter d. Our algorithms (just like the algorithms from earlier work) do not need exact knowledge of d; an upper bound on it yields the same approximation, at a cost of larger summary. Also, with some extra work, it is possible to show that our guarantees degrade gracefully when the estimate of d was too conservative and there were actually more deletions. Note that some prior knowledge on d is a reasonable assumption in the applications and that without any prior information about d it is provably impossible to achieve good approximation and memory efficiency.

1.3. Related Work

Robust submodular optimization has been studied for more than a decade (Krause et al., 2008; Orlin et al., 2018). Mirzasoleiman et al. (2017) provide streaming algorithms that process insertions and deletions on the fly with $\tilde{O}(kd)$ memory for matroid constraints. For cardinality and knapsack constraints, Mitrovic et al. (2017), Kazemi et al. (2018) and Avdiukhin et al. (2019) focused on the deletion robust setting we study and designed centralized, streaming, and distributed algorithms with $\tilde{O}(k + d)$ memory. We have employed some of their interesting techniques to design deletion robust algorithms for the matroid setting, e.g., value-based bundling and selection by sampling from large pools of candidates.

In follow-up work, Zhang et al. (2022) design a simple algorithm for the streaming version of the deletion-robust submodular maximization problem subject to a p-matroid constraint. Their algorithm has optimal space usage of $O(k + d)$ and achieves a $4p$ approximation. They handle robustness via an elegant non-uniform sampling technique that avoids the extra $O(\log k)$ due to keeping the different buckets. Due to lack of space, we refer to Appendix A for a thorough literature review.

2. Preliminaries

We consider a set function $f : 2^V \to \mathbb{R}_{\geq 0}$ on a (potentially large) ground set V. Given two sets $X, Y \subseteq V$, the *marginal gain* of X with respect to Y is defined as

$$f(X \mid Y) = f(X \cup Y) - f(Y),$$

which quantifies the change in value of adding X to Y. When X consists of a singleton x, we use the shorthand $f(x|Y)$ instead of $f(\{x\}|Y)$. The function f is called *monotone* if $f(e \mid X) \geq 0$ for each set $X \subseteq V$ and element $e \in V$, and *submodular* if for any two sets X and Y such that $X \subseteq Y \subseteq V$ and any element $e \in V \setminus Y$ we have $f(e \mid X) \geq f(e \mid Y)$. Throughout the paper, we assume that f is given in terms of a value oracle that computes $f(S)$ for given $S \subseteq V$. We also assume that f is *normalized*, i.e., $f(\emptyset) = 0$. We slightly abuse the notation and for a set X and an element e, use $X + e$ to denote $X \cup \{e\}$ and $X - e$ for $X \setminus \{e\}$. A non-empty family of sets $\mathcal{M} \subseteq 2^V$ is called a *matroid* if it satisfies the following properties. *Downward-closedness*: if $A \subseteq B$ and $B \in \mathcal{M}$, then $A \in \mathcal{M}$; *augmentation*: if $A, B \in \mathcal{M}$ with $|A| < |B|$, then there exists $e \in B$ such that $A + e \in \mathcal{M}$. We call a set $A \subseteq 2^V$ *independent*, if $A \in \mathcal{M}$, and *dependent* otherwise. An independent set that is maximal with respect to inclusion is called a *base*; all the bases of a matroid share the same cardinality k, which is referred to as the *rank* of the matroid. Dually to basis, for any $A \notin \mathcal{M}$, we can define a *circuit* $C(A)$ as a minimal dependent subset of A, i.e. $C(A) \notin \mathcal{M}$ such that all its proper subsets are independent.

The deletion robust model consists of two phases. The input of the first phase is the ground set V and a robustness parameter d, while the input of the second phase is an adversarial set of d deleted elements $D \subset V$, along with the outputs of the first phase. The goal is to design an algorithm that constructs a small size summary robust to deletions $W \subseteq V$ in the first phase, and a solution $S \subseteq W \setminus D$ that is independent with respect to matroid $\mathcal{M}$ in the second phase. The difficulty of the problem lies in the fact that the summary W has to be robust against *any* possible choice of set D by an adversary oblivious to the randomness of the algorithm.

For any set of deleted elements D, the optimum solution denoted by $\mathrm{OPT}(V \setminus D)$ is defined as

$$f(\mathrm{OPT}(V \setminus D)) = \operatorname*{argmax}_{R \subseteq V \setminus D, R \in \mathcal{M}} f(R).$$

We say that a two phase algorithm is an α approximation for this problem if for all $D \subset V$ s.t. $|D| \leq d$, it holds that

$$\mathrm{OPT}(V \setminus D) \leq \alpha \cdot \mathbb{E}\left[f(S_D)\right],$$

where S_D is the solution produced in Phase II when set D is deleted and the expectation is with respect to the eventual internal randomization of the algorithm. An important feature of a two phase algorithm is its summary size, i.e., the cardinality of the set W returned by the first phase.

In this paper we also consider the streaming version of the problem where the elements in V are presented in some arbitrary order (Phase I) and at the end of such online phase the algorithm has to output a deletion robust summary W. Finally, the deleted set D is revealed and Phase II on $W \setminus D$ takes place offline. The quality of an algorithm for the streaming problem is not only assessed by its approximation guarantee and summary size, but is also measured in terms of its *memory*, i.e., the cardinality of the buffer in the online phase.

Algorithm 1 Centralized Algorithm Phase I

1: **Input:** Precision ε and deletion parameter d
2: $\Delta \leftarrow (d+1)^{th}$ largest value in $\{f(e) \mid e \in V\}$.
3: $V_d \leftarrow$ elements with the d largest values.
4: $V \leftarrow V \setminus V_d, A \leftarrow \emptyset$.
5: $T = \left\{ (1+\varepsilon)^i \mid \varepsilon \cdot \frac{\Delta}{(1+\varepsilon)k} < (1+\varepsilon)^i \leq \Delta \right\}$
6: **for** $\tau \in T$ in decreasing order **do**
7: $\quad B_\tau \leftarrow \{e \in V \mid A+e \in \mathcal{M}, f(e \mid A) \geq \tau\}$
8: $\quad$ **while** $|B_\tau| \geq \frac{d}{\varepsilon}$ **do**
9: $\quad\quad e \leftarrow$ a random element from B_τ.
10: $\quad\quad V \leftarrow V - e, A \leftarrow A + e$.
11: $\quad\quad B_\tau \leftarrow \{e \in V \mid A+e \in \mathcal{M}, f(e \mid A) \geq \tau\}$
12: $\quad$ **end while**
13: $\quad$ Remove B_τ from V.
14: **end for**
15: $B \leftarrow V_d \cup \bigcup_{\tau \in T} B_\tau$.
16: **return** A, B.

Algorithm 2 Algorithm Phase II

1: **Input:** A and B outputs of phase I, set D of deleted elements and optimization routine ALG
2: $A' \leftarrow A \setminus D, B' \leftarrow B \setminus D$
3: $\tilde{S} \leftarrow \text{ALG}(A' \cup B')$
4: **return** $S \leftarrow \text{argmax}\{f(A'), f(\tilde{S})\}$

3. Centralized Algorithm

In this section, we present a centralized algorithm for the Deletion Robust Submodular Maximization subject to a matroid constraint. To that end, we start by defining some notations. Let $\tilde{e}$ be the $(d+1)$-th element with largest value according to f and Δ its value. Given the precision parameter ε, we define the set of threshold T as follows $T = \left\{ (1+\varepsilon)^i \mid \varepsilon \cdot \frac{\Delta}{(1+\varepsilon)k} < (1+\varepsilon)^i \leq \Delta \right\}$. The first phase of our algorithm constructs the summary in iterations using two main sets A and B. We go over the thresholds in T in decreasing order and update sets A, B, and V. Let τ be the threshold that we are considering, then B_τ contains any element $e \in V$ such that $f(e|A) > \tau$ and $A+e \in \mathcal{M}$. These are the high contribution elements that can be added to A. As long as the size of $B_\tau > d/\varepsilon$, we choose uniformly at random an element from B_τ, add it to A and recompute B_τ. We observe that A is robust to deletions, i.e., when d elements are deleted, the probability of one specific element in A being deleted is intuitively at most ε. Moreover, since each element added to A is drawn from a pool of elements with *similar* marginals, the value of this set after the deletions decreases at most by a factor $(1-\varepsilon)$ in expectation. As soon as the cardinality of B_τ drops below d/ε, we can no more add elements from it directly to A while keeping A robust and feasible. Therefore, we remove these elements from V and save them for Phase II so that they can be used if they are not deleted. During the execution of the algorithm we need to take special care of the top d element with highest f values. To avoid complications, we remove them from the instance before starting the procedure and add them to set B at the end. This does not affect the general logic and only simplifies the presentation and the proofs.

The summary W computed at the end of Phase I is com-

posed by the union of A and B. The second phase of our algorithm uses as routine an arbitrary algorithm ALG for monotone submodular maximization subject to matroid constraint. ALG takes as input a set of elements, function f and matroid $\mathcal{M}$ and returns a β-approximate solution. In this phase we simply use ALG to compute a solution among all the elements in A and B that survived the deletion and return the best among the computed solution and the value of the surviving elements in A.

Theorem 3.1. *For $\varepsilon \in (0, 1/3)$, Centralized Algorithm (Algorithm 1 and Algorithm 2) is in expectation a $(2 + \beta + O(\varepsilon))$-approximation algorithm with summary size $O(k + \frac{d}{\varepsilon^2} \log \frac{k}{\varepsilon})$, where β is the approximation ratio of the auxiliary algorithm ALG.*

Proof. We start by analyzing the size of the summary. Our algorithm returns two sets A and B. Let $A' = A \setminus D$ and $B' = B \setminus D$. Set A is independent set in matroid $\mathcal{M}$ so its size is no more than the rank of $\mathcal{M}$, therefore, $|A| \leq k$. Set B is the union of V_d and $\bigcup_{\tau \in T} B_\tau$. Each set B_τ has at most $\frac{d}{\varepsilon}$ element and there are at most $\frac{2}{\varepsilon} \log \frac{k}{\varepsilon}$ such sets. Set V_d contains d elements. Therefore $|A| + |B| \leq d + k + \frac{2d}{\varepsilon^2} \log \frac{k}{\varepsilon}$. We focus now on bounding the approximation guarantee of our approach. To that end, we fix any set D with $|D| \leq d$ and bound the ratio between the expected value of $f(S)$ and $f(\text{OPT})$ (we omit the dependence on D since it is clear from the context). We define $C = \left\{ e \in V \setminus D \mid f(e \mid A) \geq \varepsilon \cdot \frac{f(\text{OPT})}{k} \right\}$. Intuitively, C contains all the *important* elements: the remaining elements do not increase the value of the submodular function considerably if added to A. Therefore we do not lose much by ignoring them. By submodularity and monotonicity, we have that

$$f(\text{OPT}) \leq \mathbb{E}\left[f(\text{OPT} \cup A)\right]$$
$$\leq \mathbb{E}\left[f(A)\right] + \mathbb{E}\left[f(\text{OPT} \cap C \mid A)\right] + \mathbb{E}\left[f(\text{OPT} \setminus C \mid A)\right]$$
$$\leq \mathbb{E}\left[f(A)\right] + \mathbb{E}\left[f(\text{OPT} \cap C \cap B' \mid A)\right]$$
$$+ \mathbb{E}\left[f((\text{OPT} \cap C) \setminus B' \mid A)\right] + \mathbb{E}\left[f(\text{OPT} \setminus C \mid A)\right] \quad (1)$$

Recall that S is the solution returned by Algorithm 2, we now use its expected value to bound four terms in the last inequality, starting from $\mathbb{E}[f(A)]$. Intuitively, for any element that is part of A the probability of it being deleted is $O(\varepsilon)$, since it is sampled from d/ε *similar* elements uniformly at random and only d elements are deleted. We formalize this

idea in Lemma B.1 and show that

$$\mathbb{E}\left[f(A)\right] \leq (1 + 3\varepsilon)\mathbb{E}\left[f(A')\right] \leq (1 + 3\varepsilon)\mathbb{E}\left[f(S)\right]. \quad (2)$$

For the second term, observe that $\text{OPT} \cap C \cap B' \in \mathcal{M}$ and is contained in the set of elements passed to ALG, so its value is dominated by β times the value of the of $\text{ALG}(A' \cup B')$, all in all:

$$\mathbb{E}\left[f(\text{OPT} \cap C \cap B' \mid A)\right] \leq \beta \cdot \mathbb{E}\left[f(S)\right]. \quad (3)$$

Bounding the third term is more involved. The goal is to show that

$$\mathbb{E}\left[f((\text{OPT} \cap C) \setminus B' \mid A)\right] \leq (1 + \varepsilon)\mathbb{E}\left[f(A)\right]. \quad (4)$$

This statement basically bounds the approximation guarantee of our algorithm in case there are no deletions as well. It shows that our algorithm is almost a 2-approximate in that case. The formal argument is provided in Lemma B.2 and here we only provide an intuitive proof. The elements in $(\text{OPT} \cap C) \setminus B'$ have high contribution with respect to A (definition of set C) but are not in B', therefore they were discarded in Algorithm 1 due to the matroid constraint. It is possible to show that in this situation, for each element $e \in (\text{OPT} \cap C) \setminus B'$ there exists a unique element $e' \in A$ such that $f(e|A) \leq (1 + \varepsilon)f(e'|A_{e'})$ for a suitable subset $A_{e'}$ of A. By a careful telescopic argument, it is possible to finally derive Eq. (4). Note that $\mathbb{E}\left[f(A)\right]$ has already been bounded in Equation (2).

The fourth term refers to at most k elements and can be bounded based on the definition of C: any element e outside of C is such that $f(e|A) \leq \varepsilon \cdot \frac{f(\text{OPT})}{k}$, by submodularity we have then that

$$\mathbb{E}\left[f(\text{OPT} \setminus C \mid A)\right] \leq \sum_{e \in \text{OPT} \setminus C} f(e \mid A) \leq \varepsilon \cdot f(\text{OPT}).$$
$$(5)$$

Finally, the theorem follows by plugging the bounds from last inequality and Eqs. (2) to (5) into Eq. (1) and rearranging the terms. For the sake of completeness it is presented in Appendix B. $\qquad \square$

As a corollary of the previous Theorem, we get an approximation factor 4 if we use as subroutine the greedy algorithm (Fisher et al., 1978) or $2 + \frac{e}{e-1} \leq 3.58$ if we use continuous greedy as in Călinescu et al. (2011). In Appendix B we further improve the dependency on ε and achieve:

Corollary 3.2. *Fix any constant $\delta \in (0, 1)$, then there exists a constant C_δ such that for any $\varepsilon \in (0, \delta)$, in expectation a $(3.582 + \varepsilon \cdot C_\delta)$-approximation algorithm with summary size $O(k + \frac{d}{\varepsilon^2} \log \frac{k}{\varepsilon})$ exists.*

Algorithm 3 Streaming Algorithm Phase I

1: **Input:** Precision ε, deletion parameter d.
2: $T \leftarrow \{(1 + \varepsilon)^i \mid i \in \mathbb{Z}\}$
3: $A \leftarrow \emptyset$, $V_d \leftarrow \emptyset$, $B_\tau \leftarrow \emptyset$ for all $\tau \in T$
4: $V_d \leftarrow$ the first d arriving elements
5: $\Delta \leftarrow 0$, $\tau_{min} \leftarrow 0$
6: **for** every arriving element e' **do**
7:     **if** $f(e') > \min_{x \in V_d} f(x)$ **then**
8:         Add e' to V_d, pop element e with smallest value
9:     **end if**
10:     **else** $e \leftarrow e'$
11:     $\Delta \leftarrow \max\{f(e), \Delta\}$, $\tau_{min} \leftarrow \frac{\varepsilon}{1+\varepsilon} \cdot \frac{\Delta}{k}$
12:     Remove from T all the thresholds smaller than τ_{min} and delete the relative B_τ
13:     **if** $\tau_{min} > f(e \mid A)$ **then** Discard e
14:     Find the largest threshold $\tau \in T$ s.t. $f(e \mid A) \geq \tau$
15:     Add e to B_τ
16:     **while** $\exists \tau \in T$ such that $|B_\tau| \geq \frac{d}{\varepsilon}$ **do**
17:         Remove one element g from B_τ u.a.r.
18:         $w(g) \leftarrow f(g \mid A)$
19:         **if** $A + g \in \mathcal{I}$ **then**
20:             $A \leftarrow A + g$
21:         **else**
22:             $k_g \leftarrow \arg\min\{w(k) \mid k \in C(A + g)\}$
23:             **if** $w(g) > 2 \cdot w(k_g)$ **then**
24:                 $A \leftarrow A + g - k_g$
25:                 **Update** $\{B_\tau\}$ according to A
26:             **end if**
27:         **end if**
28:     **end while**
29: **end for**
30: $B \leftarrow V_d \cup_{\tau \in T} B_\tau$
31: **return** A, B

---

4. Streaming Setting

In this section we present our algorithm for Deletion Robust Submodular Maximization in the streaming setting. In this setting, the elements of V in the first phase arrive on a stream and we want to compute the summary W with limited memory. Our approach consists in carefully mimicking the swapping algorithm (Chakrabarti & Kale, 2015) in a deletion robust fashion; to that end we maintain an independent candidate solution A and buckets B_τ that contain *small* reservoirs of elements from the stream with similar marginal contribution each element with respect to the current solution A. Beyond the B_τ, an extra buffer V_d containing the best d elements seen so far is kept.

Before explaining how these sets are updated in streaming setting, let us elaborate two of the challenges that we face. The first phase of the centralized algorithm, Algorithm 1, recomputes the sets B_τ every time that an element is selected

to be added to set A. This recomputation is very powerful since it ensures that all the elements that can be added to A without violating the matroid constraint and have high marginal gain with respect to A are added to B. Therefore we process them in the order of their contribution. This cannot be achieved in the streaming setting as we cannot keep all the elements and the order that the elements on the stream is not depend on their marginal gain. Moreover the set A is changing overtime and as a results the contribution of the element changes as well. Therefore keeping set B up to date is challenging in streaming setting. Furthermore the changes in A can be problematic for the elements in A as well. Consider the case that we add elements to set A for lower value thresholds based on the elements of the stream. Afterwards there are elements that can be added to higher value threshold. In this scenario, these element cannot be skipped since they are very valuable and any *good* solution needs them. Therefore based on our previous approach we need to add them to A (even if adding them violates the matroid constraint) and remove some elements to keep set A independent. Notice that these removals changes the contribution of the rest of the elements hence some low contribution elements with respect to A can have high contribution after removals.

We start explaining the algorithm by defining the thresholds τ that set B_τ is stored. In the streaming setting we do not have an a priori estimate of OPT, so that we do not know upfront which are the thresholds corresponding to *high quality* elements. This issue is overcome by initially considering *all* the powers of $(1 + \varepsilon)$ and progressively removing the ones too small with respect to Δ, i.e., the $(d + 1)$-st largest value seen so far. Every new element that arrives is first used to update V_d: if its value is smaller then the minimum in V_d then nothing happens, otherwise it is swapped with the smallest elements in it. Then, the value of Δ is updated and all the buckets corresponding to thresholds that are too small are deleted, in order to maintain only a logarithmic number of active buckets. At this point the new element is put into the correct bucket B_τ, if such bucket still exists. Now, new elements are drawn uniformly at random from the buckets B_τ as long as no bucket contains more than d/ε elements. These drawn elements are added to A if and only if it is either feasible to add them directly or they can be *swapped* with a *less important* element in A. To make this notion of importance more precise, each element in the solution is associated with a weight, i.e., its marginal value to A when it was first considered to be added to the solution. An element in A is swapped for a more promising one only if the new one has a weight at least twice as big, while maintaining A independent.

Every time A changes, the buckets B_τ are completely updated so to maintain the invariant that B_τ contains only elements whose marginal value with respect to the current

solution A is within τ and $(1 + \varepsilon) \cdot \tau$. This property is crucial to ensure the deletion robustness: every bucket contains elements that are similar, i.e., whose marginal density with respect to the current solution is at most a multiplicative $(1 + \varepsilon)$ factor away. When the stream terminates, the algorithm returns the candidate solution A and B, containing V_d and the surviving buckets. As in the centralized framework, A and B constitute together the deletion robust summary W to be passed to Algorithm 2. The pseudocode of this algorithm is presented in Algorithm 3.

Theorem 4.1. *For $\varepsilon \in (0, 1/3)$, Streaming Algorithm (Algorithm 3 and Algorithm 2) is in expectation a $(4 + \beta + O(\varepsilon))$-approximation algorithm with summary size and memory $O(k + \frac{d}{\varepsilon^2} \log \frac{k}{\varepsilon})$, where β is the approximation ratio of the auxiliary algorithm* ALG.

Proof. We start by bounding the memory of the algorithm. Sets A and V_d always contains at most k, respectively d, elements. Every time a new element of the stream is considered, all the active B_τ contain at most d/ε elements; furthermore there are always at most $O(\frac{1}{\varepsilon} \log \frac{k}{\varepsilon})$ of them. This is ensured by the invariant that the active thresholds are smaller than Δ (by submodularity) and larger than $\tau_{\min}$. Overall, the memory of the algorithm and the summary size is $O(k + \frac{d}{\varepsilon^2} \log \frac{k}{\varepsilon})$.

As in the analysis of Theorem 3.1, we now fix any set D and study the relative expected performance of our algorithm. To do so we need some notation: let K be the set of all elements removed from the solution A at some point; moreover, for any element g added to the solution, let A_g denote the candidate solution when g was added (possibly containing the element k_g that was swapped with g). Given A and K we can define the set of important elements: $C = \left\{ e \in V \setminus D \mid f(e \mid A \cup K) \geq \varepsilon \cdot \frac{f(\text{OPT})}{k} \right\}$. Arguing similarly to the centralized case, we have that

$$f(\text{OPT} \setminus C \mid A \cup K) \leq \varepsilon \cdot f(\text{OPT}). \qquad (6)$$

All the elements of the stream that were deleted because their marginal with respect to the current solution was smaller than τ_{min} are not in C due to submodularity (this is also why we consider the marginal with respect to $A \cup K$ and not simply A as in the centralized case). We first observe the following two properties hold (proofs in the appendix):
(i) $w(K) \leq w(A) \leq f(A)$ (Lemma C.1)
(ii) $f(A \cup K) \leq w(A \cup K)$ (Lemma C.2)
Moreover, the weight function can be also used to argue about the robustness of our algorithm. More formally, in Lemma C.3 we show that

$$\mathbb{E}\left[w(A)\right] \leq (1 + 3\varepsilon)\mathbb{E}\left[f(A')\right]. \qquad (7)$$

We have all the ingredients to decompose $f(\text{OPT})$ exploit-

ing the definitions of K and C and monotonicity of f:

$$
\begin{aligned}
f(\text{OPT}) \leq & \mathbb{E}\left[f(\text{OPT} \cup A \cup K)\right] \\
= & \mathbb{E}\left[f(A \cup K)\right] \\
& + \mathbb{E}\left[f(\text{OPT} \cap C \cap B' \mid A \cup K)\right] \\
& + \mathbb{E}\left[f((\text{OPT} \cap C) \setminus B' \mid A \cup K)\right] \\
& + \mathbb{E}\left[f(\text{OPT} \setminus C \mid A \cup K)\right] \quad\quad (8)
\end{aligned}
$$

The last term has already been addressed in Equation (6), so let's focus on the remaining three. Start from the first one. Using the properties (i) and (ii) we have

$$
\begin{aligned}
\mathbb{E}\left[f(A \cup K)\right] \leq & \ \mathbb{E}\left[w(A \cup K)\right] && \text{Property (ii)} \\
\leq & \ \mathbb{E}\left[w(A) + w(K)\right] && \text{Definition of } w(\cdot) \\
\leq & \ 2 \cdot \mathbb{E}\left[w(A)\right] && \text{Property (i)} \\
\leq & \ 2(1 + 3\varepsilon)\mathbb{E}\left[f(A')\right] && \text{Equation (7)} \\
\leq & \ 2(1 + 3\varepsilon)\mathbb{E}\left[f(S)\right] && (9)
\end{aligned}
$$

The second term can be bounded similarly to Equation (3), using the assumption on ALG, given that $\text{OPT} \cap C \cap B' \in \mathcal{M}$ and is contained in $A' \cup B'$, where ALG achieves a β approximation:

$$
\mathbb{E}\left[f(\text{OPT} \cap C \cap B' \mid A \cup K)\right] \leq \beta \cdot \mathbb{E}\left[f(S)\right]. \quad (10)
$$

The proof of the last term is more involved and is provided in Lemma C.4.

$$
\mathbb{E}\left[f((\text{OPT} \cap C) \setminus B' \mid A \cup K)\right] \leq 2 \cdot \mathbb{E}\left[f(S)\right]. \quad (11)
$$

Plugging Equations (6) and (9) to (11) into Equation (8) we obtain the claimed approximation. For the sake of completeness it is presented in Appendix C. $\quad\square$

As a corollary of the previous Theorem, we get an approximation factor 6 if we use as subroutine ALG the greedy algorithm (Fisher et al., 1978) or $4 + \frac{e}{e-1} \leq 5.582$ if we use continuous greedy as in Călinescu et al. (2011). Similar to the previous section, in Appendix C we further improve the dependency on ε and achieve:

Corollary 4.2. *Fix any constant $\delta \in (0, 1)$, then there exists a constant G_δ such that for any $\varepsilon \in (0, \delta)$, in expectation a $(3.582 + \varepsilon \cdot G_\delta)$-approximation algorithm with memory and summary size $O(k + \frac{d}{\varepsilon^2} \log \frac{k}{\varepsilon})$ exists.*

5. Experiments

In the experiments we evaluate the performance of our deletion robust centralized and streaming algorithms on real world data. Prior to our work, the only deletion robust algorithm for submodular maximization subject to general matroid constraints were algorithms obtained through the black-box reduction of Mirzasoleiman et al. (2017), which runs $\Theta(d)$ copies of a (non-robust) streaming algorithm as

a subroutine. The natural choice for a practical optimization subroutine is the SWAPPING algorithm, (Algorithm 1 in Chakrabarti & Kale, 2015). Besides implementing this algorithm, that we call ROBUST-SWAPPING in our experiments, we decided to also consider an *omniscient* version of the SWAPPING algorithm, i.e., OMNISCENT-SWAPPING, that knows (and avoids) the elements that are going to be deleted by the adversary. The choice of this latter benchmark has three reasons. First, ROBUST-SWAPPING is designed for a more stringent notion of deletion robustness, and it would have been unfair to use it as only comparison to our two-phases algorithm; second, ROBUST-SWAPPING requires $\Theta(dk)$ memory and summary size, which rapidly approaches n as we consider large D in the experiments (we refer the interested reader to Appendix E for more details). Finally, note that the theoretical approximation guarantees for ROBUST-SWAPPING and OMNISCENT-SWAPPING are identical. Similarly, in the centralized setting we consider an omniscient version of the classic lazy greedy algorithm (Fisher et al., 1978; Minoux, 1978), which we refer to OMNISCENT-GREEDY, that runs lazy greedy on the surviving elements $V \setminus D$. For further details, we refer the reader to Appendix E.

To simulate the adversary, we run the lazy greedy algorithm to find and delete a high value independent set. In this way we make sure that (i) the deleted set has high value, therefore increasing the difficulty of recovering a high value sketch, (ii) the deletions are "evenly" spread according to the matroid constraint, e.g., all the partitions of a partition matroid are equally interested. This does not unfairly affect the omniscient baselines as they know the elements that are going to be deleted in advance. All the experiments were run on a common computer, and running them on any other device *would not* affect the results in any way. In the following, we explain the experimental setup (matroid constraint, submodular function and datasets used) and present the results.

Interactive Personalized Movie Recommendation. Movie recommendation systems are one of the common experiments in the context of submodular maximization (e.g., Mitrovic et al., 2017; Norouzi-Fard et al., 2018; Halabi et al., 2020; Amanatidis et al., 2020; 2021). In this experiment, we have a large collection M of movies from various genres $G_1, G_2, \ldots, G_k$ and we want to design a deletion robust recommendation system that proposes to users one movie from each genre. A natural motivation for deleted elements is given by movies previously watched by the user, or for which the user has a strong negative opinion.[2] The recommendation system has to quickly update its suggestions as the user deletes the movies. We

---

[2]It is clear that this class of deletions cannot be captured by *randomized* deletions and a stronger model is needed.
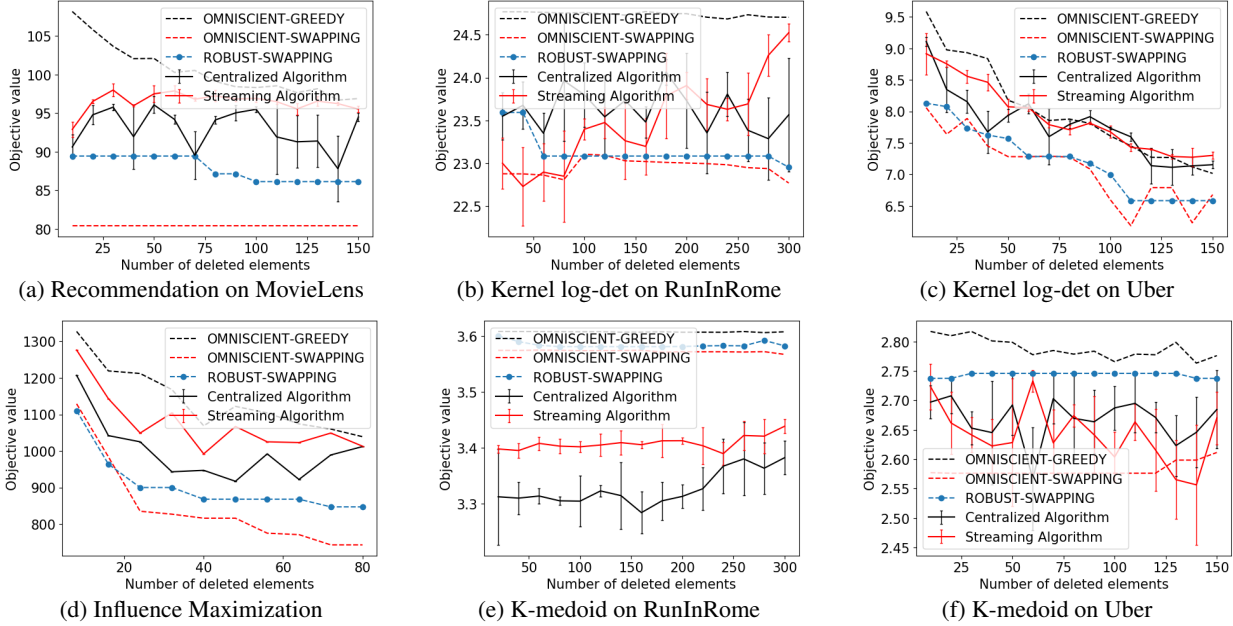
Figure 1. The value of the objective (submodular function f) for Centralized and Streaming algorithms compared to the benchmarks ROBUST-SWAPPING, OMNISCENT-GREEDY and OMNISCENT-SWAPPING with respect to $|D|$. Average and standard deviation over three runs are reported.

use the MovieLens 1M database (Harper & Konstan, 2016), that contains 1000209 ratings for 3900 movies by 6040 users. Based on the ratings, it is possible to associate to each movie m, respectively user u, a feature vector v_m, respectively v_u. More specifically, we complete the users-movies rating matrix and then extract the feature vectors using a singular value decomposition and retaining the first 30 singular values (Troyanskaya et al., 2001). Following the literature (e.g., Mitrovic et al., 2017), we measure the quality of a set of movies S with respect to user u (identified by her feature vector v_u), using the following monotone submodular objective function:

$$f_u(S) = (1 - \alpha) \sum_{s \in S} \langle v_u, v_s \rangle_+ + \alpha \cdot \sum_{m \in M} \max_{s \in S} \langle v_m, v_s \rangle,$$

where $\langle a, b \rangle_+$ denotes the positive part of the scalar product. The first term is linear and sum the *predicted scores* of user u for the movies in S, while the second term has a facility-location structure and is a proxy for how well S *covers* all the movies. Finally, parameter α balances the trade off between the two terms; in our experiments it is set to 0.95. Further details are provided in Appendix E.

Influence Maximization. In this experiment we study deletion robust influence maximization on a social network graph (Norouzi-Fard et al., 2018; Halabi et al., 2020). We use the Facebook dataset from McAuley & Leskovec (2012) that consists of 4039 nodes V and 88234 edges E. As measure of influence we consider the simple dominating

function, that is monotone submodular

$$f(S) = |\{v \in V : \exists s \in S \text{ and } (s, v) \in E\}|.$$

The vertices of the dataset are already partitioned into 11 subsets, the so-called circles, and we consider the problem of selecting at most 8 vertices while choosing at most one vertex from each circle.[3] This way the constraint can be modeled as the intersection of a partition and a uniform matroid, which is still a matroid.

Summarizing Geolocation Data. The third experimental setting that we consider is the problem of tracking geographical coordinates while respecting the privacy concerns. When the sequence of coordinates correspond, for example, to a trajectory, this is inherently a streaming problem, since the coordinates are arriving on a long sequence as the object of interest moves and the goal is to summarizes the path by keeping a short summary. To respect the privacy, some of these coordinates cannot be used in the summary and the user can freely choose them. We model these behaviour by deleting these elements from the solution to ensure privacy. In our experiments, we use two datasets. RunInRome (Fusco, 2022), that contains 8425 positions recorded by running activity in Rome, Italy and a random sample (10351 points) from the Uber pickups dataset (Kaggle, 2020).

To measure the quality of our algorithms, we consider

---

[3] In the original dataset there are elements belonging to more than one circle, for each one of them we pick one of these circles uniformly at random and associate the element only to it.

two different objective functions used in geographical data summarization: the k-medoid and the kernel log-det. Consider the k-medoid function on the metric set (V, d) $L(S) = \frac{1}{|V|} \sum_{v \in V} \min_{e \in S} d(e, v)$. By introducing an auxiliary point $e_0 \in V$ we can turn L into a monotone submodular function (Mirzasoleiman et al., 2013)

$$f(S) = L(e_0) - L(S + e_0).$$

In our experiment we set e_0 to be the first point of each dataset. For the second objective, consider a kernel matrix K that depends on the pair-wise distances of the points, i.e. $K_{i,j} = \exp\{-\frac{d(i,j)^2}{h^2}\}$ where $d(i, j)$ denotes the distance between the ith and the jth point in the dataset and h is some constant. Following Krause & Golovin (2014); Mirzasoleiman et al. (2017); Kazemi et al. (2018), a common monotone submodular objective is

$$f(S) = \log \det(I + \alpha K_{S,S}),$$

where I is the $|S|-$dimensional identity matrix, $K_{S,S}$ is the principal sub-matrix corresponding to the entries in S, and α is a regularization parameter (that we set to 10 in the experiments).

The data points of both datasets are partitioned in geographical areas and the goal is to maximize f using at most two data points from each one of them. This can be modeled with a partition matroid, so our theoretical results apply.

Experimental Results. In Fig. 1 we present the objective values for our algorithms with $\varepsilon = 0.99$ (denoted by Centralized Algorithm and Streaming Algorithm) and the benchmarks: ROBUST-SWAPPING, OMNISCENT-GREEDY and OMNISCENT-SWAPPING. We report the average and standard deviation over three runs. Choosing $\varepsilon = 0.99$ for our algorithms guarantees that the size of the summary is extremely small, i.e., it is at most $4d$, while its average is around $2d$. Smaller values of ε result in solutions with higher objective values but larger summary sizes. The results for a range of ε are presented in Appendix E. Across all datasets, our deletion robust algorithm for the centralized setting typically obtains at least 90-95% of the value of the omniscient benchmark that knows the deletions in advance.

Our deletion robust algorithm for the streaming setting even outperforms its omniscient counterpart in all experiments (up to 20% in some cases), except for Figure 1e, where it is up to 2% lower. Our streaming algorithm is sometimes better than OMNISCENT-SWAPPING because it benefits from the additional memory ($\tilde{O}(k+d)$ vs k), which in some cases outweighs the disadvantage from not knowing the deletions in advance.

As expected, ROBUST-SWAPPING typically behaves similarly to OMNISCENT-SWAPPING, outperforming (i.e. $\approx 5.8\%$, respectively $\approx 3.7\%$) our algorithms only in the last

two plots. Recall, however, that ROBUST-SWAPPING has a considerably larger $O(kd)$ memory footprint. More details in Appendix E.

6. Conclusion and Future Work

We presented the first space-efficient constant-factor approximation algorithms for deletion robust submodular maximization over matroids in both the centralized and the streaming setting. In extensive experiments, we observed that the quality of our algorithms is actually competitive with that of "all-knowing" benchmarks that know the deleted elements in advance. A natural direction for future work is to extend these results and ideas to possibly non-monotone objective, other constraints (e.g., multiple matroids and knapsack), and to consider fully dynamic versions with insertions and deletions.

References

Amanatidis, G., Fusco, F., Lazos, P., Leonardi, S., and Reiffenhäuser, R. Fast adaptive non-monotone submodular maximization subject to a knapsack constraint. In *NeurIPS*, 2020.

Amanatidis, G., Fusco, F., Lazos, P., Leonardi, S., Marchetti-Spaccamela, A., and Reiffenhäuser, R. Submodular maximization subject to a knapsack constraint: Combinatorial algorithms with near-optimal adaptive complexity. In *ICML*, pp. 231–242, 2021.

Avdiukhin, D., Mitrovic, S., Yaroslavtsev, G., and Zhou, S. Adversarially robust submodular maximization under knapsack constraints. In *KDD*, pp. 148–156, 2019.

Bach, F. R. Structured sparsity-inducing norms through submodular functions. In *NIPS*, pp. 118–126, 2010.

Bairi, R., Iyer, R. K., Ramakrishnan, G., and Bilmes, J. A. Summarization of multi-document topic hierarchies using submodular mixtures. In *ACL*, pp. 553–563, 2015.

Barbosa, R. d. P., Ene, A., Nguyen, H. L., and Ward, J. A new framework for distributed submodular maximization. In *FOCS*, pp. 645–654, 2016.

Bogunovic, I., Mitrovic, S., Scarlett, J., and Cevher, V. Robust submodular maximization: A non-uniform partitioning approach. In *ICML*, pp. 508–516, 2017.

Călinescu, G., Chekuri, C., Pál, M., and Vondrák, J. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM J. Comput.*, 40(6):1740–1766, 2011.

Chakrabarti, A. and Kale, S. Submodular maximization meets streaming: matchings, matroids, and more. *Math. Program.*, 154(1-2):225–247, 2015.

Das, A. and Kempe, D. Submodular meets spectral: Greedy algorithms for subset selection, sparse approximation and dictionary selection. In *ICML*, pp. 1057–1064, 2011.

Das, A., Dasgupta, A., and Kumar, R. Selecting diverse features via spectral regularization. In *NIPS*, pp. 1592–1600, 2012.

Ene, A., Nguyen, H. L., and Vladu, A. Submodular maximization with matroid and packing constraints in parallel. In *STOC*, pp. 90–101, 2019.

Epasto, A., Lattanzi, S., Vassilvitskii, S., and Zadimoghaddam, M. Submodular optimization over sliding windows. In *WWW*, pp. 421–430, 2017.

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

Feldman, M., Norouzi-Fard, A., Svensson, O., and Zenklusen, R. Streaming submodular maximization with matroid and matching constraints. *CoRR*, abs/2107.07183, 2021.

Fisher, M. L., Nemhauser, G. L., and Wolsey, L. A. An analysis of approximations for maximizing submodular set functions—ii. In *Polyhedral combinatorics*, pp. 73–87. Springer, 1978.

Fusco, F. RunInRome Dataset, 2022. https://github.com/fedefusco/RunInRome-Dataset.

Golovin, D. and Krause, A. Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *J. Artif. Intell. Res.*, 42:427–486, 2011.

Halabi, M. E., Mitrovic, S., Norouzi-Fard, A., Tardos, J., and Tarnawski, J. Fairness in streaming submodular maximization: Algorithms and hardness. In *NeurIPS*, 2020.

Harper, F. M. and Konstan, J. A. The MovieLens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5 (4):19, 2016.

Kaggle. Uber pickups in New York City, 2020. https://www.kaggle.com/fivethirtyeight/uber-pickups-in-new-york-city.

Kazemi, E., Zadimoghaddam, M., and Karbasi, A. Deletion-robust submodular maximization at scale. *CoRR*, abs/1711.07112, 2017.

Kazemi, E., Zadimoghaddam, M., and Karbasi, A. Scalable deletion-robust submodular maximization: Data summarization with privacy and fairness constraints. In *ICML*, volume 80, pp. 2549–2558, 2018.

Krause, A. and Golovin, D. Submodular function maximization. In *Tractability*, pp. 71–104. Cambridge University Press, 2014.

Krause, A., McMahan, H. B., Guestrin, C., and Gupta, A. Robust submodular observation selection. *J. Mach. Learn. Res.*, 9(12), 2008.

Lattanzi, S., Mitrovic, S., Norouzi-Fard, A., Tarnawski, J., and Zadimoghaddam, M. Fully dynamic algorithm for constrained submodular optimization. In *NeurIPS*, 2020.

Lin, H. and Bilmes, J. A class of submodular functions for document summarization. In *HLT*, pp. 510–520, 2011.

McAuley, J. J. and Leskovec, J. Learning to discover social circles in ego networks. In *NIPS*, pp. 548–556, 2012.

Minoux, M. Accelerated greedy algorithms for maximizing submodular set functions. In *Optimization Techniques*, pp. 234–243. Springer, 1978.

Mirrokni, V. and Zadimoghaddam, M. Randomized composable core-sets for distributed submodular maximization. In *STOC*, pp. 153–162, 2015.

Mirzasoleiman, B., Karbasi, A., Sarkar, R., and Krause, A. Distributed submodular maximization: Identifying representative elements in massive data. In *NIPS*, pp. 2049–2057, 2013.

Mirzasoleiman, B., Karbasi, A., and Krause, A. Deletion-robust submodular maximization: Data summarization with "the Right to be Forgotten". In *ICML*, pp. 2449–2458, 2017.

Mitrovic, S., Bogunovic, I., Norouzi-Fard, A., Tarnawski, J., and Cevher, V. Streaming robust submodular maximization: A partitioned thresholding approach. In *NIPS*, pp. 4557–4566, 2017.

Monemizadeh, M. Dynamic submodular maximization. In *NeurIPS*, 2020.

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 *ICML*, pp. 3826–3835, 2018.

Orlin, J. B., Schulz, A. S., and Udwani, R. Robust monotone submodular function maximization. *Math. Program.*, 172 (1):505–537, 2018.

Staib, M., Wilder, B., and Jegelka, S. Distributionally robust submodular maximization. In *AISTATS*, pp. 506–516, 2019.

Troyanskaya, O. G., Cantor, M. N., Sherlock, G., Brown, P. O., Hastie, T., Tibshirani, R., Botstein, D., and Altman, R. B. Missing value estimation methods for DNA microarrays. *Bioinform.*, 17(6):520–525, 2001.

Varadaraja, A. B. Buyback problem - approximate matroid intersection with cancellation costs. In *ICALP*, volume 6755, pp. 379–390, 2011.

Zhang, G., Tatti, N., and Gionis, A. Optimal deletion-robust coreset for submodular maximization. *CoRR*, abs/2203.01241, 2022.

Zhao, J., Shang, S., Wang, P., Lui, J. C. S., and Zhang, X. Submodular optimization over streams with inhomogeneous decays. In *AAAI*, pp. 5861–5868, 2019.

Zheng, J., Jiang, Z., Chellappa, R., and Phillips, P. J. Submodular attribute selection for action recognition in video. In *NIPS*, pp. 1341–1349, 2014.

A. Extended Related Work

Robust submodular optimization has been studied for more than a decade; in Krause et al. (2008), the authors study robustness from the perspective of multiple agents each with its own submodular valuation function. Their objective is to select a subset that maximizes the minimum among all the agents' valuation functions. In some sense, this maximum minimum objective could be seen as a max-min fair subset selection goal. Another robustness setting that deals with multiple valuation function is distributionally robust submodular optimization Staib et al. (2019) in which we have access to samples from a distribution of valuations functions. These settings are fundamentally different from the robustness setting we study in our paper.

Orlin et al. (2018) looked at robustness of a selected set in the presence of a few deletions. In their model, the algorithm needs to finalize the solution before the adversarial deletions are revealed and the adversary sees the choices of the algorithm. Therefore having at least k deletions reduces the value of any solution to zero. Here, k is the cardinality constraint or the rank of the matroid depending on the setting. This is the most prohibitive deletion robust setting we are aware of in the literature and not surprisingly the positive results of Orlin et al. (2018) and Bogunovic et al. (2017) are mostly useful when we are dealing with a few number of deletions.

Mirzasoleiman et al. (2017) study submodular maximization, and provide a general framework to empower inserting-only algorithms to process deletions on the fly as well as insertions. Their result works on general constraints including matroids. As a result, they provide dynamic algorithms that process a stream of deletions and insertions with an extra multiplicative overhead of d (on both the computation time and memory footprint) compared to insertion-only algorithms. Here d is the overall number of deletions over the course of the algorithm. They propose the elegant idea of running $d+1$ concurrent streaming submodular maximization algorithms where d is the maximum number of deletions. Every element is sent to the first algorithm. If it is not selected, it is sent to the second algorithm. If it is not selected again, it is sent to the third algorithm and so on. With this trick, they maintain the invariant that the solution of one of these algorithms is untouched by the adversarial deletions, and therefore this set of $\tilde{O}(dk)$ elements suffice to achieve robust algorithms for matroid constraints. This approach has the drawback of having per update computation time linearly dependent on d which can be prohibitive for large number of deletions. It also has a total memory of $\tilde{O}(dk)$ which could be suboptimal compared to the lower bound of $\Omega(d+k)$. For closely related dynamic setting with cardinality constraint, Lattanzi et al. (2020) provide a $1/2$ approximation algorithm for k-uniform matroids with per update computation time of poly logarithmic in the total number of updates (length of the stream of updates). This much faster computation time comes at the cost of a potentially much larger memory footprint (up to the whole ground set). Monemizadeh (2020) independently designed an algorithm with similar approximation guarantee and an update time quadratic in the cardinality constraint with a smaller dependence on logarithmic terms.

As argued in the Introduction, a simple lower bound of $\Omega(d+k)$ exists on the memory footprint needed by any robust constant-factor approximation algorithm. The gap between this lower bound and the $O(dk)$ memory footprint in Mirzasoleiman et al. (2017) motivated the follow up works that focused on designing even more memory and computationally efficient algorithms.

Mitrovic et al. (2017) and Kazemi et al. (2018) are the first to study the deletion robust setting we consider in our work. They independently designed submodular maximization algorithms for k-uniform matroids. Mitrovic et al. (2017) proposed a streaming algorithm that achieves constant competitive ratio with memory $O((k+d)\operatorname{polylog}(k))$. Their results extends to the case that the adversary is aware of the summary set W before selecting the set D of deleted elements. On the other hand, Kazemi et al. (2018) design centralized and distributed algorithms with constant factor approximation and $O(k+d\log(k))$ memory, as well as a streaming algorithm with 2 approximation factor and memory footprint similar to their distributed result and an extra $\log(k)$ factor. We have borrowed some of their ideas including bundling elements based on their marginal values and ensuring that prior to deletions, elements are added to the solution only if their are selected uniformly at random from a large enough bundle (pool of candidates).

Subsequently, Avdiukhin et al. (2019) showed how to obtain algorithms for the case of knapsack constraints with similar memory requirements and constant factor approximation. While their approximation guarantee are far from optimal, their notion of robustness is strongest than the one we consider: there the adversary can select the set of deleted elements adaptively with respect to the summary produced by the algorithm.

We aim to achieve the generality of Mirzasoleiman et al. (2017) work by providing streaming algorithms that work for all types of matroid constraints while maintaining almost optimal computation time and space efficiency of Mitrovic et al.

(2017) and Kazemi et al. (2018).

Sliding window setting is another well studied robustness model. In this case, the deletions occur as sweep through the stream of elements and in that sense they occur regularly rather than in an adversarial manner. Every element is deleted exactly W steps after its arrival. So at every moment, the most recent W elements are present and the objective is to select a subset of these present elements. Epasto et al. (2017) design a $1/3$ approximation algorithm for the case of cardinality constraints with a memory independent of the window size W. Zhao et al. (2019) provide algorithms that extend the sliding window algorithm to settings that elements have non-uniform lifespans and leave after arbitrary times.

Prior to deletion robust models and motivated by large scale applications, Mirzasoleiman et al. (2013) designed the distributed greedy algorithm and showed that it achieves provable guarantees for cardinality constraint problem under some assumptions. Mirrokni & Zadimoghaddam (2015) followed their work up by providing core-set frameworks that always achieve constant factor approximation in the distributed setting. Barbosa et al. (2016) showed how to approach the optimal $1 - 1/e$ approximation guarantee by increasing the round complexity of the distributed algorithm. For the case of matroid constraints, Ene et al. (2019) provided distributed algorithms that achieve the $1 - 1/e$ approximation with poly-logarithmic number of distributed rounds. For the streaming setting, Feldman et al. (2021) provided a 0.31 competitive ratio algorithm with $\tilde{O}(k)$ memory with a matroid constraint of rank k.

B. Proofs Omitted in Section 3

In this section, we prove the lemmas that are used in Section 3 and we inherit the notation from that section.

Lemma B.1. *For $\varepsilon \in (0, 1)$, we have*

$$\mathbb{E}\left[f(A)\right] \leq \frac{1+\varepsilon}{1-\varepsilon} \, \mathbb{E}\left[f(A')\right]. \tag{12}$$

If $\varepsilon \in (0, \frac{1}{3})$, we the previous inequality implies that

$$\mathbb{E}\left[f(A)\right] \leq (1+3\varepsilon)\mathbb{E}\left[f(A')\right]. \tag{13}$$

Proof. The proof is similar to that of Lemma 2 in Kazemi et al. (2017). For the sake of the analysis, for each threshold $\tau \in T$, let A_τ denote the set of elements in A that were added during an iteration of the for loop corresponding to threshold τ in Algorithm 1. Moreover, let $n_\tau = |A_\tau|$ and order the elements in $A_\tau = <x_{1,\tau}, x_{2,\tau}, \ldots, x_{n_\tau,\tau}>$ according to the order in which they are added to A_τ. Finally, let $I(\ell, \tau)$ be the indicator random variable corresponding to the element $x_{\ell,\tau}$ not being in D. Notice that the randomness here is with respect to the random draw of the algorithm in Phase I: D is fixed but unknown to the algorithm. The crucial argument is that $x_{\ell,\tau}$ is drawn uniformly at random from a set of cardinality at least d/ε where at most d elements lie in D, hence

$$\mathbb{P}\left(I(\ell, \tau)\right) \geq 1 - \varepsilon. \tag{14}$$

As a starting point, we can decompose the value of A as follows using submodularity:

$$f(A) = \sum_{\tau \in T} \sum_{\ell=1}^{n_\tau} f(x_{\ell,\tau} | \cup_{t>\tau} A_t \cup \{x_{1,\tau}, x_{2,\tau}, \ldots, x_{\ell-1,\tau}\}).$$

Recall now that the elements added in a specific iteration of the for loop share the same marginal up to an $(1+\varepsilon)$ factor, i.e.,

$$\tau \leq f(x_{\ell,\tau} | \cup_{t>\tau} A_t \cup \{x_{1,\tau}, x_{2,\tau}, \ldots, x_{\ell-1,\tau}\}) \leq \tau \cdot (1+\varepsilon), \quad \forall \tau \in T. \tag{15}$$

Summing up those contributions, we have

$$\sum_{\tau \in T} |A_\tau| \cdot \tau \leq f(A) \leq (1+\varepsilon) \sum_{\tau \in T} |A_\tau| \cdot \tau. \tag{16}$$

We now decompose in a similar way the value of $A' = A \setminus D$. We let $I(\ell, i) \cdot X$ to be the set X if the indicator variable is 1

and the empty set otherwise. We also define $A'_\tau = A_\tau \setminus D = A_\tau \cap A'$, we have

$$f(A') = \sum_{\tau \in T} \sum_{\ell=1}^{n_\tau} I(\ell, \tau) \cdot f(x_{\ell,\tau} | \cup_{t > \tau} A_t \cup \{I(1,\tau) \cdot x_{1,\tau}, I(2,\tau) \cdot x_{2,\tau}, \ldots, I(\ell-1,\tau) \cdot x_{\ell-1,\tau}\})$$

$$\geq \sum_{\tau \in T} \sum_{\ell=1}^{n_\tau} I(\ell, \tau) \cdot f(x_{\ell,\tau} | \cup_{t > \tau} A_t \cup \{x_{1,\tau}, x_{2,\tau}, \ldots, x_{\ell-1,\tau}\})$$

$$\geq \sum_{\tau \in T} \sum_{\ell=1}^{n_\tau} I(\ell, \tau) \cdot \tau$$

$$= \sum_{\tau \in T} |A'_\tau| \cdot \tau,$$

where the first inequality follows by submodularity and the second one follows from Eq. (15). We apply the expected value to the previous inequality, which results in

$$\mathbb{E}[f(A')] \geq \sum_{\tau \in T} \mathbb{E}[|A'_\tau|] \cdot \tau \geq (1 - \varepsilon) \sum_{\tau \in T} \mathbb{E}[|A_\tau|] \cdot \tau \geq \frac{1-\varepsilon}{1+\varepsilon} \mathbb{E}[f(A)],$$

where the second inequality follows by linearity of expectation and Eq. (14), while the last one from the right hand side of Eq. (16). The Lemma follows observing that $(1 + 3\varepsilon) \geq \frac{1+\varepsilon}{1-\varepsilon}$, for all $\varepsilon \in (0, \frac{1}{3})$. $\square$

Lemma B.2. *We have,*

$$\mathbb{E}[f((\text{OPT} \cap C) \setminus B' \mid A)] \leq (1 + \varepsilon)\mathbb{E}[f(A)].$$

Proof. Let's order the elements in $\text{OPT} \cap C = \{x_1, x_2, \ldots\}$ according to the order in which they were removed from some B_τ because of feasibility constraint (since they are in $C \setminus B$ it must be the case). Notice that once an element fails the feasibility test and is removed from some B_τ, then it is never considered again. Furthermore, for each such x_i let F_i be the set A when x_i fails the feasibility test. We know that F_i is independent since A is independent during the execution of Algorithm 1; moreover, by definition, $F_i + x_i \notin \mathcal{M}$. By Lemma D.1 there exists an injective function $h : \text{OPT} \cap C \to A$ such that $h(x_i) \in F_i$ for all i. Let A_i denote the set A when the element $h(x_i)$ gets added to it. Notice that $A_i \subseteq F_i$ because $h(x_i)$ has been added to A before x_i. We have

$$f((\text{OPT} \cap C) \setminus B' \mid A) \leq \sum_{x \in (\text{OPT} \cap C) \setminus B'} f(x|A)$$

$$\leq \sum_{x_i \in \text{OPT} \cap C} f(x_i|A_i)$$

$$\leq \sum_{x_i \in \text{OPT} \cap C} (1 + \varepsilon) \cdot f(h(x_i)|A_i)$$

$$\leq (1 + \varepsilon) \cdot f(A),$$

where the first two inequalities follow from submodularity and the fact that $A_i \subseteq A$ for all i. The third inequality uses the fact that if x_i was still feasible to add when $h(x_i)$ was added, then their marginals are at most a $(1 + \varepsilon)$ factor away. The last inequality follows from a telescopic decomposition of A, the monotonicity of f and the fact that h is injective. Applying the expectation to both extremes of the chain of inequalities gives the Lemma. $\square$

Missing part of the Proof of Theorem 3.1. For the sake of completeness we explicit the last calculations of the proof for Theorem 3.1. Plugging Eqs. (2) to (5) into Eq. (1) we obtain the following:

$$f(\text{OPT}) \leq \mathbb{E}[f(A)] + \mathbb{E}[f(\text{OPT} \cap C \cap B' \mid A)] + \mathbb{E}[f((\text{OPT} \cap C) \setminus B' \mid A)] + \mathbb{E}[f(\text{OPT} \setminus C \mid A)]$$

$$\leq (2 + \varepsilon)\mathbb{E}[f(A)] + \beta \cdot \mathbb{E}[f(S)] + \varepsilon \cdot f(\text{OPT}) \qquad \text{(Due to Eqs. (3) to (5))}$$

$$\leq (2 + \varepsilon)(1 + 3\varepsilon)\mathbb{E}[f(A')] + \beta \cdot \mathbb{E}[f(S)] + \varepsilon \cdot f(\text{OPT}) \qquad \text{(Due to Eq. (2))}$$

$$\leq [(2 + \varepsilon)(1 + 3\varepsilon) + \beta] \cdot \mathbb{E}[f(S)] + \varepsilon \cdot f(\text{OPT}) \qquad \text{(Definition of } S\text{)}$$

Rearranging terms, one gets

$$f(\text{OPT}) \leq \frac{(2 + \varepsilon)(1 + 3\varepsilon) + \beta}{1 - \varepsilon} \cdot \mathbb{E}[f(S)] \leq [2 + \beta + (2\beta + 15) \cdot \varepsilon] \cdot \mathbb{E}[f(S)],$$

where in the last inequality we used that $(1-\varepsilon)^{-1} \leq (1+2\varepsilon)$ for all $\varepsilon \in (0, 0.5)$ and that $\frac{(2+\varepsilon)(1+3\varepsilon)}{1-\varepsilon} \leq 2 + 15\varepsilon$ for all $\varepsilon \in (0, \frac{1}{3})$. The theorem follows since as long as β is a constant it holds that $(2\beta + 15) \cdot \varepsilon \in O(\varepsilon)$. $\qquad\square$

Proof of Corollary 3.2. The first step in the proof of Theorem 3.1 where we use that ε is bounded below $1/3$ is when we use Equation (13) from Lemma B.1. If we instead use Equation (12), which holds for any $\varepsilon \in (0, 1)$, and follow the exact same steps, we obtain that

$$f(\text{OPT}) \leq \left[(2+\varepsilon)\frac{1+\varepsilon}{1-\varepsilon} + \beta \right] \cdot \mathbb{E}\left[f(S)\right] + \varepsilon \cdot f(\text{OPT})$$

We use as optimization routine ALG continuous greedy, therefore we can plug in $\beta = \frac{e}{e-1}$ and, rearranging the terms we obtain

$$f(\text{OPT}) \leq \left(\frac{e}{(e-1)(1-\varepsilon)} + \frac{(2+\varepsilon)(1+\varepsilon)}{(1-\varepsilon)^2} \right) \mathbb{E}\left[f(S)\right]$$

$$\leq \left[\frac{e}{e-1} + 2 + \underbrace{\frac{8e - 7 - \delta(2e-1)}{(e-1)(\delta-1)^2}}_{C_\delta} \cdot \varepsilon \right] \mathbb{E}\left[f(S)\right],$$

where the last inequality can be numerically verified and holds for any $\varepsilon \in (0, \delta)$. $\qquad\square$

C. Proofs Omitted in Section 4

In this section we present the proofs omit in Section 4 and we inherit the notations from that section.

Lemma C.1. *We have*

- $w(K) \leq w(A) \leq f(A)$

- $w(A') \leq f(A')$.

Proof. The proof of the first inequality is similar to the one of Lemma 9 in (Chakrabarti & Kale, 2015). Crucially, the weight function w is linear and once an element enters A, its weight is fixed forever as the marginal value it contributed when entering A. During the run of the algorithm, every time an element k_g is removed from A, the weight of A increases by $w(g) - w(k_g)$ by its replacement with some element g. Moreover, $w(k_g) \leq w(g) - w(k_g)$ for every element $k_g \in K$ since $2w(k_g) \leq w(g)$. Summing up over all elements in K it holds that

$$w(K) \leq \sum_{k_g \in K} w(k_g) \leq \sum_{k_g \in K} \left[w(g) - w(k_g) \right] \leq w(A).$$

We show now the second inequality. Let $< a_1, a_2, \ldots a_\ell >$ be the elements in A, sorted by the order in which they were added to A. We have that

$$f(A) = \sum_{i=1}^{\ell} f(a_i | \{a_1, \ldots, a_{i-1}\}) \geq \sum_{i=1}^{\ell} f(a_i | A_{a_i}) = \sum_{i=1}^{\ell} w(a_i) = w(A),$$

where A_{a_i} is the solution set right before a_i is added to A. The inequality follows from submodularity, since $\{a_1, \ldots, a_{i-1}\} \subseteq A_{a_i}$. Similarly, let $I(a)$ the indicator random variable corresponding to $a \notin D$, while $I(a) \cdot a$ is a shorthand for the element a if $I(a) = 1$ and the empty set otherwise.

$$f(A') = \sum_{i=1}^{\ell} I(a_i) \cdot f(a_i | \{I(a_1) \cdot a_1, \ldots, I(a_{i-1}) \cdot a_{i-1}\})$$

$$\geq \sum_{i=1}^{\ell} I(a_i) \cdot f(a_i | \{a_1, \ldots, a_{i-1}\})$$

$$\geq \sum_{i=1}^{\ell} I(a_i) \cdot f(a_i | A_{a_i}) = w(A'). \qquad\square$$

Lemma C.2. *We have*

$$f(A \cup K) \le w(A \cup K).$$

Proof. Let $x_1, x_2, \ldots x_t$ be the elements in $A \cup K$, sorted by the order in which they were added to A. We have that

$$f(A \cup K) = \sum_{i=1}^{t} f(x_i | \{x_1, \ldots, x_{i-1}\}) \le \sum_{i=1}^{t} f(x_i | A_{x_i}) = \sum_{i=1}^{t} w(x_i) = w(A \cup K),$$

where A_{x_i} is the solution set right before x_i is added to A. The inequality follows from submodularity, since $\{x_1, \ldots, x_{i-1}\} \supseteq A_{a_i}$. The reason for the last inclusion is simple: A_{x_i} contains all the elements entered in A before x_i minus those elements that have already been removed from it. $\qquad\square$

Lemma C.3. *For $\varepsilon \in (0, 1)$, we have*

$$\mathbb{E}\left[w(A)\right] \le \frac{1+\varepsilon}{1-\varepsilon} \, \mathbb{E}\left[f(A')\right]. \tag{17}$$

If $\varepsilon \in (0, \frac{1}{3})$, the previous inequality implies that

$$\mathbb{E}\left[w(A)\right] \le (1 + 3\varepsilon)\mathbb{E}\left[f(A')\right]. \tag{18}$$

Proof. Let A_τ be the subset of elements that were added into A coming from B_τ and $A'_\tau = A_\tau \setminus D$. Moreover, let a_t^τ be the tth element added to A_τ (if any). We have the following:

$$
\begin{aligned}
\mathbb{E}\left[|A'_\tau|\right] &= \mathbb{E}\left[\sum_{t=1}^{|A_\tau|} \mathbb{1}(a_t^\tau \notin D)\right] = \sum_{t=1}^{+\infty} \mathbb{E}\left[\mathbb{1}(a_t^\tau \notin D) \cdot \mathbb{1}(|A_\tau| \ge t)\right] \\
&= \sum_{t=1}^{+\infty} \mathbb{E}\left[\mathbb{1}(a_t^\tau \notin D) \big| |A_\tau| \ge t\right] \cdot \mathbb{P}\left(|A_\tau| \ge t\right) \\
&\ge (1-\varepsilon) \sum_{t=1}^{+\infty} \mathbb{P}\left(|A_\tau| \ge t\right) \\
&= (1-\varepsilon)\mathbb{E}\left[|A_\tau|\right].
\end{aligned}
$$

Note that in the third equality we The crucial observation is now that when the algorithm decides to add an element to A from B_τ, then the probability that the element belongs to D is at most ε.

Recall the definition of A_a as the elements in A when a is added, so that $w(a) = f(a|A_a)$. Moreover $I(a)$ is the indicator variable of the event $a \in A'$ given that $a \in A$. We have

$$w(A) = \sum_{\tau \in T} \sum_{a \in A_\tau} w(a), \text{ while } w(A') = \sum_{\tau \in T} \sum_{a \in A_\tau} I(a) \cdot w(a)$$

We know that the weight of an element a coming from B_t is such that $\tau \le w(e) \le (1+\varepsilon) \cdot \tau$. Therefore we can proceed similarly to what we had in Lemma B.1:

$$
\begin{aligned}
\mathbb{E}\left[w(A')\right] &= \mathbb{E}\left[\sum_{\tau \in T} \sum_{a \in A_\tau} I(a) \cdot w(a)\right] = \sum_{\tau \in T} \mathbb{E}\left[\sum_{a \in A_\tau} I(a) \cdot w(a)\right] \ge \sum_{\tau \in T} \tau \cdot \mathbb{E}\left[\sum_{a \in A_\tau} I(a)\right] \\
&= \sum_{\tau \in T} \tau \cdot \mathbb{E}\left[|A'_\tau|\right] \ge (1-\varepsilon) \cdot \sum_{\tau \in T} \tau \cdot \mathbb{E}\left[|A_\tau|\right] \ge \frac{(1-\varepsilon)}{(1+\varepsilon)} \mathbb{E}\left[w(A)\right]
\end{aligned}
$$

The Lemma follows by Lemma C.1 and that for $\varepsilon \in (0, \frac{1}{3})$ it holds that $(1+3\varepsilon)(1-\varepsilon) \ge (1+\varepsilon)$. $\qquad\square$

Lemma C.4. *We have $\mathbb{E}\left[f((\mathrm{OPT} \cap C) \setminus B' \mid A \cup K)\right] \le 2 \cdot \mathbb{E}\left[f(S)\right]$.*

Proof. We prove a more general statement. Let Γ be the set of all the elements that, at some point of the run of the algorithm, were considered in an iteration of line 17. We have the following: for all $X \subseteq \Gamma$, $X \in \mathcal{I}$, the following inequality holds true

$$w(X) \le 2 \cdot w(A).$$

The Lemma follows from Theorem 1 of Varadaraja (2011) (using a single matroid and setting $r = 2$). More in the specific, we can imagine to restrict the stream to consider only the elements in Γ, with the order in which they are considered in line 17. $\qquad\square$

Missing part of the Proof of Theorem 4.1. For the sake of completeness we explicit the last calculations of the proof for Theorem 4.1. Plugging Eqs. (6) and (9) to (11) into Eq. (8) we obtain the following:

$$
\begin{aligned}
f(\mathrm{OPT}) \leq & \mathbb{E}\left[f(A \cup K)\right] + \mathbb{E}\left[f(\mathrm{OPT} \cap C \cap B' \mid A \cup K)\right] \\
& + \mathbb{E}\left[f((\mathrm{OPT} \cap C) \setminus B' \mid A \cup K)\right] + \mathbb{E}\left[f(\mathrm{OPT} \setminus C \mid A \cup K)\right] \\
\leq & \left[2(1 + 3\varepsilon) + \beta + 2\right] \cdot \mathbb{E}\left[f(S)\right] + \varepsilon \cdot f(\mathrm{OPT})
\end{aligned}
$$

Rearranging terms, one gets

$$
f(\mathrm{OPT}) \leq \frac{4 + 6\varepsilon + \beta}{1 - \varepsilon} \cdot \mathbb{E}\left[f(S)\right] \leq \left[4 + \beta + (2\beta + 15) \cdot \varepsilon\right] \cdot \mathbb{E}\left[f(S)\right],
$$

where in the last inequality we used that $(1 - \varepsilon)^{-1} \leq (1 + 2\varepsilon)$ for all $\varepsilon \in (0, 0.5)$ and that $\frac{4 + 6\varepsilon}{1 - \varepsilon} \leq 4 + 15\varepsilon$ for all $\varepsilon \in (0, \frac{1}{3})$. The theorem follows since as long as β is a constant it holds that $(2\beta + 15) \cdot \varepsilon \in O(\varepsilon)$. $\qquad\square$

Proof of Corollary 4.2. The first step in the proof of Theorem 4.1 where we need that ε is bounded below $1/3$ is in when we use Equation (18) from Lemma C.3. If we instead use Equation (17), which holds for any $\varepsilon \in (0, 1)$, and follow the exact same steps, we obtain that

$$
f(\mathrm{OPT}) \leq \left[2\frac{1 + \varepsilon}{(1 - \varepsilon)} + \beta + 2\right] \cdot \mathbb{E}\left[f(S)\right] + \varepsilon \cdot f(\mathrm{OPT})
$$

We use as optimization routine ALG continuous greedy, therefore we can plug in $\beta = \frac{e}{e-1}$ and, rearranging the terms we obtain

$$
\begin{aligned}
f(\mathrm{OPT}) \leq & \left(2\frac{1 + \varepsilon}{(1 - \varepsilon)^2} + \frac{3e - 2}{(1 - \varepsilon)(e - 1)}\right) \mathbb{E}\left[f(S)\right] \\
\leq & \left[\frac{e}{e - 1} + 4 + \underbrace{\frac{9e - 8 - \delta(5e - 4)}{(e - 1)(1 - \delta)^2}}_{G_\delta} \cdot \varepsilon\right] \mathbb{E}\left[f(S)\right],
\end{aligned}
$$

where the last inequality can be numerically verified and holds for any $\varepsilon \in (0, \delta)$. $\qquad\square$

D. Combinatorial Properties of Matroids

In this section we focus on showing combinatorial properties of matroids that are used in our analysis. Similar properties have been used in this line of research. We start by stating the main result of this section. To this end, consider a matroid $\mathcal{M} = (E, \mathcal{I})$, where E is a finite set (called the *ground set*) and $\mathcal{I}$ is a family of subsets of E (called the *independent sets*).

Lemma D.1. *Consider a matroid $\mathcal{M} = (E, \mathcal{I})$ and two sets $F \subseteq E$, and $G \in \mathcal{I}$. Suppose that for all $x \in G \setminus F$ there exist a set $F_x \subseteq F, F_x \in \mathcal{I}$ such that $F_x + x \notin \mathcal{I}$. Then there exists a mapping $h : G \setminus F \to F$ such that*

- *for all $x \in G \setminus F$, $h(x) \in F_x$, and*

- *for all $x, y \in G \setminus F$ with $x \neq y$, $h(x) \neq h(y)$.*

Intuitively, h as a *semi-matching* that matches all elements in $G \setminus F$ to an element in $\cup_{x \in G \setminus F} F_x$, while some of the elements in $\cup_{x \in G \setminus F} F_x \subseteq F$ can remain unmatched.

We use Hall's Marriage Theorem to prove the lemma. The combinatorial version of this theorem concerns set systems and the existence of a transversal (a.k.a. system of distinct representatives). Formally, let $\mathcal{S}$ be a family of finite subsets of a base set X ($\mathcal{S}$ may contain the same set multiple times). A *transversal* is an injective function $f : \mathcal{S} \to X$ such that $f(S) \in S$ for every set $S \in \mathcal{S}$. In other words, f selects one representative from each set in S in such a way that no two of these representatives are equal.

Definition D.2. *A family of sets $\mathcal{S}$ satisfies the* marriage condition *if for each subfamily $\mathcal{W} \subseteq \mathcal{S}$,*

$$|\mathcal{W}| \le \left| \bigcup_{F \in \mathcal{W}} F \right|.$$

Theorem D.3 (Hall (1935)). *A family of sets $\mathcal{S}$ has a transversal if and only if $\mathcal{S}$ satisfies the marriage condition.*

Let us recall two well-known properties of matriods.

Definition D.4 (restriction). *Given a matroid $\mathcal{M} = (E, \mathcal{I})$ and a set $S \subseteq E$ the* restriction *of $\mathcal{M}$ to S, denoted by $\mathcal{M} \mid S$, is matroid $\mathcal{M}' = (S, \mathcal{I}')$ where $\mathcal{I}' = \{T \subseteq S \mid T \in \mathcal{I}\}$.*

Definition D.5 (contraction). *Given a matroid $\mathcal{M} = (E, \mathcal{I})$ and a set $S \subseteq E$ the* contraction *of $\mathcal{M}$ by S, written $\mathcal{M}/S$, is the matroid M' on $E \setminus S$ with rank function $r_{\mathcal{M}'}(T) = r_{\mathcal{M}}(T \cup S) - r_{\mathcal{M}}(S)$.*

Specifically, for $S \in \mathcal{I}$, the restriction of $\mathcal{M}$ by S is the matroid $\mathcal{M}' = (E \setminus S, \mathcal{I}')$ where $\mathcal{I}' = \{T \subseteq E \setminus S \mid T \cup S \in \mathcal{I}\}$.

We are ready to present the proof of the main lemma of this section.

Proof of Lemma D.1. Let V be any subset of $G \setminus F$. Define $F_V = \bigcup_{x \in V} F_x$. We want to show that

$$|F_V| = |\bigcup_{x \in V} F_x| \ge |V|.$$

In order to do that we first show that $\mathrm{cl}(V \cup F_V) = \mathrm{cl}(F_V)$. Recall that $\mathrm{cl}(.)$ denotes the *closure* (or *span*) of a set.

We have that $V \subseteq \mathrm{cl}(F_V)$, in fact, for each element $x \in V$ there exists a subset $F_x \subseteq F_V$ such that $F_x + x \notin \mathcal{I}$ and therefore $x \in \mathrm{cl}(F_x)$, which implies, by monotonicity of the closure with respect to the inclusion, that $x \in \mathrm{cl}(F_V)$. We also know that $F_V \subseteq \mathrm{cl}(F_V)$, hence $F_V \cup V \subseteq \mathrm{cl}(F_V)$. Now, if we apply the closure to both sets, we get

$$\mathrm{cl}(F_V \cup V) \subseteq \mathrm{cl}(\mathrm{cl}(F_V)) = \mathrm{cl}(F_V) \subseteq \mathrm{cl}(F_V \cup V),$$

where the equation follows by the well-known properties of closure. This shows that $\mathrm{cl}(F_V \cup V) = \mathrm{cl}(F_V)$ as claimed.

Now let's look at the restriction of the matroid $\mathcal{M}$ to $\mathrm{cl}(F_V \cup V) = \mathrm{cl}(F_V)$. Afterwards, contract this matroid by $(F \cap G) \cap \mathrm{cl}(F_V)$. Call this matroid $\mathcal{M}'$, and denote its rank by r'. We claim that V is independent in this new matroid $\mathcal{M}'$. This is due to, $V \subseteq G \setminus F$ and $V \cup (F \cap G) \subseteq G$ and $G \in \mathcal{I}$.

We thus have

$$r_{\mathcal{M}}(\mathrm{cl}(F_V)) \ge r_{\mathcal{M}}(\mathrm{cl}(F_V) \setminus (F \cap B)) = r' \ge |V|.$$

Finally,

$$|F_V| \ge r_{\mathcal{M}}(F_V) = r_{\mathcal{M}}(\mathrm{cl}(F_V)).$$

Putting these two chains of inequalities together we obtain $|F_V| \ge |V|$ as claimed. The proof now follows by applying Theorem D.3. $\square$

Algorithm 4 Lazy Greedy

1: **Input:** Precision parameter $\varepsilon_0 > 0$
2: $A \leftarrow \emptyset$, $\Delta \leftarrow \max_{e \in V} f(e)$, max-iter $\leftarrow \frac{1}{\varepsilon_0} \log \frac{k}{\varepsilon_0}$
3: Let Q be a priority queue
4: **for** $e \in V$ **do**
5:     Add e to Q with priority $p(e) = f(e)$
6:     cont$(e) \leftarrow 0$
7: **end for**
8: **while** Q is not empty **do**
9:     Pop element e with largest priority $p(e)$ from Q
10:     **if** $A + e \notin \mathcal{M}$ or cont$(e) \geq$ max-iter **then**
11:         Discard e
12:     **else if** $p(e) \leq (1 + \varepsilon_0) \cdot f(e|A)$ **then**
13:         Add e to A
14:     **else**
15:         Add e to Q with priority $p(e) = f(e|A)$
16:         Increase cont(e) by 1
17:     **end if**
18: **end while**
19: **return** A

Algorithm 5 Swapping algorithm

1: $A \leftarrow \emptyset$
2: **for** every arriving element e **do**
3:     $w(e) \leftarrow f(e|A)$
4:     **if** $A + e \in \mathcal{M}$ **then**
5:         $A \leftarrow A + e$
6:     **else**
7:         $k \leftarrow \operatorname{argmin}\{w(k) \mid k \in C(A + e)\}$
8:         **if** $2 \cdot w(k) < w(e)$ **then**
9:             $A \leftarrow A - k + e$
10:         **end if**
11:     **end if**
12: **end for**
13: **return** A

E. More on the Experimental framework

In the experiments, we use as subroutines two famous algorithms: the lazy implementation of the greedy algorithm for monotone submodular maximization subject to a matroid constraint (Fisher et al., 1978; Minoux, 1978) and the streaming algorithm from Chakrabarti & Kale (2015). For the sake of completeness the pseudocodes are reported in Algorithm 4 and Algorithm 5, while their main properties are summarized in Theorem E.1. We use the lazy implementation of greedy because it is way quicker than simple greedy and loses only a small additive constant in the approximation. For our experiments we set this precision parameter of lazy greedy ε_0 to 0.0001.

Theorem E.1 (Folklore). *The lazy implementation of the Greedy algorithm (Algorithm 4) is a deterministic $(2 + O(\varepsilon_0))$-approximation for submodular maximization subject to a matroid constraint. Moreover it terminates after $O(\frac{n}{\varepsilon_0} \log k)$ calls to the value oracle of the submodular function. The swapping algorithm (Algorithm 5) is a a deterministic 4-approximation for streaming submodular maximization subject to a matroid constraint. Its memory is exactly k.*

Similarities between ROBUST-SWAPPING and OMNISCENT-SWAPPING. ROBUST-SWAPPING and OMNISCENT-SWAPPING are both guaranteed to produce a 4 approximation to the best independent set in $V \setminus D$. Besides, their output share the same basic structure: after the deletions, OMNISCENT-SWAPPING outputs the instance of SWAPPING with largest index between those that did not suffered deletions. After all, this is just an instance of SWAPPING that is guaranteed to not have suffered deletions and that has parsed all the elements in $V \setminus D$; this is similar to directly running OMNISCENT-SWAPPING and ignoring the elements in $V \setminus D$, the only difference being the order in which the elements of the stream were considered and the fact that OMNISCENT-SWAPPING might have contained some element (later removed) from D.

Interactive Personalized Movie Recommendation. We consider only movies with at least one rating. The movies in the datasets may belong to more than one genre. Since considering directly as constraint the resulting rule *At most one movie from each genre* would not be a matroid, we associated to each movie a genre distribution vector, then we clustered the movies using k-means++ on those vectors to recover 10 clusters/macro-genres. In the streaming set up we consider a random permutation of the movies, fixed across the experiment. The feature vector v_u of the user u to whom the reccomendation is personalized is drawn uniformly at random from $[0, 1]^{30}$.

Kernel Log-determinant. The kernel matrix is defined as $K_{i,j} = e^{-\left(\frac{d_{i,j}}{h}\right)^2}$, where $d_{i,j}$ denotes the distance between the coordinates of the ith and j^{th} locations while h is a normalization parameter. We set h to be the empirical standard

deviation of the pairwise distances in the RunInRome dataset, while we set $h^2 = 5000$ in the UberDataset. We mention that the identity matrix in the objective function is needed to be sure to take the log of the determinant of a positive definite matrix. The parameter α then tunes the importance of this regularizing perturbation. We set it to be 10 for both the datasets.

RunInRome. In the streaming experiments with this dataset we kept the original order of the positions. The partition of the data points in geographical areas is achieved by dividing the center of Rome in an equally spaced 5×5 grid (in terms of latitude and longitude). The non-empty cells of the grid are 10.

Uber Dataset. The positions considered in the experiments have been obtained by uniform sampling with parameter $1/50$ from the pickups locations of April 2014. In the streaming setting we considered the order of the dataset as sampled. We used the *base* feature of the dataset to partition the positions into 5 subsets.

Memory footprint. To have an improved picture of the quality of approximation-memory footprint trade off between our algorithms and ROBUST-SWAPPING, we report here the summary size plots corresponding to the results in Figure 1.
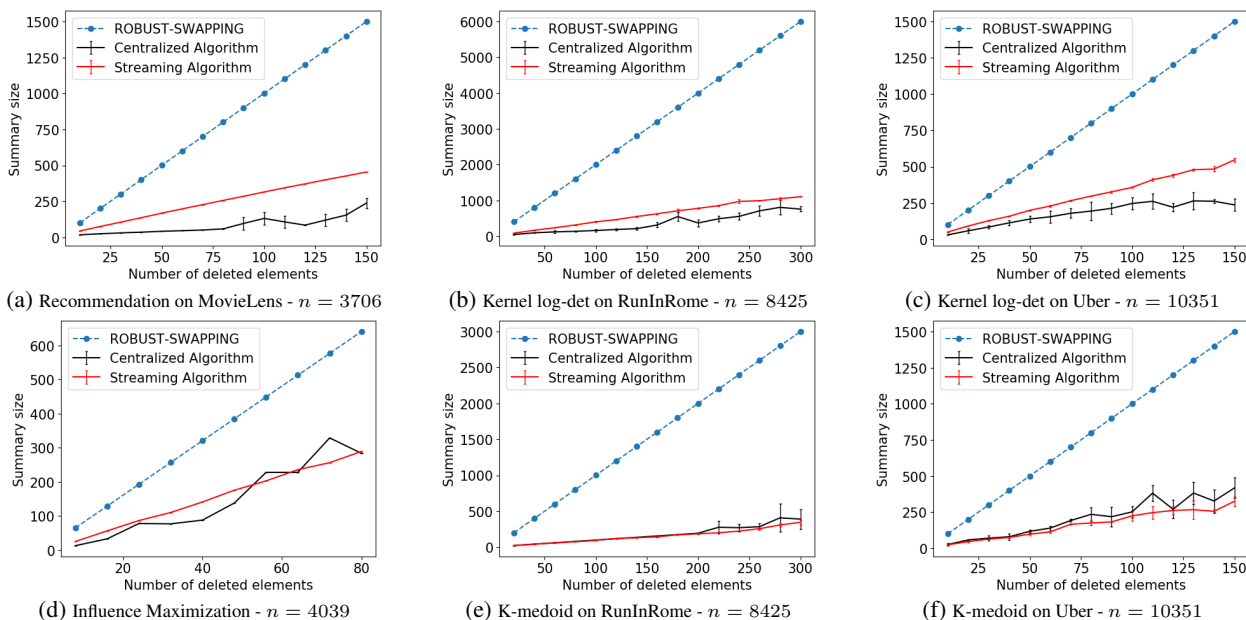

Figure 2. Summary size comparison of the Centralized and Streaming algorithms compared to that of ROBUST-SWAPPING. Average and standard deviation over three runs are reported.

More experiments. Further experiments for different values of the parameter ε are reported in Figures 3 to 8. The results of ROBUST-SWAPPING are not repeated. Average and standard deviation over three runs are reported. We observe that, as suggested by the theoretical results, the value of the objective function decreases as we increase the value of ε. For instance, in Fig. 3 the value of the objective function decreases from 1300 to 1200 when we increase ε from 0.3 to 0.99. We observe that the performances achieved in the experiments are way better than the theoretical worst-case guarantees. One explanation is that in practice the probability that an element added to A gets deleted is way larger than $1 - \varepsilon$: in our analysis we consider the pessimistic case where the adversary manages somehow to *always* delete d out of the d/ϵ elements contained in the bucket we are sampling from. This is however in general quite unlikely: the composition of the buckets may change over time and it is possible there is no large intersection between all the buckets that were considered across all the "sampling" steps of the algorithm.
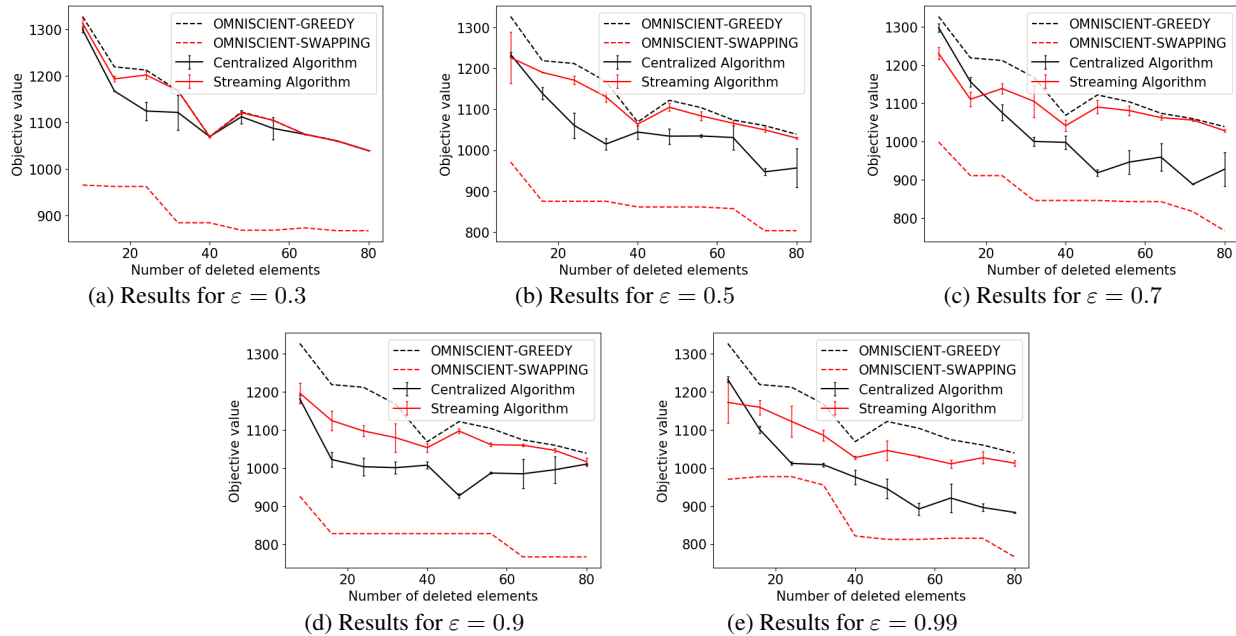
Figure 3. Results of the Influence Maximization experiments on the Facebook dataset for different values of ε. Note how as ε decreases the performances of our algorithms improve, while being always comparable with the benchmarks. The lines corresponding to OMNISCENT-SWAPPING changes in different plots since the random permutation considered changes; this however does not change its qualitative performance.
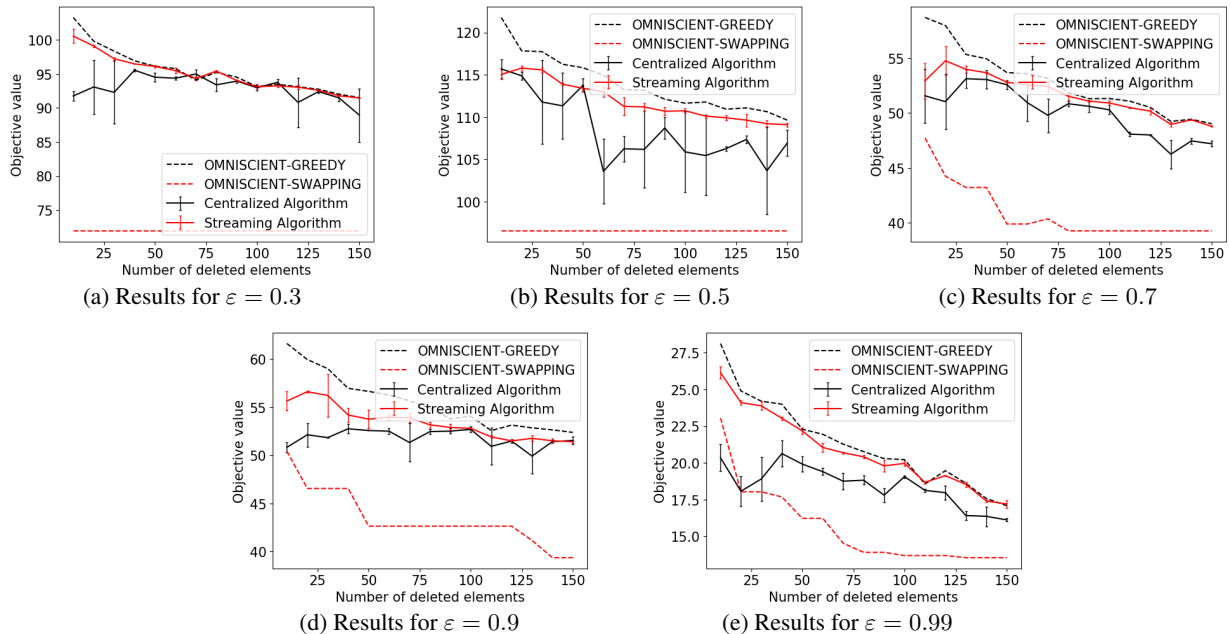

Figure 4. Results of the Interactive Personalized Movie Recommendation on the MovieLens dataset for different values of ε. Note how as ε decreases the performances of our algorithms improve, while being always comparable with the benchmarks. The performances of the benchmarks change for different values of ε because in each experiment a new user feature vector is drawn uniformly at random as well as a new permutation is considered in the streaming setting. The results are however qualitatively comparable.
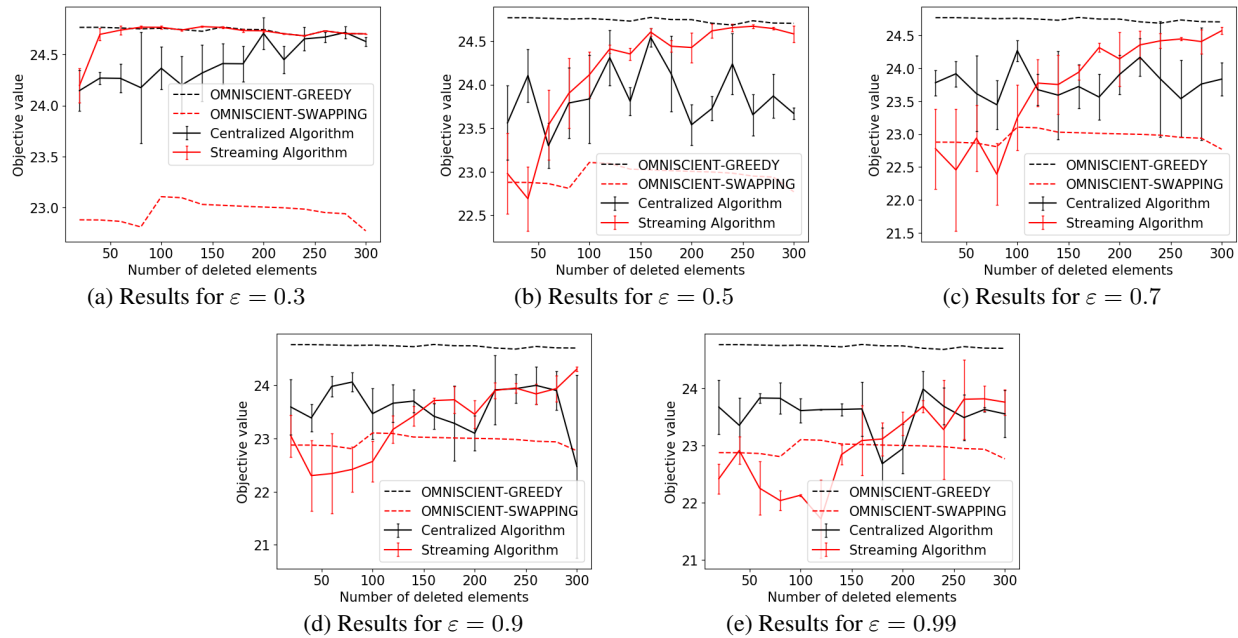
Figure 5. Results of the kernel logdet experiment on the RunInRome dataset for different values of ε. Note how as ε decreases the performances of our algorithms improve, while being always comparable with the benchmarks.
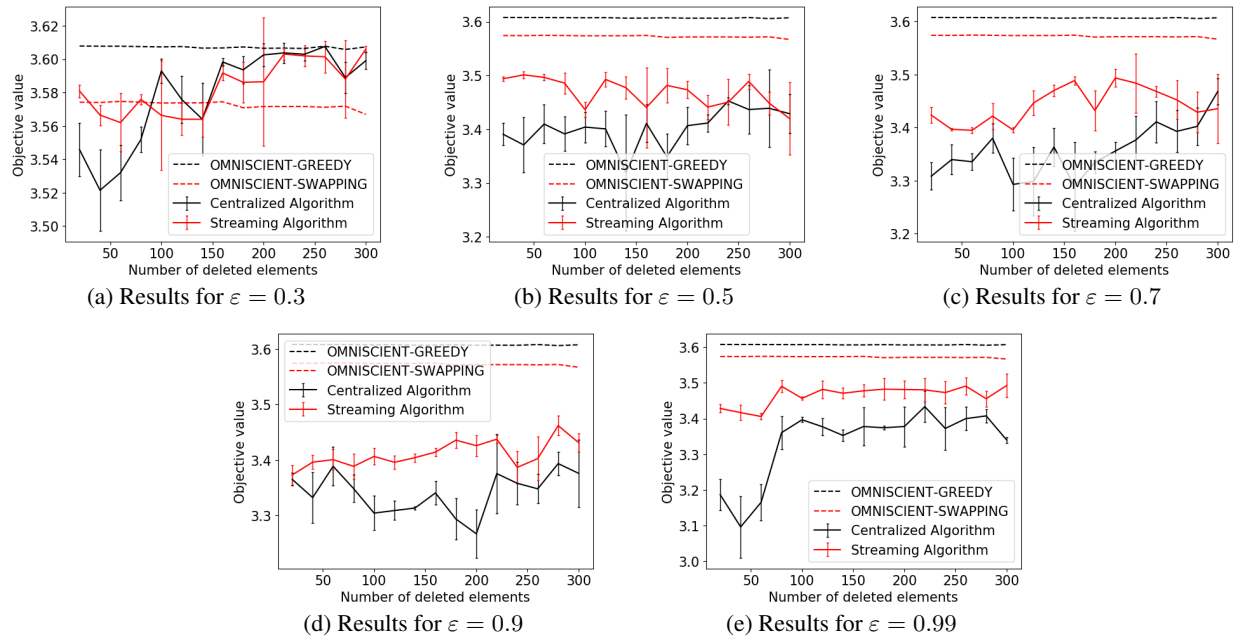

Figure 6. Results of the k-medoid experiment on the RunInRome dataset for different values of ε. Note how as ε decreases the performances of our algorithms improve, starting from a worst case of $\approx 10\%$ for $\varepsilon = 0.99$ and a small number of deletions to a nearly identical performance for $\varepsilon = 0.3$.

(a) Results for $\varepsilon = 0.3$ (b) Results for $\varepsilon = 0.5$ (c) Results for $\varepsilon = 0.7$

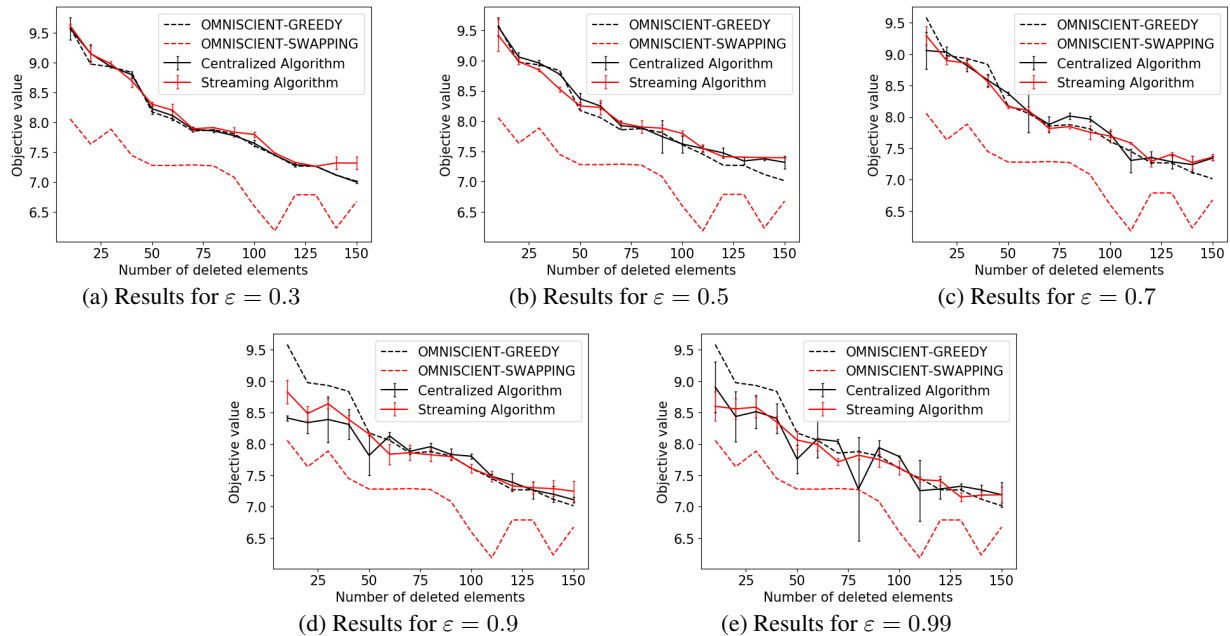(d) Results for $\varepsilon = 0.9$ (e) Results for $\varepsilon = 0.99$

Figure 7. Results of the kernel logdet experiment on the Uber dataset for different values of ε. Note that there is never a remarkable difference between our algorithms and the strongest benchmark OMNISCENT-GREEDY, but as ε decreases also the variability in the output of our algorithms decreases.
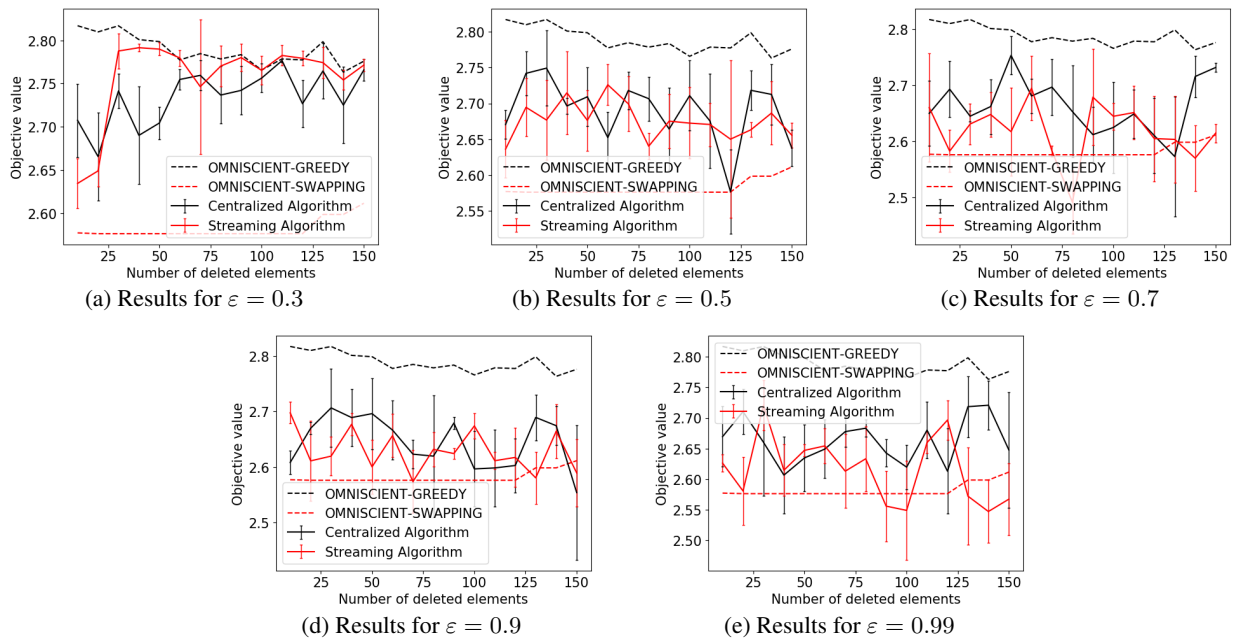

(a) Results for $\varepsilon = 0.3$ (b) Results for $\varepsilon = 0.5$ (c) Results for $\varepsilon = 0.7$

(d) Results for $\varepsilon = 0.9$ (e) Results for $\varepsilon = 0.99$

Figure 8. Results of the k-medoid experiment on the Uber dataset for different values of ε. Note how as ε decreases the performances of our algorithms improve, while being always comparable with the benchmarks.