




Streaming Submodular Maximization Under Matroid Constraints

Moran Feldman   



University of Haifa, Israel

Paul Liu   

Stanford University, CA, USA

Ashkan Norouzi-Fard  

Google Research, Zurich, Switzerland

Ola Svensson  

EPFL, Lausanne, Switzerland

Rico Zenklusen  

ETH Zürich, Switzerland

Abstract

Recent progress in (semi-)streaming algorithms for monotone submodular function maximization has led to tight results for a simple cardinality constraint. However, current techniques fail to give a similar understanding for natural generalizations, including matroid constraints. This paper aims at closing this gap. For a single matroid of rank k (i.e., any solution has cardinality at most k), our main results are:

- A single-pass streaming algorithm that uses $\tilde{O}(k)$ memory and achieves an approximation guarantee of 0.3178.
- A multi-pass streaming algorithm that uses $\tilde{O}(k)$ memory and achieves an approximation guarantee of $(1 - 1/e - \varepsilon)$ by taking a constant (depending on ε) number of passes over the stream.

This improves on the previously best approximation guarantees of $1/4$ and $1/2$ for single-pass and multi-pass streaming algorithms, respectively. In fact, our multi-pass streaming algorithm is *tight* in that any algorithm with a better guarantee than $1/2$ must make several passes through the stream and any algorithm that beats our guarantee of $1 - 1/e$ must make linearly many passes (as well as an exponential number of value oracle queries).

Moreover, we show how the approach we use for multi-pass streaming can be further strengthened if the elements of the stream arrive in uniformly random order, implying an improved result for p -matchoid constraints.

2012 ACM Subject Classification Mathematics of computing \rightarrow Submodular optimization and polymatroids; Theory of computation \rightarrow Streaming models

Keywords and phrases Submodular maximization, streaming, matroid, random order

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.59

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version:* <https://arxiv.org/abs/2107.07183v2>

Funding *Moran Feldman:* Research supported in part by the Israel Science Foundation (ISF) grants no. 1357/16 and 459/20.



Ola Svensson: Research supported by the Swiss National Science Foundation project 200021-184656 “Randomness in Problem Instances and Randomized Algorithms.”



Rico Zenklusen: Research supported in part by Swiss National Science Foundation grant number 200021_184622. This project has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No 817750).



Acknowledgements The authors are very grateful to Jan Vondrák. It is safe to say that this paper would not be nearly as good without Jan’s many interesting discussions and comments.



© Moran Feldman, Paul Liu, Ashkan Norouzi-Fard, Ola Svensson, and Rico Zenklusen; licensed under Creative Commons License CC-BY 4.0

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).

Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;

Article No. 59; pp. 59:1–59:20



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



1 Introduction

Submodular function optimization is a classic topic in combinatorial optimization (see, e.g., the book [28]). Already in 1978, Nemhauser, Wolsey, and Fisher [26] analyzed a simple greedy algorithm for selecting the most valuable set $S \subseteq V$ of cardinality at most k . This algorithm starts with the empty set S , and then, for k steps, adds to S the element u with the largest marginal value. Assuming the submodular objective function f is also non-negative and monotone, they showed that the greedy algorithm returns a $(1 - 1/e)$ -approximate solution. Moreover, the approximation guarantee of $1 - 1/e$ is known to be tight [10, 25].

A natural generalization of a cardinality constraint is that of a matroid constraint. While a matroid constraint is much more expressive than a cardinality constraint, it has often been the case that further algorithmic developments have led to the same or similar guarantees for both types of constraints. Indeed, for the problem of maximizing a monotone submodular function subject to a matroid constraint, Călinescu, Chekuri, Pál, and Vondrák [5] developed the more advanced continuous greedy method, and showed that it recovers the guarantee $1 - 1/e$ in this more general setting. Since then, other methods, such as local search [14], have been developed to recover the same optimal approximation guarantee.

More recently, applications in data science and machine learning [19], with huge problem instances, have motivated the need for space-efficient algorithms, i.e., (semi-)streaming algorithms for (monotone) submodular function maximization. This is now a very active research area, and recent progress has resulted in a tight understanding of streaming algorithms for maximizing monotone submodular functions with a single cardinality constraint: the optimal approximation guarantee is $1/2$ for single-pass streaming algorithms, and it is possible to recover the guarantee $1 - 1/e - \varepsilon$ in $O_\varepsilon(1)$ passes. That it is impossible to improve upon $1/2$ in a single pass is due to [11], and the first single-pass streaming algorithm to achieve this guarantee is a simple “threshold” based algorithm [2] that, intuitively, selects elements with marginal value at least $\text{OPT}/(2k)$. The $(1 - 1/e - \varepsilon)$ guarantee in $O_\varepsilon(1)$ passes can be obtained using smart implementations of the greedy approach [3, 17, 22, 23, 27].

It is interesting to note that simple greedy and threshold-based algorithms have led to tight results for maximizing a monotone submodular function subject to a cardinality constraint in both the “offline” RAM and data stream models. However, in contrast to the RAM model, where more advanced algorithmic techniques have generalized these guarantees to much more general constraint families, current techniques fail to give a similar understanding in the data stream model, both for single-pass and multi-pass streaming algorithms. Closing this gap is the motivation for our work. In particular, current results leave open the intriguing possibility to obtain the same guarantees for a matroid constraint as for a cardinality constraint. Our results make significant progress on this question for single-pass streaming algorithms and completely close the gap for multi-pass streaming algorithms.

► **Theorem 1.** *There is a single-pass semi-streaming algorithm for maximizing a non-negative monotone submodular function subject to a matroid constraint of rank k (any solution has cardinality at most k) that stores $O(k)$ elements, requires $\tilde{O}(k)$ additional memory, and achieves an approximation guarantee of 0.3178.*

The last theorem improves upon the previous best approximation guarantee of $1/4 = 0.25$ [6]. Moreover, the techniques are versatile and also yield a single-pass streaming algorithm with an improved approximation guarantee for non-monotone functions (improving from 0.1715 [12] to 0.1921).

Our next result is a tight multi-pass guarantee of $1 - 1/e - \varepsilon$, improving upon the previously best guarantee of $1/2 - \varepsilon$ [18].

► **Theorem 2.** *For every constant $\varepsilon > 0$, there is a multi-pass semi-streaming algorithm for maximizing a non-negative monotone submodular function subject to a matroid constraint of rank k (any solution has cardinality at most k) that stores $O(k/\varepsilon)$ elements, makes $O(1/\varepsilon^3)$ many passes, and achieves an approximation guarantee of $1 - 1/e - \varepsilon$.*

The result is tight (up to the exact dependency on ε) in the following strong sense: any streaming algorithm with a better approximation guarantee than $1/2$ must make more than one pass [11], and any algorithm with a better guarantee than $1 - 1/e$ must make linearly (in the length of the stream) many passes [22] (see Section 6 for more detail).

The way we obtain Theorem 2 is through a rather general and versatile framework based on the “Accelerated Continuous Greedy” algorithm of [3], which was designed for the classic (non-streaming) setting. This allows us to obtain results with an improved number of passes or more general constraints in specific settings. First, if the elements of the stream arrive in uniformly random order, then we can improve the number of passes as stated below.

► **Theorem 3.** *If the elements arrive in an independently random order in each pass, then for every constant $\varepsilon > 0$, there is a multi-pass semi-streaming algorithm for maximizing a non-negative monotone submodular function subject to a matroid constraint of rank k (any solution has cardinality at most k) that stores $O(k/\varepsilon)$ elements, makes $O(\varepsilon^{-2} \log \varepsilon^{-1})$ many passes, and achieves an approximation guarantee of $1 - 1/e - \varepsilon$.*

Second, also in the uniformly random order model, we can obtain results with even fewer passes, and that also extend to p -matchoid constraints, but at the cost of weaker approximation guarantees.

► **Theorem 4.** *If the elements arrive in an independently random order in each pass, then for every constant $\varepsilon > 0$, there is a multi-pass semi-streaming algorithm for maximizing a non-negative monotone submodular function subject to a matroid constraint of rank k (any solution has cardinality at most k) that stores $O(k)$ elements, makes $O(\log \varepsilon^{-1})$ many passes, and achieves an approximation guarantee of $1/2 - \varepsilon$.*

Moreover, if the matroid constraint is replaced with a more general p -matchoid constraint, the above still holds except that now the approximation guarantee is $1/(p+1) - \varepsilon$ and the number of passes is $O(p^{-1} \log \varepsilon^{-1})$.

The p -matchoid result of Theorem 4 improves, in the random order model, over an algorithm of [18] that achieves the same approximation factor, but needs $O(p/\varepsilon)$ passes, whereas our algorithm requires a number of passes that only logarithmically depends on ε^{-1} and decreases (rather than increases) with p . (However, the procedure in [18] does not require random arrival order, and obtains its guarantees even in the adversarial arrival model.)

1.1 Our Technique

Before getting into the technical details of our approaches, we provide an overview of the main ingredients behind the techniques we employ.

Single pass algorithms

The 4-approximation single pass algorithm due to Chakrabarti and Kale [6] (and later algorithms based on it such as [8, 12]) maintains an integral solution in the following way. Whenever a new element u arrives, the algorithm considers inserting u into the solution at the expense of some element u' that gets removed from the solution; and this swap is performed if it is beneficial enough. Naturally, the decision to make the swap is a binary

decision: we either make the swap or we do not do that. The central new idea in our improved single pass algorithms (Theorem 1) is that we make the swap fractional. In other words, we start inserting fractions of u at the expense of fractions of u' (the identity of u' might be different for different fractions of u), and we continue to do that as long as the swap is beneficial enough. Since “beneficial enough” depends on properties of the current solution, the swapping might stop being beneficial enough before all of u is inserted into the solution, which explains why our fractional swapping does not behave like the integral swapping used by previous algorithms.

While our single pass algorithms are based on the above idea, they are presented in a slightly different way for simplicity of the presentation and analysis. In a nutshell, the differences can be summarized by the following two points.

- Instead of maintaining a fractional solution, we maintain multiple sets A_i (for $i \in \mathbb{Z}$). Membership of an element u in each of these sets corresponds to having a fraction of $1/m$ (for a parameter m of the algorithm) of u in the fractional solution.
- We do not remove elements from our fractional solution. Instead, we add new elements to sets A_i with larger and larger i indexes with the implicit view that only fractions corresponding to sets A_i with relatively large indices are considered part of the fractional solution.

To make the above points more concrete, we note that the fractional solution is reconstructed from the sets A_i according to the above principles at the very end of the execution of our algorithms. The reconstructed fractional solution is denoted by s in these algorithms.

Multi-pass algorithms

Badanidiyuru and Vondrák [3] described an algorithm called “Accelerated Continuous Greedy” that obtains $1 - 1/e - O(\varepsilon)$ approximation (for every $\varepsilon \in (0, 1)$) for maximizing a monotone submodular function subject to a matroid constraint. Even though their algorithm is not a data stream algorithm, it accesses the input only in a well-defined restricted way, namely through a procedure called “Decreasing-Threshold Procedure”. Originally, this procedure was implemented using a greedy algorithm on an altered objective function. However, we observe that the algorithm of [3] can work even if Decreasing-Threshold Procedure is modified to return any local maximum of the same altered objective function. Therefore, to get a multiple pass data stream algorithm, it suffices to design such an algorithm that produces an (approximate) local maximum (or a solution that is as good as such a local maximum); this algorithm can then be used as the implementation of Decreasing-Threshold Procedure. This is the framework we use to get our $(1 - 1/e - \varepsilon)$ -approximation algorithms.

To prove Theorem 2 using the above framework, we show that a known algorithm (a variant of the algorithm of Chakrabarti and Kale [6] due to Huang, Thiery, and Ward [18]) can be repurposed to produce an approximate local maximum using $O(\varepsilon^{-2})$ passes, which, when used in Accelerated Continuous Greedy, leads to the claimed $O(\varepsilon^{-3})$ many passes. Similarly, by adapting an algorithm of Shadravan [29] working in the random order model, and extending it to multiple passes, we are able to get a solution that is as good as an approximate local maximum in only $O(\varepsilon^{-1} \log \varepsilon^{-1})$ random-order passes, which leads to Theorem 3 when combined with the above framework.

Interestingly, any (approximate) local maximum also has an approximation guarantee of its own (without employing the above framework). This means that the above procedures for producing approximate local maxima can also be viewed as approximation algorithms

in their own right, which leads to Theorem 4.¹ It is important to note that Theorem 4 uses fewer passes than what is used in the proof of Theorem 3 to get a solution which is at least as good as an approximate local maximum. This discrepancy happens because in Theorem 4 we only aim for a solution with some approximation ratio r , where r is an approximation ratio guaranteed by any approximate local maximum in any instance. In contrast, Theorem 3 needs a solution that is as good as some approximate local maximum of the particular instance considered.

1.2 Additional Related Work

As mentioned above, Călinescu et al. [5] proposed a $(1 - 1/e)$ -approximation algorithm for maximizing a monotone submodular function subject to a matroid constraint in the offline (RAM) setting, which is known to be tight [10, 25]. The corresponding problem with a non-monotone objective is not as well understood. A long line of work [9, 13, 20] on this problem culminated in a 0.385-approximation due to Buchbinder and Feldman [4] and an upper bound by Oveis Gharan and Vondrák [15] of 0.478 on the best obtainable approximation ratio.

The first semi-streaming algorithm for maximizing a monotone submodular function subject to a matroid constraint was described by Chakrabarti and Kale [6], who obtained an approximation ratio of $1/4$ for the problem. This remained state-of-the-art prior to this work. However, Chan, Huang, Jiang, Kang, and Tang [7] managed to get an improved approximation ratio of 0.3178 for the special case of a partition matroid in the related preemptive online model. We note that the last approximation ratio is identical to the approximation ratio stated in Theorem 1, which points to some similarity that exists between the algorithms (in particular, both use fractional swaps). However, the algorithm of [7] is not a semi-streaming algorithm (and moreover, it is tailored to partition matroids). The first semi-streaming algorithm for the non-monotone version of the above problem was obtained by Chekuri, Gupta, and Quanrud [8], and achieved a $(1/(4 + e) - \epsilon) \approx 0.1488$ -approximation. This was later improved to 0.1715-approximation by Feldman, Karbasi, and Kazemi [12].²

Outline of the paper

In Section 2, we introduce notations and definitions used throughout this paper. Afterwards, in Section 3, we present and analyze our single-pass algorithms. The framework used to prove Theorems 2 and 3 is presented in detail in Section 4, and in the two sections after it we describe the algorithms for obtaining approximate local maxima (or equally good solutions) necessary for using this framework. Specifically, in Section 5 we show how to get such an algorithm for adversarial order streams (leading to Theorem 2), and in Section 7 we show how to get such an algorithm for random order streams (leading to Theorems 3 and 4). It is worth noting that Section 3 is independent of all the other sections, and therefore, can be skipped by a reader interested in the other parts of this paper.

¹ Technically, we can also get a result for adversarial order streams in this way, but we omit this result since it is weaker than a known result of [18].

² Mirzasoleiman et al. [24] claimed another approximation ratio for the problem (weaker than the one given later by [12]), but some problems were found in their analysis (see [16] for details).

2 Preliminaries

Recall that we are interested in the problem of maximizing a submodular function subject to a matroid constraint. In Section 2.1 we give the definitions necessary for formally stating this problem. Then, Section 2.2 defines the data stream model in which we study the problem. Finally, in Section 2.3 we present some additional notation and definitions that we use.

2.1 Problem Statement

Submodular Functions

Given a ground set \mathcal{N} , a *set function* $f: 2^{\mathcal{N}} \rightarrow \mathbb{R}$ is a function that assigns a numerical value to every subset of \mathcal{N} . Given a set $S \subseteq \mathcal{N}$ and an element $u \in \mathcal{N}$, it is useful to denote by $f(u | S)$ the marginal contribution of u to S with respect to f , i.e., $f(u | S) := f(S \cup \{u\}) - f(S)$. Similarly, we denote the marginal contribution of a set $T \subseteq \mathcal{N}$ to S with respect to f by $f(T | S) := f(S \cup T) - f(S)$.

A set function $f: 2^{\mathcal{N}} \rightarrow \mathbb{R}$ is called *submodular* if for any two sets S and T such that $S \subseteq T \subseteq \mathcal{N}$ and any element $u \in \mathcal{N} \setminus T$ we have $f(u | S) \geq f(u | T)$. Moreover, we say that f is *monotone* if $f(S_1) \leq f(S_2)$ for any sets $S_1 \subseteq S_2 \subseteq \mathcal{N}$, and f is *non-negative* if $f(S) \geq 0$ for every $S \subseteq \mathcal{N}$.

Matroids

A set system is a pair $M = (\mathcal{N}, \mathcal{I})$, where \mathcal{N} is a finite set called the *ground set*, and $\mathcal{I} \subseteq 2^{\mathcal{N}}$ is a collection of subsets of the ground set. We say that a set $S \subseteq \mathcal{N}$ is *independent* in M if it belongs to \mathcal{I} (otherwise, we say that it is a *dependent* set); and the rank of the set system M is defined as the maximum size of an independent set in it. A set system is a *matroid* if it has three properties: i) The empty set is independent, i.e., $\emptyset \in \mathcal{I}$. ii) Every subset of an independent set is independent, i.e., for any $S \subseteq T \subseteq \mathcal{N}$, if $T \in \mathcal{I}$ then $S \in \mathcal{I}$. iii) If $S \in \mathcal{I}$, $T \in \mathcal{I}$ and $|S| < |T|$, then there exists an element $u \in T \setminus S$ such that $S \cup \{u\} \in \mathcal{I}$.³

A matroid constraint is simply a constraint that allows only sets that are independent in a given matroid. Matroid constraints are of interest because they have a rich combinatorial structure and yet are able to capture many constraints of interest such as cardinality, independence of vectors in a vector space, and being a non-cyclic sub-graph.

Matchoids and p -matchoids

The matchoid notion (for the case of $p = 2$) was proposed by Jack Edmonds as a common generalization of matching and matroid intersection. Let $M_1 = (\mathcal{N}_1, \mathcal{I}_1)$, $M_2 = (\mathcal{N}_2, \mathcal{I}_2)$, \dots , $M_q = (\mathcal{N}_q, \mathcal{I}_q)$ be q matroids, and let $\mathcal{N} = \mathcal{N}_1 \cup \dots \cup \mathcal{N}_q$ and $\mathcal{I} = \{S \subseteq \mathcal{N} \mid S \cap \mathcal{N}_\ell \in \mathcal{I}_\ell \text{ for every } 1 \leq \ell \leq q\}$. The set system $M = (\mathcal{N}, \mathcal{I})$ is a *p -matchoid* if each element $u \in \mathcal{N}$ is a member of \mathcal{N}_ℓ for at most p indices $\ell \in [q]$. Informally, a p -matchoid is an intersection of matroids in which every particular element $u \in \mathcal{N}$ is affected by at most p matroids. It is easy to see that a 1-matchoid is just a matroid, and vice versa. 2-matchoids are often referred to simply as matchoids (without a parameter p).

³ The last property is often referred to as the *exchange axiom* of matroids.

Problem

In the **Submodular Maximization subject to a Matroid Constraint** problem (**SMMatroid**), we are given a non-negative⁴ submodular function $f: 2^{\mathcal{N}} \rightarrow \mathbb{R}_{\geq 0}$ and a matroid $M = (\mathcal{N}, \mathcal{I})$ over the same ground set. The objective is to find an independent set $S \in \mathcal{I}$ that maximizes f . An important special case of **SMMatroid** is the **Monotone Submodular Maximization subject to a Matroid Constraint** problem (**MSMMatroid**) in which we are guaranteed that the objective function f is monotone (in addition to being non-negative and submodular).

2.2 Data Stream Model

In the data stream model, the input appears in a sequential form known as the *input stream*, and the algorithm is allowed to read it only sequentially. In the context of our problem, the input stream consists of the elements of the ground set sorted in either an adversarially chosen order or a uniformly random order, and the algorithm is allowed to read the elements from the stream only in this order. Often the algorithm is allowed to read the input stream only once (such algorithms are called *single-pass* algorithms), but in other cases it makes sense to allow the algorithm to read the input stream multiple times – each such reading is called a *pass*. The order of the elements in each pass might be different; in particular, when the order is random, we assume that it is chosen independently for each pass.

A trivial way to deal with the restrictions of the data stream model is to store the entire input stream in the memory of the algorithm. However, we are often interested in a stream carrying too much data for this to be possible. Thus, the goal in this model is to find a high quality solution while using significantly less memory than what is necessary for storing the input stream. The gold standard are algorithms that use memory of size nearly linear in the maximum possible size of an output; such algorithms are called *semi-streaming* algorithms.⁵ For **SMMatroid** and **MSMMatroid**, this implies that a semi-streaming algorithm is a data stream algorithm that uses $O(k \log^{O(1)} |\mathcal{N}|)$ space, where k is the rank of the matroid constraint.

The description of submodular functions and matroids can be exponential in the size of their ground sets, and therefore, it is important to define the way in which an algorithm may access them. We make the standard assumption that the algorithm has two oracles: a *value oracle* and an *independence oracle* which, given a set $S \subseteq \mathcal{N}$ of elements that are explicitly *stored* in the memory of the algorithm, returns the value of $f(S)$ and an indicator whether $S \in \mathcal{I}$, respectively.

2.3 Additional Notation and Definitions

Multilinear Extension

A set function $f: 2^{\mathcal{N}} \rightarrow \mathbb{R}$ assigns values only to subsets of \mathcal{N} . If we think of a set S as equivalent to its characteristic vector $\mathbf{1}_S$ (a vector in $\{0, 1\}^{\mathcal{N}}$ that has a value of 1 in every coordinate $u \in S$ and a value of 0 in the other coordinates), then we can view f as a function over the integral vectors in $[0, 1]^{\mathcal{N}}$. It is often useful to extend f to general

⁴ The assumption of non-negativity is necessary to allow multiplicative approximation guarantees.

⁵ The similar term *streaming* algorithms often refers to algorithms whose space complexity is poly-logarithmic in the parameters of their input. Such algorithms are irrelevant for the problem we consider because they do not have enough space even for storing the output of the algorithm.

vectors in $[0, 1]^{\mathcal{N}}$. There are multiple natural ways to do that. However, in this paper, we only need the multilinear extension F . Given a vector $x \in [0, 1]^{\mathcal{N}}$, let $\mathcal{R}(x)$ denote a random subset of \mathcal{N} including each element $u \in \mathcal{N}$ with probability x_u , independently. Then, $F(x) = \mathbb{E}[f(\mathcal{R}(x))] = \sum_{S \subseteq \mathcal{N}} [f(S) \cdot \prod_{u \in S} x_u \cdot \prod_{u \notin S} (1 - x_u)]$. One can observe that, as is implied by its name, the multilinear extension is a multilinear function. Thus, for every vector $x \in [0, 1]^{\mathcal{N}}$, the partial derivative $\frac{\partial F}{\partial x_u}(x)$ is equal to $F(x + (1 - x_u) \cdot \mathbf{1}_u) - F(x - x_u \cdot \mathbf{1}_u)$. Note that in the last expression we have used $\mathbf{1}_u$ as a shorthand for $\mathbf{1}_{\{u\}}$. We often also use $\partial_u F(x)$ as a shorthand for $\frac{\partial F}{\partial x_u}(x)$. When f is submodular, its multilinear extension F is known to be concave along non-negative directions [5].

General Notation

Given a set $S \subseteq \mathcal{N}$ and an element $u \in \mathcal{N}$, we denote by $S + u$ and $S - u$ the expressions $S \cup \{u\}$ and $S \setminus \{u\}$, respectively. Additionally, given two vectors $x, y \in [0, 1]^{\mathcal{N}}$, we denote by $x \vee y$ and $x \wedge y$ the coordinate-wise maximum and minimum operations, respectively.

Additional Definitions from Matroid Theory

Matroid theory is extensive, and we refer the reader to [28] for a more complete coverage of it. Here, we give only a few basic definitions from this theory that we employ below. Given a matroid $M = (\mathcal{N}, \mathcal{I})$, a set $S \subseteq \mathcal{N}$ is called *base* if it is an independent set that is maximal with respect to inclusion (i.e., every super-set of S is dependent), and it is called *cycle* if it is a dependent set that is minimal with respect to inclusion (i.e., every subset of S is independent). An element $u \in \mathcal{N}$ is called a loop if $\{u\}$ is a cycle. Notice that such elements cannot appear in any feasible solution for either **SMMatroid** or **MSMMatroid**, and therefore, one can assume without loss of generality that there are no loops in the ground set.

The rank of a set $S \subseteq \mathcal{N}$, denoted by $\text{rank}_M(S)$, is the maximum size of an independent set $T \in \mathcal{I}$ which is a subset of S . The subscript M is omitted when it is clear from the context. We also note that $\text{rank}_M(\mathcal{N})$ is exactly the rank of the matroid M (i.e., the maximum size of an independent set in M), and therefore, it is customary to define $\text{rank}(M) = \text{rank}_M(\mathcal{N})$. We say that a set $S \subseteq \mathcal{N}$ spans an element $u \in \mathcal{N}$ if adding u to S does not increase the rank of the set S , i.e., $\text{rank}(S) = \text{rank}(S + u)$ – observe the analogy between this definition and being spanned in a vector space. Furthermore, we denote by $\text{span}_M(S) := \{u \in \mathcal{N} \mid \text{rank}(S) = \text{rank}(S + u)\}$ the set of elements that are spanned by S . Again, the subscript M is dropped when it is clear from the context.

3 Single-Pass Algorithm

In this section, we present our single-pass algorithm for the **Monotone Submodular Maximization subject to a Matroid Constraint** problem (**MSMMatroid**). The properties of the algorithm we present are given by the following theorem.

► **Theorem 1.** *There is a single-pass semi-streaming algorithm for maximizing a non-negative monotone submodular function subject to a matroid constraint of rank k that stores $O(k)$ elements, requires $\tilde{O}(k)$ additional memory, and achieves an approximation guarantee of 0.3178.*

Our algorithm can be extended to the case in which the objective function is non-monotone (i.e., the **SMMatroid** problem) at the cost of obtaining a lower approximation factor, yielding the following theorem. However, for the sake of concentrating on our main new ideas, we devote this section to the algorithm of Theorem 1 and defer the proof of Theorem 5 to the full version.

► **Theorem 5.** *There is a single-pass semi-streaming algorithm for maximizing a non-negative (not necessarily monotone) submodular function subject to a matroid constraint of rank k that stores $O(k)$ elements, requires $\tilde{O}(k)$ additional memory, and achieves an approximation guarantee of 0.1921.*

Throughout this section, we denote by $P_M := \{x \in \mathbb{R}_{\geq 0}^{\mathcal{N}} : x(S) \leq \text{rank}(S) \ \forall S \subseteq \mathcal{N}\}$ the matroid polytope of M . The algorithm we use to prove Theorem 1 appears as Algorithm 1. This algorithm gets a parameter $\varepsilon > 0$ and starts by initializing a constant α to be approximately the single positive value obeying $\alpha + 2 = e^\alpha$. We later prove that the approximation ratio guaranteed by the algorithm is at least $\frac{1}{\alpha+2} - \varepsilon$, which is better than the approximation ratio stated in Theorem 1 for a small enough ε . After setting the value of α , Algorithm 1 defines some additional constants m , c , and L using ε and α . We leave these variables representing different constants as such in the procedure and analysis, which allows for obtaining a better understanding later on of why these values are optimal for our analysis. We also note that, as stated, Algorithm 1 is efficient (i.e., runs in polynomial time) only if the multilinear extension and its partial derivatives can be efficiently evaluated. If that cannot be done, then one has to approximate F and its derivatives using Monte-Carlo simulation, which is standard practice (see, for example, [5]). We omit the details to keep the presentation simple, but we note that, as in other applications of this standard technique, the incurred error can easily be kept negligible, and therefore, does not affect the guarantee stated in Theorem 1.

Algorithm 1 uses sets A_i and vectors $a_i \in [0, 1]^{\mathcal{N}}$ for certain indices $i \in \mathbb{Z}$. Throughout the algorithm, we only consider finitely many indices $i \in \mathbb{Z}$. However, we do not know upfront which indices within \mathbb{Z} we will use. To simplify the presentation, we therefore use the convention that whenever the algorithm uses for the first time a set A_i or vector a_i , then A_i is initialized to be \emptyset and a_i is initialized to be the zero vector. The largest index ever used in the algorithm is q , which is computed toward the end of the algorithm at Line 13.

For each $i \in \mathbb{Z}$, the set A_i is an independent set consisting of elements u that already arrived and for which the marginal increase with respect to a reference vector a (at the moment when u arrives) is at least c^i . More precisely, whenever a new element $u \in \mathcal{N}$ arrives and its marginal return $\partial_u F(a)$ exceeds c^i for an index $i \in \mathbb{Z}$ in a relevant range, then we add u to A_i if $A_i + u$ remains independent. When adding u to A_i , we also increase the u -entry of the vector a_i by $\frac{c^i}{m \cdot \partial_u F(a)}$. The vector a built up during the algorithm has two key properties. First, its multilinear value approximates $f(\text{OPT})$ up to a constant factor. Second, one can derive from the sets A_i a vector s (see Algorithm 1) such that $F(s)$ is close to $F(a)$ and s is contained in the matroid polytope P_M .

Whenever an element $u \in \mathcal{N}$ arrives, the algorithm first computes the largest index $i(u) \in \mathbb{Z}$ fulfilling $c^{i(u)} \leq \partial_u F(a)$. It then updates sets A_i and vectors a_i for indices $i \leq i(u)$. Purely conceptually, the output of the algorithm would have the desired guarantees even if all infinitely many indices below $i(u)$ were updated. However, to obtain an algorithm running in finite (even polynomial) time and linear memory, we do not consider indices below $\max\{b, i(u) - \text{rank}(M) - L\}$ in the update step. Capping the considered indices like this has only a minor impact in the analysis since the contribution of the vectors a_i to the multilinear extension value of the vector a is geometrically decreasing with decreasing index i .

In the algorithm, and also in its analysis, we sometimes use sums over indices that go up to ∞ . However, whenever this happens, beyond some finite index, all terms are zero. Hence, such sums are well defined.

Finally, we provide details on the return statement in Line 17 of the algorithm. This statement is based on a fact stated in [5], namely that a point in the matroid polytope can be rounded losslessly to an independent set. More formally, given any point $y \in P_M$ in the

■ **Algorithm 1** Single-Pass Semi-Streaming Algorithm for **MSMMatroid**.

-
- 1: Set $\alpha = 1.1462$, $m = \lceil \frac{3\alpha}{\varepsilon} \rceil$, $c = \frac{m}{m-\alpha}$, and $L = \lceil \log_c(\frac{2c}{\varepsilon(c-1)}) \rceil$.
 - 2: Set $a = 0 \in [0, 1]^{\mathcal{N}}$ to be the zero vector, and let $b = -\infty$.
 - 3: **for** every element arriving $u \in \mathcal{N}$, if $\partial_u F(a) > 0$ **do**
 - 4: Let $i(u) = \lfloor \log_c(\partial_u F(a)) \rfloor$. ▷ Thus, $i(u)$ is largest index $i \in \mathbb{Z}$ with $c^i \leq \partial_u F(a)$.
 - 5: **for** $i = \max\{b, i(u) - \text{rank}(M) - L\}$ **to** $i(u)$ **do**
 - 6: **if** $A_i + u \in \mathcal{I}$ **then**
 - 7: $A_i \leftarrow A_i + u$.
 - 8: $a_i \leftarrow a_i + \frac{c^i}{m \cdot \partial_u F(a)} \mathbf{1}_u$.
 - 9: Set $b \leftarrow h - L$, where h is largest index $i \in \mathbb{Z}$ satisfying $\sum_{j=i}^{\infty} |A_j| \geq \text{rank}(M)$.
 - 10: $a \leftarrow \sum_{i=b}^{\infty} a_i$.
 - 11: Delete from memory all sets A_i and vectors a_i with $i \in \mathbb{Z}_{<b}$.
 - 12: Set $S_k \leftarrow \emptyset$ for $k \in \{0, \dots, m-1\}$.
 - 13: Let q be largest index $i \in \mathbb{Z}$ with $A_i \neq \emptyset$.
 - 14: **for** $i = q$ **to** b (stepping down by 1 at each iteration) **do**
 - 15: **while** $\exists u \in A_i \setminus S_{(i \bmod m)}$ with $S_{(i \bmod m)} + u \in \mathcal{I}$ **do**
 - 16: $S_{(i \bmod m)} \leftarrow S_{(i \bmod m)} + u$.
 - 17: **return** a rounding $R \in \mathcal{I}$ of the fractional solution $s := \frac{1}{m} \sum_{k=0}^{m-1} \mathbf{1}_{S_k}$ with $f(R) \geq F(s)$.
-

matroid polytope, there is an independent set $I \in \mathcal{I}$ with $f(I) \geq F(y)$. Moreover, assuming that the multilinear extension F can be evaluated efficiently, such an independent set I can be computed efficiently. As before, if one is only given a value oracle for f , then the exact evaluation of F can be replaced by a strong estimate obtained through Monte-Carlo sampling, leading to a randomized algorithm to round y to an independent set I with $f(I) \geq (1 - \delta)F(y)$ for an arbitrarily small constant $\delta > 0$.

Due to space constraints, the analysis of Algorithm 1 is deferred to the full version.

4 Framework for Multi-pass Algorithms

In this section we present the details of the framework used to prove our $(1 - 1/e)$ -approximation results (Theorems 2 and 3). We remind the reader that the proofs of these theorems (using the framework) can be found in Sections 5 and 7, respectively. Badanidiyuru and Vondrák [3] described an algorithm called “Accelerated Continuous Greedy” that obtains an approximation guarantee of $1 - 1/e - O(\varepsilon)$ for **MSMMatroid** for every $\varepsilon \in (0, 1)$. Their algorithm is not a data stream algorithm, but it enjoys the following nice properties.

- The algorithm includes a procedure called “Decreasing-Threshold Procedure”. This procedure is the only part of the algorithm that directly accesses the input.
- The Decreasing-Threshold Procedure is called $O(\varepsilon^{-1})$ times during the execution of the algorithm.
- In addition to the space used by this procedure, Accelerated Continuous Greedy uses only space that is linear in the space necessary to store the outputs of the various executions of the Decreasing-Threshold Procedure.
- The Decreasing-Threshold Procedure returns a base D of M after every execution, and this base is guaranteed to obey Equation (1) stated below. The analysis of the approximation ratio of Accelerated Continuous Greedy treats Decreasing-Threshold Procedure as a black box except for the fact that its output is a base D of M obeying Equation (1),

and therefore, this analysis will remain valid even if Decreasing-Threshold Procedure is replaced by any other algorithm with the same guarantee. Furthermore, one can verify that the analysis continues to work (with only minor technical changes) even if the output D of the replacing algorithm obeys Equation (1) only in expectation.

Let us now formally state the property that the output base of Decreasing-Threshold Procedure obeys. Let P_M be the matroid polytope of M , and let F be the multilinear extension of f . Decreasing-Threshold Procedure gets as input a point $x \in (1 - \varepsilon) \cdot P_M$, and its output base D is guaranteed to obey

$$F(x') - F(x) \geq \varepsilon[(1 - 3\varepsilon) \cdot f(\text{OPT}) - F(x')] , \quad (1)$$

where $x' = x + \varepsilon \cdot \mathbf{1}_D$ and OPT denotes an optimal solution.

Our objective in Sections 5 and 7 is to describe semi-streaming algorithms that can function as replacements for the offline procedure Decreasing-Threshold Procedure. The next proposition states that plugging such a replacement into Accelerated Continuous Greedy yields a roughly $(1 - 1/e)$ -approximation semi-streaming algorithm.

► **Proposition 6.** *Assume there exists a semi-streaming algorithm that given a point $x \in (1 - \varepsilon) \cdot P_M$ makes p passes over the input stream, stores $O(k/\varepsilon)$ elements, and outputs a base D obeying Equation (1) in expectation. Then, there exists a semi-streaming algorithm for maximizing a non-negative monotone submodular function subject to a matroid constraint of rank k that stores $O(k/\varepsilon)$ elements, makes $O(p/\varepsilon)$ many passes and achieves an approximation guarantee of $1 - 1/e - \varepsilon$.*

Proof. Observe that the proposition is trivial when $\varepsilon \geq 1 - 1/e$, and therefore, we assume below that $\varepsilon < 1 - 1/e$. Furthermore, for simplicity, we describe an algorithm with an approximation ratio of $1 - 1/e - O(\varepsilon)$ rather than a clean ratio of $1 - 1/e - \varepsilon$. However, one can switch between the two ratios by scaling ε by an appropriate constant.

Let us denote by ALG the algorithm whose existence is promised by the statement of the proposition, and consider an algorithm called **Data Stream Continuous Greedy (DSCG)** obtained from the Accelerated Continuous Greedy algorithm of [3] when every execution of the Decreasing-Threshold Procedure by the last algorithm is replaced with an execution of ALG . We explain below why **DSCG** has all the properties guaranteed by the proposition. We begin by recalling that since the approximation ratio analysis of Accelerated Continuous Greedy in [3] treats the Decreasing-Threshold Procedure as a black box that in expectation has the guarantee stated in Equation (1), and ALG also has this guarantee, this analysis can be applied as is also to **DSCG**, and therefore, **DSCG** is a $(1 - 1/e - O(\varepsilon))$ -approximation algorithm.

Recall now that Accelerated Continuous Greedy accesses its input only through the Decreasing-Threshold Procedure, which implies that **DSCG** is a data stream algorithm just like ALG . Furthermore, since Accelerated Continuous Greedy accesses the Decreasing-Threshold Procedure $O(\varepsilon^{-1})$ times, the number of passes used by **DSCG** is larger by a factor of $O(\varepsilon^{-1})$ compared to the number of passes used by ALG (which is denoted by p). Hence, **DSCG** uses $O(p/\varepsilon)$ passes.

It remains to analyze the space complexity of **DSCG**. Since Accelerated Continuous Greedy uses space of size linear in the space necessary to keep the $O(\varepsilon^{-1})$ bases that it receives from the Decreasing-Threshold Procedure, the space complexity of **DSCG** is larger than the space complexity of the semi-streaming algorithm ALG only by an additive term of $\tilde{O}(k/\varepsilon)$. As this term is nearly linear in k for any constant ε , we get that **DSCG** has a low enough space

59:12 Streaming Submodular Maximization Under Matroid Constraints

complexity to be a semi-streaming algorithm. Furthermore, since the $O(\varepsilon^{-1})$ bases that DSCG gets from ALG can include only $O(k/\varepsilon)$ elements, this expression upper bounds the number of elements stored by DSCG in addition to the $O(k/\varepsilon)$ elements stored by ALG itself. ◀

It turns out that one natural way to get a base D obeying Equation (1) is to output a local maximum with respect to the objective function $g(S) = F(x + \varepsilon \cdot \mathbf{1}_S)$ (i.e., a base D whose value with respect to this objective cannot be improved by replacing an element of D with an element of $\mathcal{N} \setminus D$). Getting such a maximum using a semi-streaming algorithm with a reasonable number of passes is challenging; however, one can define weaker properties that still allow us to get Equation (1). Specifically, for any $\varepsilon \in (0, 1)$, we say that a set D is an ε -approximate local maximum with respect to g if

$$g(D \mid \emptyset) \geq g(B \mid D) + \sum_{u \in B \cap D} g(u \mid D - u) - \varepsilon \cdot g(\text{OPT}_g \mid \emptyset)$$

for every base B of M , where OPT_g is a base maximizing g . (Intuitively, one should think of B as being the optimal solution with respect to f .)

One property of an approximate local maximum is that its value (with respect to g) is an approximation to $g(\text{OPT}_g)$.

► **Observation 7.** *For every $\varepsilon \in (0, 1)$, if D is an ε -approximate local maximum with respect to g , then $g(D) \geq \frac{1-\varepsilon}{2} \cdot g(\text{OPT}_g)$.*

Proof. One can verify that the non-negativity, monotonicity and submodularity of f implies that g also has these properties. Therefore,

$$\begin{aligned} g(D) &\geq g(D \mid \emptyset) \geq g(\text{OPT}_g \mid D) + \sum_{u \in \text{OPT}_g \cap D} g(u \mid D - u) - \varepsilon \cdot g(\text{OPT}_g \mid \emptyset) \\ &\geq g(\text{OPT}_g \mid D) - \varepsilon \cdot g(\text{OPT}_g \mid \emptyset) \geq (1 - \varepsilon) \cdot g(\text{OPT}_g) - g(D) , \end{aligned}$$

where the first inequality holds by the non-negativity of g , the second inequality follows from the fact that D is an ε -approximate local maximum (for $B = \text{OPT}_g$), the third inequalities follow from the monotonicity of g , and the last inequality hold by g 's non-negativity and monotonicity. Rearranging the above inequality now yields the observation. ◀

Using the last observation we can prove that any approximate local maximum with respect to g obeys Equation (1), and the same holds also for any solution that is almost as good as some approximate local maximum.

► **Lemma 8.** *For every $\varepsilon \in (0, 1)$, if D' is an ε -approximate local maximum with respect to g , then any (possibly randomized) set D such that $\mathbb{E}[g(D \mid \emptyset)] \geq (1 - \varepsilon) \cdot g(D' \mid \emptyset)$ obeys Equation (1) in expectation. In particular, this is the case for $D = D'$ since the monotonicity of f implies that g is non-negative.*

Proof. We need to consider two cases. The simpler case is when $g(\text{OPT}_g \mid \emptyset) \geq 2\varepsilon \cdot f(\text{OPT})$, where we recall that OPT is an optimal base with respect to f . Since $x' = x + \varepsilon \cdot \mathbf{1}_D$ by definition, we get in this case

$$\begin{aligned} \mathbb{E}[F(x')] - F(x) &= \mathbb{E}[F(x + \varepsilon \cdot \mathbf{1}_D)] - F(x) = \mathbb{E}[g(D \mid \emptyset)] \geq (1 - \varepsilon) \cdot g(D' \mid \emptyset) \\ &\geq \frac{(1-\varepsilon)^2}{2} g(\text{OPT}_g \mid \emptyset) \geq \varepsilon(1 - 2\varepsilon) \cdot f(\text{OPT}) \geq \varepsilon((1 - 3\varepsilon) \cdot f(\text{OPT}) - \mathbb{E}[F(x')]) , \end{aligned}$$

where the second inequality holds by Observation 7.

In the rest of the proof we consider the case of $g(\text{OPT}_g \mid \emptyset) \leq 2\varepsilon \cdot f(\text{OPT})$. We note that, in this case,

$$\begin{aligned} \frac{\mathbb{E}[F(x')] - F(x)}{1 - \varepsilon} &= \frac{\mathbb{E}[F(x + \varepsilon \cdot \mathbf{1}_D)] - F(x)}{1 - \varepsilon} = \frac{\mathbb{E}[g(D \mid \emptyset)]}{1 - \varepsilon} \geq g(D' \mid \emptyset) \\ &\geq g(\text{OPT} \mid D') + \sum_{u \in \text{OPT} \cap D'} g(u \mid D' - u) - \varepsilon \cdot g(\text{OPT}_g \mid \emptyset) \\ &\geq g(\text{OPT} \mid D') + \sum_{u \in \text{OPT} \cap D'} g(u \mid D' - u) - 2\varepsilon^2 \cdot f(\text{OPT}) , \end{aligned}$$

where the second inequality holds since D' is an ε -approximate local maximum (by plugging $B = \text{OPT}$ into the definition of such maxima). Let us now further develop the first two terms on the rightmost side of the last inequality. By the submodularity and monotonicity of f , if we denote $y = x + \varepsilon \cdot \mathbf{1}_{D'}$, then

$$\begin{aligned} &g(\text{OPT} \mid D') + \sum_{u \in \text{OPT} \cap D'} g(u \mid D' - u) \\ &= F(x + \varepsilon \cdot \mathbf{1}_{\text{OPT} \cup D'}) - F(x + \varepsilon \cdot \mathbf{1}_{D'}) + \sum_{u \in \text{OPT} \cap D'} [F(x + \varepsilon \cdot \mathbf{1}_{D'}) - F(x + \varepsilon \cdot \mathbf{1}_{D' - u})] \\ &\geq F(y + \varepsilon \cdot \mathbf{1}_{\text{OPT} \setminus D'}) - F(y) + \sum_{u \in \text{OPT} \cap D'} [F((y + \varepsilon \cdot \mathbf{1}_{\{u\}}) \wedge \mathbf{1}_N) - F(y)] \\ &\geq F((y + \varepsilon \cdot \mathbf{1}_{\text{OPT}}) \wedge \mathbf{1}_N) - F(y) . \end{aligned}$$

Combining the last two inequalities yields

$$\begin{aligned} \mathbb{E}[F(x')] - F(x) &\geq (1 - \varepsilon)[F((y + \varepsilon \cdot \mathbf{1}_{\text{OPT}}) \wedge \mathbf{1}_N) - F(y)] - 2\varepsilon^2 \cdot f(\text{OPT}) \\ &\geq (1 - \varepsilon)[F(y + \varepsilon((\mathbf{1}_N - y) \wedge \mathbf{1}_{\text{OPT}})) - F(y)] - 2\varepsilon^2 \cdot f(\text{OPT}) \\ &\geq \varepsilon(1 - \varepsilon)[F(y \vee \mathbf{1}_{\text{OPT}}) - F(y)] - 2\varepsilon^2 \cdot f(\text{OPT}) \\ &\geq \varepsilon((1 - \varepsilon)f(\text{OPT}) - \mathbb{E}[F(x')]) - 2\varepsilon^2 \cdot f(\text{OPT}) \\ &= \varepsilon \cdot ((1 - 3\varepsilon)f(\text{OPT}) - \mathbb{E}[F(x')]) , \end{aligned}$$

where the second inequality holds by the monotonicity of f , the third inequality holds because the submodularity of f guarantees that F is concave along non-negative directions (such as $(\mathbf{1}_N - y) \wedge \mathbf{1}_{\text{OPT}}$) and the last inequality holds by the monotonicity of f and the observation that

$$F(y) = g(D') = g(\emptyset) + g(D' \mid \emptyset) \leq g(\emptyset) + \frac{\mathbb{E}[g(D \mid \emptyset)]}{1 - \varepsilon} \leq \frac{\mathbb{E}[g(D)]}{1 - \varepsilon} = \frac{\mathbb{E}[F(x')]}{1 - \varepsilon} . \quad \blacktriangleleft$$

In Section 5 we describe a semi-streaming algorithm that can be used to find an ε -approximate local maximum of a non-negative monotone submodular function. By applying this algorithms to g , we get (via Lemma 8) an algorithm having all the properties assumed by Proposition 6; which proves Theorem 2. In Section 7 we attempt to use the same approach to get a result for random order streams. However, in this setting we are not able to guarantee an ε -approximate local maximum. Instead, we design an algorithm whose output has in expectation a value that is almost as good as the value of the worst approximate local maximum. This leads to a proof of Theorem 3.

5 Approximate Local Maximum for Adversarial Streams

In this section we prove following proposition, which guarantees the existence of a semi-streaming multi-pass algorithm for finding an ε -approximate local maximum in adversarial streams, i.e., when the order of the elements in the input stream is arbitrary. We note that this section highly depends on Section 4, and should not be read before that section.

► **Proposition 9.** *For every constant $\varepsilon > 0$, there is a multi-pass semi-streaming algorithm that given an instance of **MSMMatroid** with a matroid of rank k stores $O(k)$ elements, makes $O(\varepsilon^{-2})$ many passes, and outputs an ε -approximate local maximum.*

By Lemma 8 and Proposition 6, the last proposition implies Theorem 2. Therefore, we concentrate in this section on proving Proposition 9. The first data stream algorithm for **MSMMatroid** was described by Chakrabarti and Kale [6]. The first step towards proving Proposition 9 is a re-analysis of a variant of this algorithm that was described by Huang, Thiery and Ward [18] (based on ideas of Chekuri et al. [8]). The following proposition summarizes the properties of this variant that we prove in this re-analysis. Due to space constraints, the proof of this proposition is deferred to the full version.

► **Proposition 10.** *There exists a single-pass semi-streaming algorithm that given a base S_0 of M and value $c > 1$ outputs a base S_n that obeys $(c-1) \cdot f(S_n | \emptyset) + \frac{3c-2}{c-1} [f(S_n) - f(S_0)] \geq f(B | S_0 \setminus B) - f(S_0 | \emptyset) \geq f(B | S_0) + \sum_{u \in B \cap S_0} f(u | S_0 - u) - f(S_0 | \emptyset)$ for every base B of M . Furthermore, this algorithm stores $O(k)$ elements at any point during its execution.*

Below we refer to the algorithm whose existence is guaranteed by Proposition 10 as **SinglePass**. Next, we would like to show that **SinglePass** can be used to get an ε -approximate local maximum. The algorithm we use to do that is given as Algorithm 2, and it gets $\varepsilon \in (0, 1)$ as a parameter.

■ **Algorithm 2** MULTIPLE LOCAL SEARCH PASSES (ε).

-
- 1: Find a base T_0 of M using a single pass (by simply initializing T_0 to be the empty set, and then adding to it any elements that arrives and can be added to T_0 without violating independence in M).
 - 2: Let T_1 be the output of **SinglePass** when given $S_0 = T_0$ and $c = 2$.
 - 3: **for** $i = 2$ **to** $2 + \lceil 40\varepsilon^{-2} \rceil$ **do**
 - 4: Let T_i be the output of **SinglePass** when given $S_0 = T_{i-1}$ and $c = 1 + \varepsilon/2$.
 - 5: **if** $f(T_i) - f(T_{i-1}) \leq \varepsilon^2/10 \cdot f(T_1 | \emptyset)$ **then**
 - 6: **return** T_{i-1} .
 - 7: Indicate failure if the execution of the algorithm has arrived to this point.
-

Intuitively, Algorithm 2 works by employing the fact that every execution of **SinglePass** increases the value of its input base T_{i-1} significantly, unless this input base is close to being a local maximum, and therefore, if the execution produces a base T_i which is not much better than T_{i-1} , then we know that T_{i-1} is an ε -approximate local maximum. The following lemma proves this formally.

► **Lemma 11.** *If Algorithm 2 does not indicate a failure, then its output set T obeys $f(B | T) + \sum_{u \in B \cap T} f(u | T - u) - f(T | \emptyset) < \varepsilon \cdot f(\text{OPT} | \emptyset)$ for every base B of M . Note that the last inequality implies that T is an ε -approximate local maximum with respect to f .*

Proof. Since T_1 is a base of M , $f(T_1 | \emptyset) = f(T_1) - f(\emptyset) \leq f(\text{OPT}) - f(\emptyset) = f(\text{OPT} | \emptyset)$. This implies that when Algorithm 2 returns a set T_{i-1} , then

$$f(T_i) - f(T_{i-1}) \leq (\varepsilon^2/10) \cdot f(\text{OPT} | \emptyset) .$$

Plugging this inequality and the fact that $f(T_i | \emptyset) \leq f(\text{OPT} | \emptyset)$ (because T_i is a base of M) into the guarantee of Proposition 10 for the execution of **SinglePass** that has created T_i yields

$$\begin{aligned}
\varepsilon \cdot f(\text{OPT} \mid \emptyset) &\geq (\varepsilon/2) \cdot f(\text{OPT} \mid \emptyset) + \varepsilon(3\varepsilon/2 + 1)/5 \cdot f(\text{OPT} \mid \emptyset) \\
&\geq (\varepsilon/2) \cdot f(T_i \mid \emptyset) + \frac{3\varepsilon/2+1}{\varepsilon/2} \cdot [f(T_i) - f(T_{i-1})] \\
&\geq f(B \mid T_{i-1}) + \sum_{u \in B \cap T_{i-1}} f(u \mid T_{i-1} - u) - f(T_{i-1} \mid \emptyset) . \quad \blacktriangleleft
\end{aligned}$$

One could imagine that it is possible for the value of the solution maintained by Algorithm 2 to increase significantly following every iteration of the loop starting on Line 3, which will result in the algorithm indicating failure rather than ever returning a solution. However, it turns out that this cannot happen because the value of the solution of Algorithm 2 cannot exceed $f(\text{OPT})$, which implies a bound on the number of times this value can be increased significantly. This idea is formalized by the next two claims.

► **Observation 12.** $f(T_1 \mid \emptyset) \geq \frac{1}{5}f(\text{OPT} \mid \emptyset)$.

Proof. If we set $B = \text{OPT}$, then by applying Proposition 10 to the execution of `SinglePass` on Line 2 of Algorithm 2, we get

$$\begin{aligned}
f(T_1 \mid \emptyset) + 4[f(T_1) - f(T_0)] &\geq f(\text{OPT} \mid T_0) + \sum_{u \in \text{OPT} \cap T_0} f(u \mid T_0 - u) - f(T_0 \mid \emptyset) \\
&\geq f(\text{OPT} \mid T_0) - f(T_0 \mid \emptyset) ,
\end{aligned}$$

where the second inequality follows from the monotonicity of f . Since the leftmost side the last inequality is equal to $5f(T_1 \mid \emptyset) - 4f(T_0 \mid \emptyset)$, this inequality implies

$$\begin{aligned}
5f(T_1 \mid \emptyset) &\geq f(\text{OPT} \mid T_0) + 3f(T_0 \mid \emptyset) = f(\text{OPT} \cup T_0) + 2f(T_0) - 3f(\emptyset) \\
&\geq f(\text{OPT}) - f(\emptyset) = f(\text{OPT} \mid \emptyset) ,
\end{aligned}$$

where the second inequality follows again from the monotonicity of f . The observation now follows by dividing the last inequality by 5. ◀

► **Lemma 13.** *Algorithm 2 never indicates failure.*

Proof. If $f(\text{OPT} \mid \emptyset) = 0$, then the value of every base of M according to f is $f(\emptyset)$, which guarantees that Algorithm 2 returns T_1 during the first iteration of the loop starting on its Algorithm 2. Therefore, we assume below that $f(\text{OPT} \mid \emptyset) > 0$. Furthermore, assume towards a contradiction that Algorithm 2 indicates failure. By Observation 12, this assumption implies that the value of the solution maintained by Algorithm 2 increases by at least $(\varepsilon^2/10) \cdot f(T_1 \mid \emptyset) \geq \frac{\varepsilon^2}{50}f(\text{OPT} \mid \emptyset)$ after every iteration of the loop starting on Line 3. Therefore, after all the $1 + \lceil 40\varepsilon^{-2} \rceil$ iterations of this loop, the value of the solution of Algorithm 2 is at least

$$f(T_1) + (1 + \lceil 40\varepsilon^{-2} \rceil) \cdot \frac{\varepsilon^2}{50}f(\text{OPT} \mid \emptyset) > f(\emptyset) + \frac{1}{5}f(\text{OPT} \mid \emptyset) + \frac{4}{5}f(\text{OPT} \mid \emptyset) = f(\text{OPT}) ,$$

which is a contradiction since the solution of Algorithm 2 is always kept as a base of M . ◀

We now observe that Algorithm 2 has all the properties guaranteed by Proposition 9. In particular, we note that Algorithm 2 can be implemented as a semi-streaming algorithm storing $O(k)$ elements because it needs to store at most two solutions at any given time in addition to the elements and space required by `SinglePass`.

6 Discussion of a Lower Bound by McGregor and Vu [22]

McGregor and Vu [22] showed that any data stream algorithm for the Maximum k -Coverage Problem (which is a special case of **MSMMatroid** in which f is a coverage function and M is a uniform matroid of rank k) that makes a constant number of passes must use $\Omega(m/k^2)$ memory to achieve $(1 + \varepsilon) \cdot (1 - (1 - 1/k)^k)$ -approximation with probability at least 0.99, where m is the number of sets in the input, and it is assumed that these sets are defined over a ground set of size $n = \Omega(\varepsilon^{-2}k \log m)$. Understanding the implications of this lower bound for **MSMMatroid** requires us to handle two questions.

- The first question is how the lower bound changes as a function of the number of passes. It turns out that when the number of passes is not dropped from the asymptotic expressions because it is considered to be a constant, the lower bound of McGregor and Vu [22] on the space complexity becomes $\Omega(m/(pk^2))$, where p is the number of passes done by the algorithm.
- The second question is about the modifications that have to be done to the lower bound when it is transferred from the Maximum k -Coverage Problem to **MSMMatroid**. Such modifications might be necessary because of input representation issues. However, as it turns out, the proof of the lower bound given by [22] can be applied to **MSMMatroid** directly, yielding the same lower bound (except for the need to replace m with the corresponding value in **MSMMatroid**, namely, $|\mathcal{N}|$). Furthermore, McGregor and Vu [22] had to use a very large ground set so that random sets will behave as one expects with high probability. When the objective function is a general submodular function, rather than a coverage function, it can be chosen to display the above-mentioned behavior of random sets, and therefore, ε can be set to 0.

We summarize the above discussion in the following corollary.

► **Corollary 14** (Corollary of McGregor and Vu [22]). *For any $k \geq 1$, any p -pass data stream algorithm for **MSMMatroid** that achieves an approximation guarantee of $1 - (1 - 1/k)^k \leq 1 - 1/e + 1/k$ with probability at least 0.99 must use $\Omega(|\mathcal{N}|/(pk^2))$ memory, and this is the case even when the matroid M is restricted to be a uniform matroid of rank k .*

7 Approximate Local Maximum for Random Streams

In this section we study **MSMMatroid** in random order streams by building on ideas from the analysis of Liu et al. [21] for optimizing f under a cardinality constraint. We begin with simplifying and reanalyzing the single-pass local search algorithm of Shadravan [29]. By applying this algorithm multiple times (in multiple passes), we are able to prove the following proposition. Proposition 15 implies Theorem 3 by Lemma 8 and Proposition 6.

► **Proposition 15.** *For every constant $\varepsilon > 0$, there is a multi-pass semi-streaming algorithm that given an instance of **MSMMatroid** with a matroid of rank k stores $O(k/\varepsilon)$ elements and makes $O(\varepsilon^{-1} \log \varepsilon^{-1})$ many passes. Assuming the order of the elements in the input stream is chosen uniformly at random in each pass, this algorithm outputs a solution D such that $\mathbb{E}[f(D | \emptyset)] \geq (1 - \varepsilon) \cdot f(D' | \emptyset)$, where D' is the ε -approximate local maximum whose value with respect to f is the smallest.*

In the full version we show that our single-pass algorithm can naturally be extended to p -matchoids. Then, we create a multi-pass algorithm based on this extended single-pass algorithm, which proves Theorem 4.

Intuitively, a local search algorithm should make a swap in its solution whenever this is beneficial. In the adversarial setting, one has to make a swap only when it is beneficial enough to avoid making too many negligible swaps. However, in the random order setting there is a better solution for this problem. Specifically, we (randomly) partition the input stream into *windows* (αk contiguous chunks of the stream with expected size $n/(\alpha k)$ each for some parameter $\alpha > 1$), and then make the best swap within each window. Formally, our random partition is generated according to Algorithm 3.

■ **Algorithm 3** Partitioning of the input stream (α).

-
- 1: Draw $|\mathcal{N}|$ integers uniformly and independently from $1, 2, \dots, \alpha k$.
 - 2: **for** $i = 1$ to αk **do**
 - 3: Let $n_i \leftarrow \#$ of integers equal to i .
 - 4: Let $t_i \leftarrow \sum_{j=1}^{i-1} n_j$.
 - 5: Let $w_i \leftarrow$ elements $t_i + 1$ to $t_i + n_i$ in \mathcal{N} .
 - 6: **return** $\{w_1, w_2, \dots, w_{\alpha k}\}$.
-

Our full single pass algorithm, which uses the partition defined by Algorithm 3, is given as Algorithm 4. The input for the algorithm includes the parameter α and some base L_0 of the matroid \mathcal{M} . Additionally, during the execution of the algorithm, the set L_i represents the current solution, and H is the set of all elements that were added to this solution at some point. When processing window w_i , Algorithm 4 constructs a set C_i of elements that can potentially be swapped into the solution. This set contains all the elements of the window plus some historical elements (the set R_i). The idea of using a set H to store previously valuable elements is inspired from [1, 21]. Reintroducing previously seen elements allows us to give any element not in the solution a chance of being introduced into the solution in the future, which helps us avoid issues that result from the dependence that exists between the current solution and the set of elements in the current window.

■ **Algorithm 4** MATROIDSTREAM(α, L_0).

-
- 1: Partition \mathcal{N} into windows $w_1, w_2, \dots, w_{\alpha k}$.
 - 2: Let $H \leftarrow \emptyset$.
 - 3: **for** $i = 1$ to αk **do**
 - 4: Let R_i be a random subset of H including every $u \in H$ with probability $\frac{1}{\alpha k}$, independently.
 - 5: Let $C_i \leftarrow w_i \cup R_i$
 - 6: Let u^* and u_r^* be elements maximizing $f(L_i - u_r^* + u^*)$ subject to the constraints: $u^* \in C_i, u_r^* \in L_i$ and $L_i - u_r^* + u^* \in \mathcal{I}$.
 - 7: **if** $f(L_i) < f(L_i - u_r^* + u^*)$ **then**
 - 8: Update $H \leftarrow H + u^*$.
 - 9: Let $L_{i+1} \leftarrow L_i - u_r^* + u^*$.
 - 10: **return** $L_{\alpha k}$.
-

Note that the number of elements stored by Algorithm 4 is $O(\alpha k)$, as this number is dominated by the size of the set H . For the same reason Algorithm 4 is a semi-streaming algorithm whenever α is constant.

► **Definition 16.** Let H_i denote the state of the set H maintained by Algorithm 4 immediately after processing window i . We define \mathcal{H}_i to be the set of all pairs (u, j) such that element $u \in H_i$ was added to the solution while window j was processed (i.e., $u \in H_i \cap w_j$). For convenience, sometimes we treat \mathcal{H}_i as a set of elements, and say that $u \in \mathcal{H}_i$ if $u \in H_i$.

One can observe that \mathcal{H}_i encodes all the changes that the algorithm made to its state while processing the first i windows because the element removed from the solution when u is added is deterministic. Additionally, we note that different random permutations of the input and random coins in Line 4 of Algorithm 4 may produce the same history, and we average over all of them in the analysis.

The next lemma is from [21]. It captures the intuition that any element not selected by the algorithm still appears uniformly distributed in future windows, and bounds the probability with which this happens. The proof of this lemma can be found in the full version.

► **Lemma 17.** *Fix a history \mathcal{H}_{i-1} for some $i \in [\alpha k]$. For any element $u \in \mathcal{N} \setminus \mathcal{H}_{i-1}$, and any $i \leq j \leq \alpha k$, we have $\Pr[u \in w_j \mid \mathcal{H}_{i-1}] \geq 1/(\alpha k)$.*

Let B an arbitrary base of \mathcal{M} (one can think of B as an optimal solution because the monotonicity of f guarantees that some optimal solution is a base, but we sometimes need to consider other bases as B). We now define “active” windows, which are windows for which we can show a definite gain in our solution. Specifically, we show below that in any active window the value of the current solution L increases roughly by $\frac{1}{k}(f(B) - 2f(L))$ in expectation, which yields an approximation ratio of $\frac{1}{2}(1 - 1/e^2)$ after αk windows have been processed in one pass because we expect roughly one in every α windows to be active.

► **Definition 18.** *For window w_i , let p_u^i be the probability that $u \in w_i$ conditioned \mathcal{H}_{i-1} . Define the active set A_i of w_i to be the union of R_i and a set obtained by sampling each element $u \in w_i$ with probability $1/(\alpha k p_u^i)$. We call w_i an **active window** if $|B \cap A_i| \geq 1$.*

Note that the construction of active sets in Definition 18 is valid as Lemma 17 guarantees that $1/(\alpha k p_u^i)$ is a valid probability (i.e., it is not more than 1). More importantly, the active set A_i includes every element of \mathcal{N} with probability exactly $1/(\alpha k)$, even conditioned on the history \mathcal{H}_{i-1} ; which implies that, since each element appears in A_i independently, a window is active with probability $(1 - 1/(\alpha k))^k \geq 1 - e^{-1/\alpha} \approx 1/\alpha$ conditioned on any such history. Let \mathcal{A}_i denote the event that window i is active. The following lemma lower bounds the increase in the value of the solution of Algorithm 4 in an active window.

► **Lemma 19.** *For every integer $0 \leq i < \alpha k$,*

$$\begin{aligned} \mathbb{E}[f(L_{i+1}) - f(L_i) \mid \mathcal{H}_i, \mathcal{A}_{i+1}] &\geq \frac{1}{k} \mathbb{E} \left[f(B \mid L_i) + \sum_{u \in B \cap L_i} f(u \mid L_i - u) - f(L_i \mid \emptyset) \mid \mathcal{H}_i \right] \\ &\geq \frac{1}{k} \mathbb{E}[f(B) - 2f(L_i) \mid \mathcal{H}_i] . \end{aligned}$$

Moreover, the above inequality holds even when B is a random base as long as it is deterministic when conditioned on any given \mathcal{H}_i .

Lemma 19 completes the statement of the properties of Algorithm 4 that we need to prove our results. Specifically, the first inequality of the lemma is used to prove that multiple “concatenated” executions of Algorithm 4 output, in expectation, a solution which is almost as good as some ε -approximation local maximum (i.e., Proposition 15), and the rightmost side of the lemma is used to prove Theorem 4. Due to space constraints, the proof of Lemma 19, and the use of this lemma to prove Proposition 15 are deferred to the full version.

References

- 1 Shipra Agrawal, Mohammad Shadravan, and Cliff Stein. Submodular secretary problem with shortlists. In Avrim Blum, editor, *10th Innovations in Theoretical Computer Science Conference (ITCS)*, pages 1:1–1:19, 2019. doi:10.4230/LIPIcs.ITCS.2019.1.

- 2 Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. Streaming submodular maximization: Massive data summarization on the fly. In *Proc. of the 20th ACM Conference on Knowledge Discovery and Data Mining (KDD)*, pages 671–680, 2014.
- 3 Ashwinkumar Badanidiyuru and Jan Vondrák. Fast algorithms for maximizing submodular functions. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, SODA '14, pages 1497–1514, 2014.
- 4 Niv Buchbinder and Moran Feldman. Constrained submodular maximization via a nonsymmetric technique. *Mathematics of Operations Research*, 44(3):988–1005, 2019.
- 5 Gruia Călinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM J. Comput.*, 40(6):1740–1766, 2011. doi:10.1137/080733991.
- 6 Amit Chakrabarti and Sagar Kale. Submodular maximization meets streaming: matchings, matroids, and more. *Mathematical Programming*, 154(1-2):225–247, 2015.
- 7 T-H. Hubert Chan, Zhiyi Huang, Shaofeng H.-C. Jiang, Ning Kang, and Zhihao Gavin Tang. Online submodular maximization with free disposal: Randomization beats 1/4 for partition matroids. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1204–1223, 2017.
- 8 Chandra Chekuri, Shalmoli Gupta, and Kent Quanrud. Streaming algorithms for submodular function maximization. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming*, pages 318–330, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.
- 9 Alina Ene and Huy L. Nguyen. Constrained submodular maximization: Beyond 1/e. In *IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 248–257, 2016.
- 10 Uriel Feige. A threshold of $\ln n$ for approximating set cover. *J. ACM*, 45(4):634–652, 1998.
- 11 M. Feldman, A. Norouzi-Fard, O. Svensson, and R. Zenklusen. The one-way communication complexity of submodular maximization with applications to streaming and robustness. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory on Computing (STOC)*, pages 1363–1374, 2020.
- 12 Moran Feldman, Amin Karbasi, and Ehsan Kazemi. Do less, get more: Streaming submodular maximization with subsampling. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, pages 730–740, 2018. URL: <https://proceedings.neurips.cc/paper/2018/hash/d1f255a373a3cef72e03aa9d980c7eca-Abstract.html>.
- 13 Moran Feldman, Joseph Naor, and Roy Schwartz. A unified continuous greedy algorithm for submodular maximization. In *IEEE 52nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 570–579, 2011.
- 14 Yuval Filmus and Justin Ward. Monotone submodular maximization over a matroid via non-oblivious local search. *SIAM Journal on Computing*, 43(2):514–542, 2014.
- 15 Shayan Oveis Gharan and Jan Vondrák. Submodular maximization by simulated annealing. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1098–1116, 2011.
- 16 Ran Haba, Ehsan Kazemi, Moran Feldman, and Amin Karbasi. Streaming submodular maximization under a k-set system constraint. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, pages 3939–3949, 2020.
- 17 Chien-Chung Huang and Naonori Kakimura. Multi-pass streaming algorithms for monotone submodular function maximization. *CoRR*, abs/1802.06212, 2018.
- 18 Chien-Chung Huang, Theophile Thiery, and Justin Ward. Improved multi-pass streaming algorithms for submodular maximization with matroid constraints, 2021. arXiv:2102.09679.
- 19 Andreas Krause. Submodularity in machine learning. <http://submodularity.org/>.
- 20 Jon Lee, Vahab S. Mirrokni, Viswanath Nagarajan, and Maxim Sviridenko. Non-monotone submodular maximization under matroid and knapsack constraints. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 323–332. ACM, 2009. doi:10.1145/1536414.1536459.

- 21 Paul Liu, Aviad Rubinfeld, Jan Vondrák, and Junyao Zhao. Cardinality constrained submodular maximization for random streams. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. [arXiv:2111.07217](#).
- 22 Andrew McGregor and Hoa T. Vu. Better streaming algorithms for the maximum coverage problem. *Theory of Computing Systems*, 63(7):1595–1619, 2019. [doi:10.1007/s00224-018-9878-x](#).
- 23 Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, and Amin Karbasi. Fast constrained submodular maximization: Personalized data summarization. In *Proceedings of the 33rd International Conference on Machine Learning, ICML*, pages 1358–1367, 2016.
- 24 Baharan Mirzasoleiman, Stefanie Jegelka, and Andreas Krause. Streaming non-monotone submodular maximization: Personalized video summarization on the fly. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*, pages 1379–1386, 2018.
- 25 George L. Nemhauser and Laurence A. Wolsey. Best algorithms for approximating the maximum of a submodular set function. *Math. Oper. Res.*, 3(3):177–188, 1978.
- 26 George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. An analysis of approximations for maximizing submodular set functions – I. *Math. Program.*, 14(1):265–294, 1978.
- 27 Ashkan Norouzi-Fard, Jakub Tarnawski, Slobodan Mitrovic, Amir Zandieh, Aidasadat Mousavifar, and Ola Svensson. Beyond 1/2-approximation for submodular maximization on massive data streams. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pages 3826–3835. PMLR, 2018. URL: <http://proceedings.mlr.press/v80/norouzi-fard18a.html>.
- 28 Alexander Schrijver. *Combinatorial Optimization : Polyhedra and Efficiency*. Springer-Verlag Berlin Heidelberg, 2003.
- 29 Mohammad Shadravan. Submodular Matroid Secretary Problem with Shortlists, January 2020. [arXiv:2001.00894](#).