Consistent Submodular Maximization

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

Abstract

Maximizing monotone submodular functions under cardinality constraints is a classic optimization task with several applications in data mining and machine learning. In this paper we study this problem in a dynamic environment with consistency constraints: elements arrive in a streaming fashion and the goal is maintaining a constant approximation to the optimal solution while having a stable solution (i.e., the number of changes between two consecutive solutions is bounded). We provide algorithms in this setting with different trade-offs between consistency and approximation quality. We also complement our theoretical results with an experimental analysis showing the effectiveness of our algorithms in real-world instances.

1. Introduction

Submodular optimization is a powerful framework for modeling and solving problems that exhibit the widespread diminishing returns property. Thanks to its effectiveness, it has been applied across diverse domains, including video analysis (Zheng et al., 2014), data summarization (Lin & Bilmes, 2011; Bairi et al., 2015), sparse reconstruction (Bach, 2010; Das & Kempe, 2011), and active learning (Golovin & Krause, 2011; Amanatidis et al., 2022).

In this paper, we focus on submodular maximization under cardinality constraints: given a submodular function f, a universe of elements V, and a cardinality constraint k, the goal is to find a set S of at most k elements that maximizes $f(S)$. Submodular maximization under cardinality constraints is NP-hard, nevertheless efficient approximation algorithms exist for this task in both the centralized and the streaming setting (Nemhauser et al., 1978; Badanidiyuru et al., 2014; Kazemi et al., 2019).

[*]Equal contribution [1]Google Research [2]Sapienza University of Rome, Rome, Italy. Correspondence to: Federico Fusco <federico.fusco@uniroma1.it>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

One aspect of efficient approximation algorithms for submodular maximization that has received little attention so far, is the stability of the solution. In fact, for some of the known algorithms, even adding a single element to the universe of elements V may completely change the final output (see Appendix A for some examples). Unfortunately, this is problematic in many real-world applications where consistency is a fundamental system requirement. Indeed, a flurry of recent work has started to explore various optimization problems under stability and consistency constraints such as clustering (Lattanzi & Vassilvitskii, 2017; Cohen-Addad et al., 2022; Fichtenberger et al., 2021; Guo et al., 2021; Łacki et al., 2024), facility location (Cohen-Addad et al., 2019; Bhattacharya et al., 2022), and online learning (Jaghargh et al., 2019).

Having solutions that evolve smoothly is central in many practical application of submodular optimization. Consider, for example, the data summarization task in an evolving setting where elements are added to the universe V. In this setting, having a stable summary that changes as little as possible from step to step is very important both for serving the summary to a user or for using it in a machine learning model. In fact, in both settings a drastic change of the solution may have negative impact on system usability, it could harm user attention, and adversely effect the performance of the machine learning model.

For these reasons, in this paper we initiate the study of submodular maximization under consistency constraints, where we allow the solutions to change only slightly after each element insertion. More formally, consider a stream V of exactly n elements, chosen by an adversary. Denote by $V_t = \{e_1, \dots, e_t\} \subseteq V$ the set of all elements inserted up to the t-th stream operation, and let OPT_t be an optimum feasible solution for V_t. Our goal is to design an algorithm with two key properties. On the one hand, we want the algorithm to maintain, at the end of each operation t, a solution $S_t \subseteq V_t$, with $|S_t| \leq k$, of high value $f(S_t)$. In particular, we say that an algorithm is an α-approximation of the best solution if $\alpha f(S_t) \geq f(\mathrm{OPT}_t)$, for all $t = 1, \dots, n$. On the other hand, we want the dynamic solution to not change much after consecutive insertions: we say that an algorithm is C-consistent if $|S_t \setminus S_{t-1}| \leq C$ for all $t = 2, \dots, n$. In general, we say that an algorithm is consistent, without specifying C, when C is constant.

It is interesting to note that the SWAPPING algorithm by Chakrabarti & Kale (2015) already conjugates constant approximation with constant consistency[*]. SWAPPING maintains a dynamic feasible solution and each new arriving element is added to the solution if either it fits into the cardinality constraint or it is possible to swap it with some low-value element. It is well known that SWAPPING achieves a 4-approximation, and from the previous description it is also clear that it is 1-consistent.[†]

Putting consistency aside, it is NP-hard to get an approximation guarantee better than $e/(e-1)$ (Feige, 1998), which can be achieved by recomputing a greedy solution (Nemhauser et al., 1978) from scratch after every insertion. However, such approach is not consistent (see Appendix A).

A line of work that is related to our model is that of fully-dynamic submodular maximization (e.g., Lattanzi et al., 2020; Monemizadeh, 2020; Dütting et al., 2023; Banihashem et al., 2024). There, the algorithm is given an arbitrary stream of insertions and deletions, and the goal is to maintain a good dynamic solution with low amortized running time. While the constraint on running time naturally induces algorithms characterized by solutions that do not change often, known algorithms for fully dynamic submodular maximization are not consistent, as they all contemplate the possibility of recomputing the solution from scratch from time to time.

Our Contribution. Given these considerations, it is natural to ask if it is possible to obtain a better trade-off between quality and consistency. We answer this question positively:

- We first provide a $(3.147 + O(1/k))$-approximation algorithm that is 1-consistent, improving on the guarantees of the SWAPPING algorithm.
- We then provide a $(2.619 + \varepsilon)$-approximation[‡] algorithm that is $\tilde{O}(1/\varepsilon)$-consistent, where the $\tilde{O}$ notation hides poly-logarithmic factors in $1/\varepsilon$.

We complement our positive results with a lower bound showing that for any constant C, no deterministic algorithm can be C-consistent and return a better than 2 approximation. Since both our algorithms are deterministic, the lower bound shows that our algorithms obtain a near-optimal quality-consistency tradeoff. We leave the resolution of the remaining gaps, and the study of randomized algorithms as exciting directions for future work.

We also present extensive experiments with real-world data sets and a synthetic data set (Section 6 and Appendices B and C). The experiments show that our algorithms achieve comparable value as SWAPPING and the non-consistent SIEVE-STREAMING (Badanidiyuru et al., 2014) on real-world data sets; while achieving significant savings in the total number of changes. Furthermore, the synthetic data set – constructed using a hard instance for SWAPPING presented in Appendix B – confirms the improvements in the worst-case approximation guarantees relative to SWAPPING from our theoretical analysis, showing that there too the gains can be significant (in the order of the 21.325% and 34.525% improvements that we show in our analysis).

Our Techniques. Our first algorithm, ENCOMPASSING-SET, maintains a benchmark set B_t that is used to decide whether to add or discard any new element. More precisely, any arriving element e_t is added to B_{t-1} if, upon arrival,the marginal contribution of e_t to B_{t-1}, that is $f(e_t \mid B_{t-1})$, is at least $\beta/k \cdot f(B_{t-1})$. Here β is a judiciously chosen constant that is larger than 1, namely $\beta = 1.14$. At any given time t, the solution S_t maintained by the algorithm consists of the last k elements added to B_t.

This algorithm is 1-consistent by construction, while the approximation guarantee descends from the following two properties of this algorithm. First, the (potentially infeasible) benchmark set B_t achieves a $(1 + \beta)$-approximation to $f(\mathrm{OPT}_t)$ (where $1+\beta = 2.14$ by the choice of β). Second, due to the exponential nature of the condition by which elements are added to the benchmark set, the elements in B_t that are not part of S_t only account for a small fraction of the value of B_t; namely, $f(B_t) \geq (1 + \beta/k)^k f(B_t \setminus S_t)$. Intuitively, the second property shows that S_t captures a significant fraction of $f(B_t)$, while the first property shows that $f(B_t)$ is a good approximation to $f(\mathrm{OPT}_t)$. A careful analysis shows that the two properties lead to the claimed factor of $3.147 + O(1/k)$.

Our second algorithm, CHASING-LOCAL-OPT, provides a better approximation guarantee at the cost of possibly performing more than one swap per step (but still at most constantly many). Rather than maintaining a benchmark set, this algorithm only maintains a solution S_t, and updates it via local improvements. It applies a similar swapping condition as ENCOMPASSING-SET, by requiring that the marginal value of an arriving element e_t to S_{t-1} should be at least $\phi/k \cdot f(S_{t-1})$, where $\phi \approx 1.61$ is the golden ratio. Other than ENCOMPASSING-SET, however, rather than swapping out the oldest element that was added, it swaps

[*]Following e.g., Dütting et al. (2022; 2023), we call SWAPPING the instantiation of the general framework by Chakrabarti & Kale (2015) for the special case of matroid constraints. We refer to Appendix B for the pseudocode.

[†]It is possible to show that the 4 is tight for the approximation factor. For an example please refer to Appendix B.

[‡]As is common in the submodular maximization literature, the parameter ε is intended to be a small constant that the algorithm designer can tune according to the application at hand: it is possible to attain an approximation arbitrarily close to 2.619, at the cost of a worse consistency.

out an element r whose marginal contribution $f(r \mid S - r)$[§] to the current solution S is less than a $1/k$-fraction of the current solution's value $f(S)$.

Moreover, after the arrival of each element e_t and its possible addition to S_t, it performs up to $N = \tilde{O}(1/\varepsilon)$ additional swaps. For this it considers all elements that have arrived so far, which we denote by V_t, and it tries to add them one-by-one by the same condition and procedure that is used for newly arriving elements. The purpose of these extra swaps is to drive the maintained solution S_t closer to a local optimum: a solution $S \subseteq V_t$ such that there is no element $x \in V_t$ such that $f(x \mid S) \geq \phi/k \cdot f(S)$. The improved approximation guarantee then stems from the fact that either the algorithm was at a local optimum in the not too distant past, or it performed many swaps since.

2. Preliminaries

We consider a set function $f : 2^V \to \mathbb{R}_{\geq 0}$ on a ground set V of cardinality n. Given two sets $X, Y \subseteq V$, the *marginal gain* of X with respect to Y, $f(X \mid Y)$, quantifies the change in value of adding X to Y and is defined as

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

When X consists of a singleton x, we use the shorthand $f(x \mid Y)$ instead of $f(\{x\} \mid Y)$. 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 $X \subseteq Y \subseteq V$ 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 monotone and that it is *normalized*, i.e., $f(\emptyset) = 0$. We model access to the submodular function f via a value oracle that computes $f(S)$ for given $S \subseteq V$. The problem of maximizing a function f under a *cardinality constraint* k is defined as selecting a set $S \subseteq V$ with $|S| \leq k$ that maximizes $f(S)$.

3. Impossibility Result

Putting computational efficiency aside, it may be possible to design a consistent algorithm which maintains the optimal solution, or an arbitrarily good approximation. We prove that this is not the case: no deterministic algorithm with constant consistency enjoys an approximation guarantee better than 2. We remark that this is an information-theoretical bound, and concerns the streaming nature of the problem.

Theorem 3.1. *Fix any constant C and precision parameter $\varepsilon \in (0, 1)$. No C-consistent (deterministic) algorithm provides a $(2 - \varepsilon)$-approximation.*

Proof. Fix any constant C, precision parameter $\varepsilon > 0$, and a deterministic algorithm $\mathcal{A}$ that is C-consistent, we construct a covering instance such that $\mathcal{A}$ does not maintain a $(2 - \varepsilon)$ approximation. Let $G = \{g_1, \ldots, g_n\}$ be a ground set and V be a family of subsets of G such that V contains all the subsets of G of cardinality 1 and k, with $k = n/2$. The covering function f is naturally defined on V, and we consider the task of maximizing f with cardinality k.

Observe the behaviour of $\mathcal{A}$ on the sequence $\{g_1\}, \ldots \{g_n\}$. At the end of this partial sequence $\mathcal{A}$ maintains a certain solution $S = \{\{g_{i_1}\}, \ldots, \{g_{i_\ell}\}\}$, with $\ell \leq k$. Now suppose the next element to arrive is $\{g_{i_1}, \ldots, g_{i_\ell}, g_{i_{\ell+1}}, \ldots, g_{i_k}\}$, where $g_{i_{\ell+1}}, \ldots, g_{i_k}$ are some arbitrary elements not covered by S. The value of the optimal solution after this insertion is $2k - 1$ (just take the last subset and $k - 1$ non overlapping singletons). The value of S is $\ell \leq k$ and, even if $\mathcal{A}$ adds to S the subset $\{g_{i_1}, \ldots, g_{i_\ell}, g_{i_{\ell+1}}, \ldots, g_{i_k}\}$ and $C - 1$ other singletons, it cannot get a solution of value more than $k + C$. The theorem follows by choosing appropriate values for k: $k \geq 3C/\varepsilon$. □

[§] We use $S - r$ instead of $S \setminus \{r\}$ and $S + x$ instead of $S \cup \{x\}$.

Algorithm 1 ENCOMPASSING-SET

```
 1: Environment: Stream $V$, function $f$, cardinality $k$
 2: Threshold parameter $\beta \leftarrow 1.14$
 3: $B_0 \leftarrow \emptyset$, $S_0 \leftarrow \emptyset$, and $t \leftarrow 1$
 4: for $e_t$ new element arriving do
 5:     if $f(e_t \mid B_{t-1}) \geq \frac{\beta}{k} f(B_{t-1})$ then
 6:         $B_t \leftarrow B_{t-1} + e_t$
 7:         $S_t \leftarrow S_{t-1} + e_t$
 8:         if $|S_t| = k + 1$ then
 9:             remove from $S_t$ the element $e_s$ with smallest $s$
10:         $t \leftarrow t + 1$
```

4. ENCOMPASSING-SET

In this section, we present the ENCOMPASSING-SET algorithm, which achieves an approximation guarantee of $3.146 + O(1/k)$ and 1-consistency (changes at most one element for each insertion). ENCOMPASSING-SET maintains a benchmark set B_t to which it adds all the elements that, upon arrival, exhibit a marginal contribution to B_t that is at least $\beta/k \cdot f(B_t)$. At any given stream operation t, the solution S_t is given by the last k elements added to B_t. We refer to the pseudocode for further details. We prepare the analysis of the properties of ENCOMPASSING-SET with two Lemmata. We start relating the value of the optimal solution with that of the benchmark.

Lemma 4.1. *After each insertion e_t, the following holds:*

$$f(\mathrm{OPT}_t) \leq (1 + \beta) \cdot f(B_t).$$

Proof. Consider any element that belongs to OPT_t but not to the benchmark set B_t after the computation following the insertion of e_t, i.e., $e_s \in \mathrm{OPT}_t \setminus B_t$, with $s \leq t$. Element

e_s has not been included to B_s (because it does not belong to $B_t \supseteq B_s$) upon its insertion, so the following holds:

$$\begin{aligned} f(e_s \mid B_t) &\le f(e_s \mid B_{s-1}) && \text{(by submodularity)}\\ &\le \tfrac{\beta}{k} f(B_{s-1}) && \text{(since } e_s \notin B_s)\\ &\le \tfrac{\beta}{k} f(B_t). && \text{(by monotonicity)} \end{aligned}$$

So, for any element $e_s \in \mathrm{OPT}_t \setminus B_t$, it holds that

$$f(e_s \mid B_t) \le \tfrac{\beta}{k} f(B_t). \tag{1}$$

The above inequality is the crucial ingredient of the proof:

$$\begin{aligned} f(\mathrm{OPT}_t) &\le f(\mathrm{OPT}_t \cup B_t) && \text{(by monotonicity)}\\ &\le f(B_t) + \sum_{e_s \in \mathrm{OPT}_t \setminus B_t} f(e_s \mid B_t) \\ &\le f(B_t) + |\,\mathrm{OPT}_t\,| \cdot \tfrac{\beta}{k} f(B_t) && \text{(by Ineq. 1)}\\ &\le (1+\beta) f(B_t). && \text{(because } |\,\mathrm{OPT}_t\,| \le k) \end{aligned}$$

Note, the second inequality comes from submodularity. □

As a second preliminary step, we argue that the elements in B_t that are not included to the current solution S_t only account for a small fraction of $f(B_t)$.

Lemma 4.2. *After each insertion e_t, the following inequality holds:*

$$f(B_t) \ge \left(1 + \frac{\beta}{k}\right)^k f(B_t \setminus S_t).$$

Proof. The elements in the current solution are naturally sorted according to the order in which they are inserted in the stream and then added to the solution: $S_t = \{s_{t_1}, s_{t_2}, \dots s_{t_\ell}\}$. Element s_{t_ℓ} is the last one added and, clearly, $\ell \le k$ and $t_\ell \le t$. Each one of these e_{t_i} elements has been added to the solution because it passed the value test: $f(s_{t_i} \mid B_{t_i - 1}) \ge \frac{\beta}{k} f(B_{t_i-1})$.

Now, set B_{t_i-1} can be rewritten in terms of the current benchmark set B_t and the elements in the solution S_t: $B_{t_i-1} = B_t \setminus \{s_{t_i}, \dots, s_{t_\ell}\}$, so the previous inequality can be rewritten as

$$f(s_{t_i} \mid B_t \setminus \{s_{t_i}, \dots, s_{t_\ell}\}) \ge \tfrac{\beta}{k} f(B_t \setminus \{s_{t_i}, \dots, s_{t_\ell}\}).$$

If we add to both sides of the above inequality the term $f(B_t \setminus \{s_{t_i}, \dots, s_{t_\ell}\})$, we get that

$$f(B_t \setminus \{s_{t_i+1}, \dots, s_{t_\ell}\}) \ge \left(1 + \frac{\beta}{k}\right) f(B_t \setminus \{s_{t_i}, \dots, s_{t_\ell}\}).$$

Iterating the above argument we get the desired bound:

$$\begin{aligned} f(B_t) &\ge \left(1 + \tfrac{\beta}{k}\right) f(B_t \setminus \{s_{t_\ell}\})\\ &\ge \left(1 + \tfrac{\beta}{k}\right)^2 f(B_t \setminus \{s_{t_{\ell-1}}, s_{t_\ell}\})\\ &\ge \dots \ge \left(1 + \tfrac{\beta}{k}\right)^\ell f(B_t \setminus S_t). \end{aligned}$$

The Lemma follows by recalling that $\ell \le k$. □

We now have all the ingredients to analyze Encompassing-Set.

Theorem 4.3. Encompassing-Set *is* 1*-consistent and maintains a* $3.147 + O(1/k)$ *approximation.*

Proof. First observe that the algorithm is indeed 1-consistent: every time the solution S_t changes, exactly one element is inserted and exactly one is removed from it.

We move our attention to the approximation guarantee. We start by noting that

$$\begin{aligned} &f(B_t) + \left(1 + \tfrac{\beta}{k}\right)^k f(S_t)\\ &\quad\ge \left(1 + \tfrac{\beta}{k}\right)^k [f(S_t) + f(B_t \setminus S_t)] && \text{(Lemma 4.2)}\\ &\quad\ge \left(1 + \tfrac{\beta}{k}\right)^k f(B_t). && \text{(by submodularity)} \end{aligned}$$

By rearranging terms and applying Lemma 4.1 we get:

$$\begin{aligned} f(S_t) &\ge \frac{\left(1 + \frac{\beta}{k}\right)^k - 1}{\left(1 + \frac{\beta}{k}\right)^k} f(B_t)\\ &\ge \frac{\left(1 + \frac{\beta}{k}\right)^k - 1}{\left(1 + \frac{\beta}{k}\right)^k (1+\beta)} f(\mathrm{OPT}_t). \end{aligned} \tag{2}$$

We conclude the proof by providing a general lower bound for the multiplier of the right-hand side of the last inequality. We know that the following simple chain of inequality holds:

$$\left(1 + \frac{\beta}{k}\right)^k \le e^\beta \le \left(1 + \frac{\beta}{k}\right)^k \left(1 - \frac{\beta^2}{k}\right)^{-1}$$

Plugging the above inequality into the multiplier in Equation (2), we have

$$\begin{aligned} \frac{\left(1 + \frac{\beta}{k}\right)^k - 1}{\left(1 + \frac{\beta}{k}\right)^k (1+\beta)} &\ge \frac{e^\beta - 1}{e^\beta(1+\beta)} - \frac{\beta^2}{k(1+\beta)}\\ &\ge 0.3178 - \frac{1}{k}. && (\beta = 1.14) \end{aligned}$$

Taking the inverse yields the desired factor. □

5. Chasing-Local-Opt

In this section we present and analyze the Chasing-Local-Opt algorithm, which exhibits a better approximation factor than both Swapping and Encompassing-Set. We refer to the pseudocode for further details. There are

Algorithm 2 MIN-SWAP(S, x)

```
1: Input: Set $S$ and element $x$
2: Environment: Function $f$ and cardinality $k$
3: if $|S| < k$, then return $S + x$
4: Let $r \in S$ be any element s.t. $f(r \mid S - r) \leq f(S)/k$
5: return $S - r + x$
```

Algorithm 3 CHASING-LOCAL-OPT

```
1: Input: Precision parameter $\varepsilon$
2: Environment: Stream $V$, function $f$, cardinality $k$
3: $\phi \leftarrow \frac{\sqrt{5}+1}{2}$, $N \leftarrow \lceil \frac{1}{\varepsilon} \log_\phi \frac{12}{\varepsilon} \rceil$
4: $S_0 \leftarrow \emptyset$ and $t \leftarrow 1$
5: for $e_t$ new element arriving do
6:     if $f(e_t \mid S_{t-1}) \geq \frac{\phi}{k} f(S_{t-1})$ then
7:         $S_t \leftarrow$ MIN-SWAP$(S_{t-1}, e_t)$
8:     for $i = 1, \ldots, N$ do
9:         if $\exists\, x \in V_t$ such that $f(x \mid S_t) \geq \frac{\phi}{k} f(S_t)$ then
10:            $S_t \leftarrow$ MIN-SWAP$(S_t, x)$
11:    $t \leftarrow t + 1$
```

two differences with respect to ENCOMPASSING-SET. First, the way in which elements in the solution are swapped out: it is not the "oldest" element to be removed, but one with small enough value. This is formalized in the routine MIN-SWAP, which takes as input a set S and an element x, and is responsible for inserting x into S; if S already contains k elements, then x is swapped with an element r in S with marginal value not larger than the average value of S (so to maintain the cardinality of S bounded by k). Note, such an element r always exists by submodularity and a simple averaging argument:

$$f(S) \geq \sum_{x \in S} f(x \mid S - x) \geq k \cdot \min_{x \in S} f(x \mid S - x) \,.$$

The second difference is that after the arrival of each element and possibly its addition to the current solution, the algorithm performs up to $N \in \tilde{O}(1/\varepsilon)$ additional swaps from V_t into the solution, using the same rule and subroutine as for newly arriving elements. The additional swaps performed by CHASING-LOCAL-OPT drive the maintained solution closer to a local optimum defined as follows.

Definition 5.1. We say that a dynamic solution S_t is a local optimum if there exists no element x in V_t such that $f(x \mid S_t) \geq \frac{\phi}{k} f(S_t)$.

The improved approximation guarantee stems from the fact that at any point in time, either the solution maintained by the algorithm was a local optimum not too far in the past, or many swaps were performed since.

Theorem 5.2. CHASING-LOCAL-OPT *is $\tilde{O}(1/\varepsilon)$-consistent and maintains a $(\phi + 1 + 9\varepsilon)$-approximation, where $\phi \approx 1.619$ is the golden ratio.*

Before proving the theorem, we introduce a notational convention. During the execution of the algorithm, elements may be added and removed multiple times from the dynamic solution. Rather than thinking of such an element as one and the same element, it is convenient to think of this happening to multiple distinct copies of the same element so that each element is added and removed at most once. This allows us to work with sets instead of multi-sets in the analysis.

Proof of Theorem 5.2. The bound on the consistency is immediate, as for each insertion there are at most $N + 1 = \lceil 1/\varepsilon \log_\phi 12/\varepsilon \rceil + 1 = \tilde{O}(1/\varepsilon)$ changes in the solution. The rest of the proof is devoted to the analysis of the approximation guarantee, which we prove by induction on the number of insertions. For the first element e_1 of the stream there is nothing to prove, as $S_1 = V_1 = \{e_1\}$. We analyze now the generic insertion e_t, with $t > 1$, assuming that the desired approximation holds for any previous insertion $s < t$. Let t' be the last insertion index before t in which the solution $S_{t'}$ was a local optimum (see Definition 5.1), and denote with τ the maximum between t' and $(t - \lceil \varepsilon k \rceil)$. We remark that t' is at least 1, so τ is well defined. We have that OPT_t is the optimum after insertion e_t, and OPT_τ is the optimum after insertion e_τ. Sets V_t and V_τ, V_t and V_τ are defined in a similar way. Consider how the solution changed between S_τ and S_t: some elements in S_τ were removed, some were added and remained in S_t, while others were added and later removed, possibly multiple times. To ease the analysis, we sort these inserted elements $s_1, s_2, \ldots, s_L$ according to the order in which they were added (recall that multiple "copies" of the same element may appear in this sequence); this induces a natural sorting on the removed elements: we call r_ℓ the element that was swapped out to make room for s_ℓ (to avoid confusion, if no element was swapped out, we let r_ℓ be a dummy element with no value). We now define an auxiliary sequence of sets A_ℓ that interpolates between the solution at insertion τ and that at insertion t: $A_\ell = S_\tau \cup \{s_1, \ldots, s_\ell\} \setminus \{r_1, \ldots, r_\ell\}$.

It holds that $S_\tau = A_0$, while $S_t = A_L$. Moreover, the definition of the auxiliary sets motivates this relation:

$$A_{\ell-1} + s_\ell = A_\ell + r_\ell. \tag{3}$$

By a telescopic argument, the above relation and the design of CHASING-LOCAL-OPT we have the following claim.

Claim 5.3. *The following inequality holds true:*

$$f(S_t) \geq f(S_\tau) + (\phi - 1) \sum_{\ell=1}^{L} f(s_\ell \mid A_{\ell-1} - r_\ell).$$

Proof of Claim 5.3. The change in value between two consecutive auxiliary sets can be decomposed as follows exploiting the relation in Equation (3):

$$f(A_\ell) - f(A_{\ell-1}) = f(s_\ell \mid A_{\ell-1}) - f(r_\ell \mid A_\ell). \tag{4}$$

Now, the marginal value of s_ℓ with respect to $A_{\ell-1}$ is at least $\phi/k \cdot f(A_{\ell-1})$, by the swapping conditions in lines 6 and 9 of CHASING-LOCAL-OPT. Furthermore, by the design of MIN-SWAP, we know that the element r_ℓ that is removed to make room for s_ℓ has small value. In formula,

$$f(s_\ell \mid A_{\ell-1}) \geq \tfrac{\phi}{k} f(A_{\ell-1}) \geq \phi f(r_\ell \mid A_{\ell-1} - r_\ell) \qquad (5)$$

We can now prove directly the inequality in the statement:

$$\begin{aligned} &f(S_t) - f(S_\tau) \\ &\quad= \sum_{\ell=1}^{L} f(A_\ell) - f(A_{\ell-1}) \quad \text{(telescopic argument)} \\ &\quad= \sum_{\ell=1}^{L} f(s_\ell \mid A_{\ell-1}) - f(r_\ell \mid A_\ell) \quad \text{(by Eqn. 4)} \\ &\quad\geq \sum_{\ell=1}^{L} f(s_\ell \mid A_{\ell-1}) - f(r_\ell \mid A_{\ell-1} - r_\ell) \\ &\quad\geq (\phi - 1) \sum_{\ell=1}^{L} f(r_\ell \mid A_{\ell-1} - r_\ell). \quad \text{(by Eqn. 5)} \end{aligned}$$

Note, the second to last inequality follows by submodularity and the fact that $A^{\ell-1} - r_\ell = A_\ell - s_\ell \subseteq A_\ell$, due to the relation in Equation (3). □

Denote now with I the set of elements that were inserted between e_τ and e_t : $I = V_t \setminus V_\tau$, and with A the set of all the elements that were, at some point, in the solution between time τ and t: $A = \cup_{\ell=\tau}^{t} A_\ell$. It is possible to relate the value of S_t with that of the elements in I and A:

Claim 5.4. *The following inequality holds true:*

$$f(I \cup A) \leq (1 + 4\varepsilon) f(S_t) + \sum_{\ell=1}^{L} f(r_\ell \mid A_{\ell-1} - r_\ell).$$

Proof of Claim 5.4. Consider any element g in $I \cup A$. We have three cases: either element g belongs to S_t, g was added to the solution but was later swapped out, or it failed the swapping condition in line 6 upon insertion. Now, sort these elements according to the order in which they were discarded by the algorithm: $(I \cup A) \setminus S_t = \{g_1, \dots, g_J\}$ ($g \in I \setminus A$ is discarded upon insertion, while $g \in A \setminus (I \cup S_t)$ is discarded when gets swapped out by the solution). For simplicity, denote with G_j the set of the first $j-1$ such elements, we have the following two facts: (i) if $g_j \in I \setminus A$, then it means that $g_j = e_{t'}$ for some $t' \in \{\tau, \dots, t\}$, and the solution $S_{t'} \subseteq S_t \cup G_j$; (ii) if $g_j \in A \setminus (I \cup S_t)$, then it means that $g_j = r_\ell$ for some $\ell \in \{1, \dots, L\}$, and it holds that $A_\ell - r_\ell \subseteq S_t \cup G_j$.

Exploiting these two facts and submodularity, we have the following chain of inequalities:

$$\begin{aligned} &f(I \cup A) - f(S_t) \\ &\quad= \sum_{j=1}^{J} f(g_j \mid S_t \cup G_j) \\ &\quad\leq \sum_{e_{t'} \in I \setminus (S_t \cup A)} f(e_{t'} \mid S_{t'-1}) + \sum_{\ell=1}^{L} f(r_\ell \mid A_{\ell-1} - r_\ell) \\ &\quad\leq \frac{\phi}{k} \sum_{e_{t'} \in I \setminus (S_t \cup A)} f(S_{t'-1}) + \sum_{\ell=1}^{L} f(r_\ell \mid A_{\ell-1} - r_\ell) \\ &\quad\leq 4\varepsilon f(S_t) + \sum_{\ell=1}^{L} f(r_\ell \mid A_{\ell-1} - r_\ell). \end{aligned}$$

Note, the second inequality holds by the fact that e_j failed the swapping condition in line 6 upon insertion; while the third inequality follows by observing that the sequence of $f(S_{t'})$ is non-decreasing, there are at most $2\varepsilon k$ elements in $I \setminus (S_t \cup A)$, and $\phi \in (1, 2)$. □

Another useful property of the auxiliary sets A_ℓ is to provide a clean way to formalize that adding new elements to the solution multiplicatively improves the value of the solution.

Claim 5.5. *The following inequality holds true:*

$$f(S_t) \geq \left(1 + \frac{\phi - 1}{k}\right)^L f(S_\tau).$$

Proof of Claim 5.5. Consider the generic subsequent terms $\ell - 1$ and ℓ, for $\ell = 1, \dots, L$. Starting from rearranging Equation (4), we have the following:

$$\begin{aligned} f(A_\ell) &= f(s_\ell \mid A_{\ell-1}) - f(r_\ell \mid A_\ell) + f(A_{\ell-1}) \\ &\geq f(s_\ell \mid A_{\ell-1}) - f(r_\ell \mid A_{\ell-1} - r_\ell) + f(A_{\ell-1}) \\ &\geq \frac{\phi - 1}{\phi} f(s_\ell \mid A_{\ell-1}) + f(A_{\ell-1}) \quad \text{(by Eqn. 5)} \\ &\geq \left(1 + \frac{\phi - 1}{k}\right) f(A_{\ell-1}), \end{aligned}$$

where the first inequality follows by submodularity and the relation in Equation (3), while the last one by the design of MIN-SWAP: an element is added to the solution only if its marginal contribution is at least a ϕ/k fraction of $f(A_{\ell-1})$. Applying iteratively the above argument from $S_t = A_L$ to $S_\tau = A_0$ yields the desired result. □

We now have all the ingredients to directly address the crux of the proof. We have two cases we analyze separately: either S_τ is a local optimum, or it is not.

S_τ is a local optimum. If S_τ is a local optimum, then all the elements in OPT_t that arrived before e_τ, i.e., $\mathrm{OPT}_t \cap V_\tau$ have low marginal contribution with respect to S_τ. Formally, we have the following result.

Claim 5.6. *If S_τ is a local optimum, then*

$$(1+4\varepsilon) f(S_t) + \sum_{\ell=1}^{L} f(r_\ell \mid A_{\ell-1} - r_\ell) \geq f(\mathrm{OPT}_t) - \phi f(S_\tau).$$

Proof of Claim 5.6. To prove this result, it suffices to argue that the right-hand side of the inequality in the statement is at most $f(I \cup A)$, as it is then possible to conclude the argument by combining it with Claim 5.4. We have the following:

$$\begin{aligned} f(\mathrm{OPT}_t) &- f(I \cup A) \\ &\leq f(\mathrm{OPT}_t \mid I \cup A) && \text{(by monotonicity)} \\ &= f(\mathrm{OPT}_t \cap V_\tau \mid I \cup A) && \text{(since } I = V_t \setminus V_\tau\text{)} \\ &\leq \sum_{e \in \mathrm{OPT}_t \cap V_\tau} f(e \mid S_\tau) \\ &\leq \phi f(A_\tau). && (A_\tau \text{ local optimum}) \end{aligned}$$

Note, the second inequality holds by submodularity as S_τ is contained into A. Reordering the terms of the inequality we get the desired lower bound on $f(I \cup A)$. □

Summing the inequality in Claim 5.6 with ϕ times the inequality in Claim 5.3 yields the desired bound, thus concluding the argument for the first case:

$$\begin{aligned} &(1+\phi+4\varepsilon) f(S_t) \\ &\quad \geq f(\mathrm{OPT}_t) + [\phi(\phi-1)-1] \sum_{\ell=1}^{L} f(r_\ell \mid A_{\ell-1} - r_\ell) \\ &\quad = f(\mathrm{OPT}_t). \end{aligned}$$

In the previous inequality we crucially used the definition of the golden ratio as the solution of $\phi^2 - \phi - 1 = 0$.

S_τ is not a local optimum. If S_τ is not a local optimum, then it means that L, the total number of swaps between insertion e_τ and e_t, is at least $\varepsilon \cdot k \cdot N$, where N is defined in the pseudocode as $\lceil 1/\varepsilon \cdot \log_\phi 12/\varepsilon \rceil$. If we complement this with Claim 5.5 we get:

$$\begin{aligned} f(S_t) &\geq f(S_\tau) \cdot \left(1 + \frac{\phi - 1}{k}\right)^{k \log_\phi 12/\varepsilon} \\ &\geq \frac{12}{\varepsilon} f(S_\tau) && \text{(because } (1+\tfrac{x}{n})^n \geq 1 + x\text{)} \\ &\geq \frac{12}{(1+\phi+9\varepsilon)\varepsilon} f(\mathrm{OPT}_\tau), \end{aligned}$$

where in the last inequality we crucially used the inductive assumption. By rearranging and using that $\varepsilon \in (0,1)$ and $\phi \in (1,2)$, we get the following simple relation, which proves that the value of the elements arrived up to time τ can be safely ignored:

$$f(\mathrm{OPT}_\tau) \leq \varepsilon f(S_t). \tag{6}$$

We have all the ingredient to deal with the last case:

$$\begin{aligned} f(\mathrm{OPT}_t) &\leq f(\mathrm{OPT}_t \cap V_\tau) + f(I) && \text{(by submodularity)} \\ &\leq f(\mathrm{OPT}_\tau) + f(I) && \text{(by optimality of } \mathrm{OPT}_\tau\text{)} \\ &\leq \varepsilon f(S_t) + f(I) && \text{(by Equation 6)} \\ &\leq \varepsilon f(S_t) + f(I \cup A) && \text{(by monotonicity)} \\ &\leq (1+5\varepsilon) f(S_t) + \sum_{\ell=1}^{L} f(r_\ell \mid A_{\ell-1} - r_\ell) \\ & && \text{(by Claim 5.4)} \\ &\leq \left(1 + \frac{1}{\phi - 1} + 9\varepsilon\right) f(S_t) && \text{(by Claim 5.3)} \\ &= (1+\phi+9\varepsilon) f(S_t), \end{aligned}$$

where in the last equality we used the definition of ϕ as the golden ratio. This last case concludes the proof. □

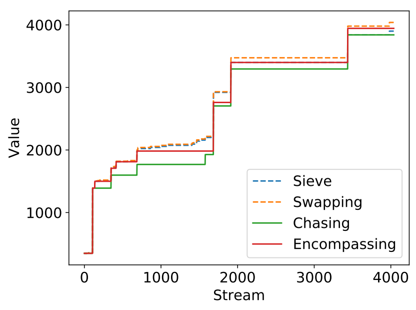
6. Experiments

In this section we evaluate the performance of our two algorithms on real-world data sets[¶]. We report here three case studies, while we defer to Appendix C other (qualitatively analogous) results, as well as further implementation details. We present additional experimental results that illustrate the gains in worst-case approximation guarantee in Appendix B. As benchmarks we consider the SWAPPING algorithm which provides a 4-approximation and is 1-consistent and the SIEVE-STREAMING algorithm, a $(2+\varepsilon)$-approximation that is not consistent (see Appendix A for further details on the instability of the algorithm).

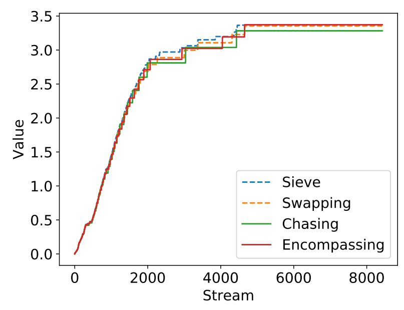
Influence Maximization. For our first case study we consider the problem of influence maximization on a social network graph (e.g., Norouzi-Fard et al., 2018; Halabi et al., 2020), where the goal is to maintain a subset of the nodes to "influence" the rest of the graph. In such application, consistency is crucial as changing nodes may entail costs relative to terminating and issuing new contracts. We use the Facebook dataset from McAuley & Leskovec (2012) that consists of 4039 nodes V and 88234 edges E and, as measure of influence we consider the monotone and submodular dominating function:

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

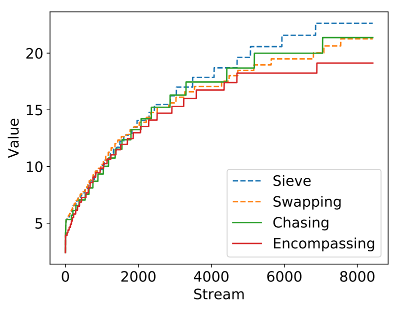
[¶]The code of the experiments is available at https://github.com/fedefusco/Consistent-Submodular.



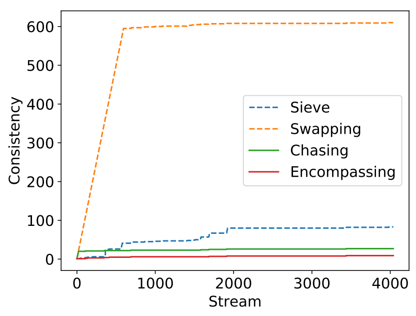
(a) Influence Maximization on Facebook



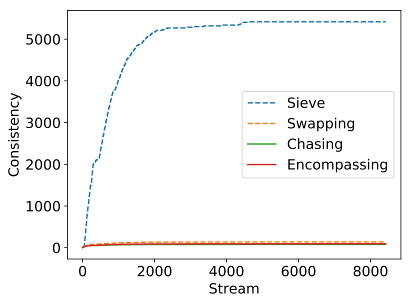
(b) k-medoid Clustering on RunInRome



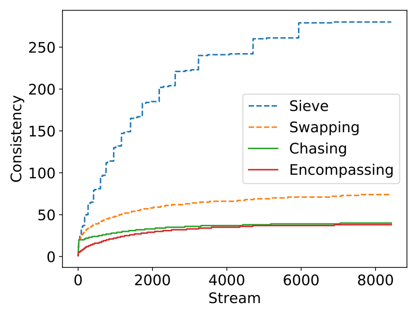
(c) LogDet Maximization on RunInRome



(d) Influence Maximization on Facebook



(e) k-medoid Clustering on RunInRome



(f) LogDet Maximization on RunInRome

Figure 1: Experimental Results. The first row reports the objective values, the second one the cumulative consistency.

Summarizing Geolocation Data. Our second and third case study concern the problem of maintaining a stable and representative summary from a sequence of geographical coordinates (e.g., Mirzasoleiman et al., 2017; Dütting et al., 2022). We use the RunInRome dataset (Fusco, 2022), that contains 8425 positions recorded by running activity in Rome, Italy. We consider 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 i^{th} and the j^{th} point in the dataset and h is some constant. Following Krause & Golovin (2014), another 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).

Experimental Results. In Figure 1, we present the performance of our algorithms and the benchmarks. The first row (Figures 1a to 1c) features the objective value of the dynamic solution maintained by the algorithms, while the second row (Figures 1d to 1f) reports the cumulative number of changes in the solutions. The experiments show that our algorithms, ENCOMPASSING-SET and CHASING-LOCAL-OPT, achieve comparable value as SWAPPING and SIEVE-STREAMING; while achieving notable savings in the total number of changes. For instance, in the setting of Figure 1d, SWAPPING is significantly less consistent on aggregate than our algorithms (around a factor 25), while SIEVE-STREAMING changes the solution about $3-4$ times more often. The superior cumulative consistency of our algorithms is also clear in the other experiments; in the settings of Figure 1e and Figure 1f SIEVE-STREAMING performs order of magnitudes more changes than either of our algorithms (about 500x and 10x), while SWAPPING performs between 50% and 100% more. The strict "insertion rules" implemented by our two algorithms seem to guarantee that only the crucial elements of the dataset are added to the solution. This phenomenon empirically induces a desirable global stability over the entire stream – which goes beyond the theoretical per-round guarantees – at the cost of possibly discarding moderately good elements.

7. Conclusion

In this paper, we initiate the study of consistency in submodular maximization. Consistency is a natural measure of stability of the online solution maintained by an algorithm, and has been extensively studied for clustering, facility location and online learning. We present two consistent algorithms, ENCOMPASSING-SET and CHASING-LOCAL-OPT, that exhibit a different approximation-consistency trade off ($3.147 + O(1/k)$ and 1-consistent vs. $2.619 + O(\varepsilon)$ and $O(1/\varepsilon)$-consistent). They both substantially improve on the state of the art (a consistent 4-approximation), moving the approximability boundary closer to the optimal approximation factor, as evidenced by the information-theoretical lower bound of 2 that we prove to hold for any consistent deterministic algorithm. Besides closing the remaining gap in the approximation factor, our work raises many natural and compelling questions. First, the investigation of randomized algorithms may lead to better results, even beyond the lower bound of 2. Second, while some known algorithms already exhibit consistency, the explicit study of consistent algorithms for possibly non-monotone submodular functions and more general constraints (e.g., matroids and knapsack) may lead to improved results.

Impact Statement

The work is theoretical in nature and does not have any significant ethical implications.

Acknowledgments

Federico Fusco is supported by the PNRR FAIR (Future Artificial Intelligence Research) project PE0000013 Spoke 5, the ERC Advanced Grant 788893 AMDROMA "Algorithmic and Mechanism Design Research in Online Markets", the PNRR MUR project IR0000013-SoBigData.it, and the MUR PRIN project "Learning in Markets and Society".

References

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.

Amanatidis, G., Fusco, F., Lazos, P., Leonardi, S., and Reiffenhäuser, R. Fast adaptive non-monotone submodular maximization subject to a knapsack constraint. *J. Artif. Intell. Res.*, 74:661–690, 2022.

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

Badanidiyuru, A., Mirzasoleiman, B., Karbasi, A., and Krause, A. Streaming submodular maximization: massive data summarization on the fly. In *KDD*, pp. 671–680, 2014.

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.

Banihashem, K., Biabani, L., Goudarzi, S., Hajiaghayi, M., Jabbarzade, P., and Monemizadeh, M. Dynamic algorithms for matroid submodular maximization. In *SODA*, pp. 3485–3533. SIAM, 2024.

Bhattacharya, S., Lattanzi, S., and Parotsidis, N. Efficient and stable fully dynamic facility location. In *NeurIPS*, 2022.

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

Cohen-Addad, V., Hjuler, N., Parotsidis, N., Saulpic, D., and Schwiegelshohn, C. Fully dynamic consistent facility location. In *NeurIPS*, pp. 3250–3260, 2019.

Cohen-Addad, V., Lattanzi, S., Maggiori, A., efficient, N., and stabledis. Online and consistent correlation clustering. In *ICML*, pp. 4157–4179, 2022.

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

Dütting, P., Fusco, F., Lattanzi, S., Norouzi-Fard, A., and Zadimoghaddam, M. Deletion robust submodular maximization over matroids. In *ICML*, pp. 5671–5693, 2022.

Dütting, P., Fusco, F., Lattanzi, S., Norouzi-Fard, A., and Zadimoghaddam, M. Fully dynamic submodular maximization over matroids. In *ICML*, volume 202 of *Proceedings of Machine Learning Research*, pp. 8821–8835. PMLR, 2023.

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

Fichtenberger, H., Lattanzi, S., Norouzi-Fard, A., and Svensson, O. Consistent k-clustering for general metrics. In *SODA*, pp. 2660–2678, 2021.

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.

Guo, X., Kulkarni, J., Li, S., and Xian, J. Consistent k-median: Simpler, better and robust. In *AISTATS*, pp. 1135–1143, 2021.

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

Halabi, M. E., Fusco, F., Norouzi-Fard, A., Tardos, J., and Tarnawski, J. Fairness in streaming submodular maximization over a matroid constraint. In *ICML*, volume 202 of *Proceedings of Machine Learning Research*, pp. 9150–9171. PMLR, 2023.

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

Jaghargh, M. R. K., Krause, A., Lattanzi, S., and Vassilvitskii, S. Consistent online optimization: Convex and submodular. In *AISTATS*, pp. 2241–2250, 2019.

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

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 *ICML*, pp. 3311–3320, 2019.

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

Lattanzi, S. and Vassilvitskii, S. Consistent k-clustering. In *ICML*, pp. 1975–1984, 2017.

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. A class of submodular functions for document summarization. In *ACL*, pp. 510–520, 2011.

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

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*, volume 70 of *Proceedings of Machine Learning Research*, pp. 2449–2458. PMLR, 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.

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

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.

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.

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

Łacki, J., Haeupler, B., Grunau, C., Jayaram, R., and Rozhoň, V. Fully dynamic consistent k-center clustering. In *SODA*, pp. 3463–3484, 2024.

A. Instability of known algorithms

We propose here two instances that highlight the instability of known algorithms. The instance in Example A.1 is such that both the optimal solution and the output of the greedy algorithm (Nemhauser et al., 1978) change entirely after every insertion. We then briefly discuss, in Example A.2, a simple instance that forces the SIEVE-STREAMING algorithm (Badanidiyuru et al., 2014) and its modified version SIEVE-STREAMING++ (Kazemi et al., 2019) to behave in a non-consistent way.

Example A.1. Let $\delta \in (0,1)$ be a small parameter used to break ties, and consider the following weighted covering instance, parameterized by an integer i and cardinality constraint k. The base set E is given by the pairs $\{(a,b),\ \text{for } a,b \in \{0,\dots,i\}\}$. We refer to each pair (a,b) as an item. The weights of the items are as follows: all items have unitary weight, but the following:

$$\begin{cases} w_{(0,0)} = & 0 \\ w_{(a,0)} = & \delta \cdot (2a+1) \quad \text{for } a \neq 0 \\ w_{(0,b)} = & \delta \cdot 2b \quad \text{for } b \neq 0 \end{cases}$$

The weighted covering function is monotone submodular and is defined as follows:

$$f(S) = \sum_{(a,b): \exists s \in S, (a,b) \in s} w_{(a,b)}.$$

Note, f is defined over subsets of E, not on items. The subsets of E we consider in our instance are the rows and columns of E: R_a is defined as $\{(a,0),\dots,(a,i)\}$, while C_b is defined as $\{(0,b),\dots,(i,b)\}$. The stream is constructed as follows: $C_1, R_1, C_2, R_2, \dots, C_\ell, R_\ell, \dots, C_i, R_i$. Consider now what happens after $2k$ insertions. The optimal solution (which is the same output by running greedy on the elements arrived so far) is as follows: if the last arrived element is a row, then the optimal solution is given by the last k arrived rows; conversely, if the last arrived element is a column, then the optimal solution is given by the last k arrived columns. This means that the k elements in the dynamic solution change after each insertion! Note, the elements in the first row and first column are only there for tie-breaking.

Example A.2. The SIEVE-STREAMING algorithms lazily maintains a set of geometrically increasing active thresholds O (of the type $\tau = (1+\varepsilon)^j$, for some $j \in \mathbb{Z}$ and input parameter ε) and a candidate solution for each one of them; then outputs the best of these candidates. In particular, when a new element e_t arrives, with value way larger than all the previous ones, a new threshold is activated and the corresponding candidate solution S_τ is initiated ($S_\tau = \{e_t\}$). It is then clear that any instance characterized by elements with dramatically increasing values would force the algorithm to continuously change its solution. For instance, consider an additive function with $f(e_t) = 2^t$: after each insertion, the solution output by SIEVE-STREAMING would be the singleton $\{e_t\}$. Playing with similar arguments, it is not hard to construct an instance that completely change solution every k insertions (e.g., $f(e_t) = k^{\lceil t/k \rceil}$).

B. The analysis of SWAPPING is tight

The SWAPPING algorithm is known to provide a 4-approximation to the optimum (Chakrabarti & Kale, 2015). In this Section we first report the pseudocode for completeness, and then prove that the analysis is tight, meaning that for any $\varepsilon \in (0,1)$, there exists an instance of the problem where the solution computed by SWAPPING is at least a $(4-\varepsilon)$ factor away from the optimal one (Example B.1). Finally, in Figure 2 we report the empirical performances of SWAPPING, SIEVE-STREAMING, and our algorithms on such hard instance.

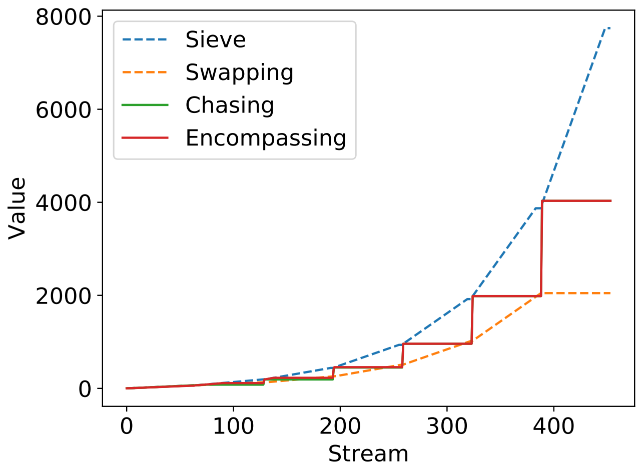
Example B.1. Fix any $\varepsilon \in (0,1)$, and consider the following weighted covering instance, parameterized by an integer i that we set later and the cardinality constraint $k = 2^i$. The set of items is $E = \{e_\ell^j \mid j \in \{0,\dots,i\}, \ell \in \{1,\dots,k\}\}$. Consider the partition of E into $i+1$ bundles of items $E^0, E^1, \dots, E^i$, where each bundle has k items $E^j = \{e_1^j, \dots, e_k^j\}$. Let $\delta > 0$ be a small positive constant, which we will set later. The weight of the generic element e_ℓ^j in E is $w_\ell^j = 2^j$ if $j \neq i$ and $w_\ell^i = 2^i - \delta$ otherwise. Now that we have the auxiliary set E, we can define the stream π of subsets of E as follows. For $0 \leq j < i$, let π^j be the subsequence $\{e_1^j\}, \dots, \{e_k^j\}, E^j$. Let π^i be the subsequence $\{e_1^i\}, \dots, \{e_k^i\}$ (without bundle E^i at the end). Then π is given by the concatenation of $\pi^0, \pi^1, \dots, \pi^i$.

Now, the behaviour of SWAPPING on π is clear: it maintains in the solution the last k singletons that arrived up to bundle E^{i-1} and ignores the elements in E^i (because of the small δ). In particular, at the end of the stream outputs the solution $S = \{\{e_1^{i-1}\}, \dots, \{e_k^{i-1}\}\}$, for a value of

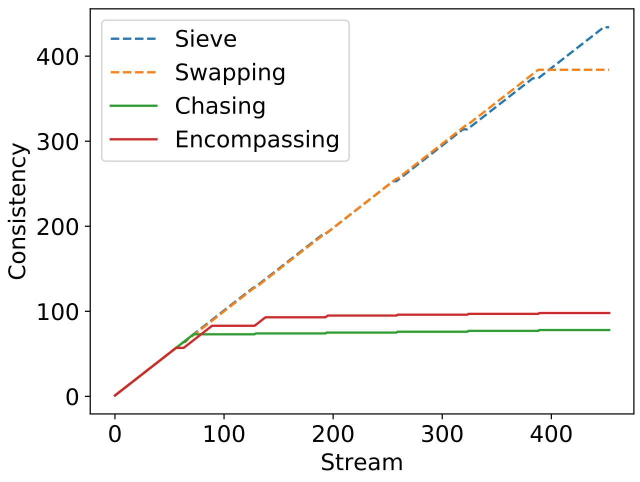
$$f(S) = \sum_{\ell=1}^{k} w_\ell^{i-1} = k \cdot 2^{i-1}.$$

Algorithm 4 SWAPPING

```
1:  Environment: stream $\pi$ of elements, function $f$, cardinality $k$
2:  $S \leftarrow \emptyset$
3:  for each new arriving element $e$ from $\pi$ do
4:      $w(e) \leftarrow f(e \mid S)$
5:      if $|S| < k$ then
6:          $S \leftarrow S + e$
7:      else
8:          $s_e \leftarrow \operatorname{argmin}\{w(y) \mid y \in S\}$
9:          if $2 \cdot w(s_e) \leq w(e)$ then
10:             $S \leftarrow S - s_e + e$
11: Return $S$
```



(a) Weighted Covering - Objective Value



(b) Weighted Covering - Cumulative Consistency

Figure 2: Experimental results on the weighted covering instance of Example B.1, for $\delta = 0.01$, $i = 7$, and $\varepsilon = 0.1$.

Consider now the optimal solution $S^\star$ given by the i bundles $E^0, \ldots, E^{i-1}$ and $k - i$ singletons from the last bundle, e.g., $\{e_1^i\}, \ldots, \{e_{k-i}^i\}$, for a value of

$$f(S^\star) = \sum_{j=0}^{i} \sum_{\ell=1}^{k} w_\ell^j - \sum_{\ell=k-i+1}^{k} w_\ell^i \geq k \cdot (2^{i+1} - 1) - k\delta - i \cdot 2^i.$$

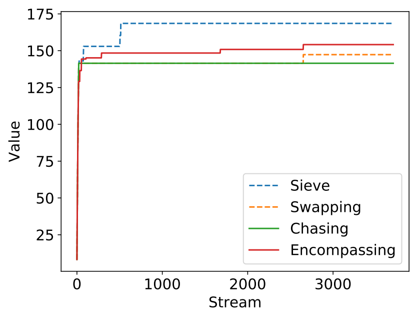
Note, $S^\star$ is indeed the optimal solution because of our choice of $k = 2^i$: the total weight of the elements in E_0 is k, while a singleton from E_i has weight $2^i - \delta$. We can now focus on the approximation factor, we have:

$$\frac{f(S^\star)}{f(S)} \geq 4 - \frac{2}{2^i}(\delta + 1 + i).$$

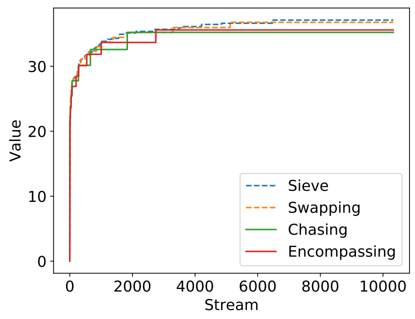
Now, the negative terms go to zero when i goes to infinity (and δ is small enough), thus for any fixed precision ε it is possible to set i and δ so that $f(S^\star)/f(S) \geq 4 - \varepsilon$.

C. Further Experimental Results

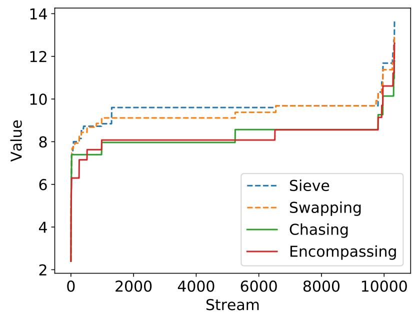
In our experiments, we set $\varepsilon = 0.1$ in SIEVE-STREAMING and CHASING-LOCAL-OPT, while the cardinality constraint k is consistently set to 20. The order of the stream of elements is the one intrinsic in the dataset we consider. In Figure 3, we report three extra experimental case studies. Besides studying the k-medoid and logdet objective on a random sample (10332 points) from the Uber pickups dataset (Kaggle, 2020) (see the last two columns of Figure 3 for the results), we present results for Personalized Movie Recommendation (first column of Figure 3).



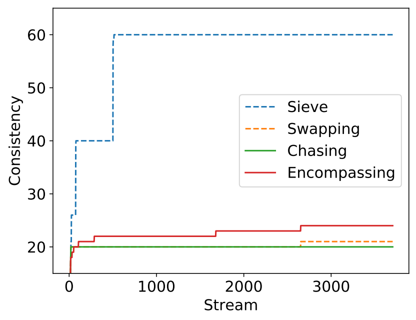
(a) MovieLens Dataset



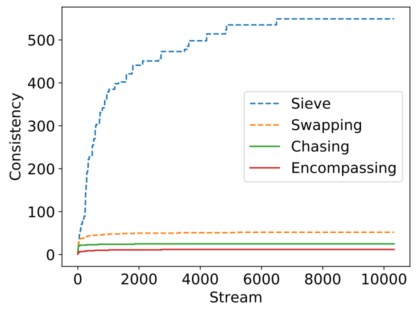
(b) k-medoid Clustering on Uber Dataset



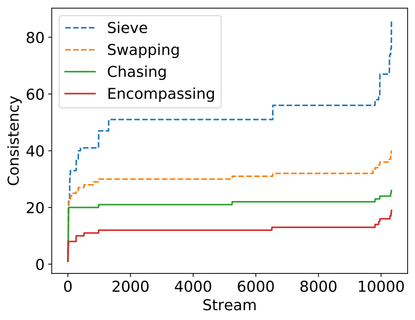
(c) LogDet Maximization on Uber Dataset



(d) MovieLens Dataset



(e) k-medoid Clustering on Uber Dataset



(f) LogDet Maximization on Uber Dataset

Figure 3: Further Experimental Results. The first row reports the objective values, the second one the cumulative consistency.

Personalized Movie Recommendation. Movie recommendation systems are one of the common experiments in the context of submodular maximization (e.g., Amanatidis et al., 2021; Dütting et al., 2022; Halabi et al., 2023). In this experiment, we have a large collection M of movies that arrive online and we want to design a recommendation system that proposes movies to users. For example, the summary may be a carousel of 'recommended movies' presented to a downstream user, and we would like the selection to be fairly stable. We 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 (that is chosen as a random point in $[0,1]^{30}$ in our experiments) 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.