Streaming Submodular Maximization under Matroid Constraints

Moran Feldman
University of Haifa

Paul Liu
Stanford University

Ashkan Norouzi-Fard
Google Research

Ola Svensson
EPFL

Jan Vondrák
Stanford University

Rico Zenklusen
ETH Zurich

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 $\widetilde{O}(k)$ memory and achieves an approximation guarantee of 0.3178.
- A multi-pass streaming algorithm that uses $\widetilde{O}(k)$ 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 → Submodular optimization and polymatroids; Theory of computation → Streaming models

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

Digital Object Identifier 10.4230/LIPIcs...

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).

1 Introduction

Submodular function optimization is a classic topic in combinatorial optimization (see, e.g., the book [32]). Already in 1978, Nemhauser, Wolsey, and Fisher [30] 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 [13, 29].

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 [7] 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 [18], have been developed to recover the same optimal approximation guarantee.

More recently, applications in data science and machine learning [23], 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 [15], 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 $\mathrm{OPT}/(2k)$. The $(1-1/e-\varepsilon)$ guarantee in $O_\varepsilon(1)$ passes can be obtained using smart implementations of the greedy approach [3, 21, 26, 27, 31].

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 $\widetilde{O}(k)$ additional memory, and achieves an approximation guarantee of* 0.3178.

The last theorem improves upon the previous best approximation guarantee of $1/4 = 0.25$ [8]. 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 [16] to 0.1921).

Our next result is a tight multi-pass guarantee of $1 - 1/e - \varepsilon$, improving upon the previously best guarantee of $1/2 - \varepsilon$ [22].

▶ **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 [15], and any algorithm with a better guarantee than $1 - 1/e$ must make linearly (in the length of the stream) many passes [26] (see Appendix A 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 [22] 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 [22] 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 [8] (and later algorithms based on it such as [10, 16]) 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 though 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 [8] due to Huang, Thiery, and Ward [22]) 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 [34] 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

[1] Technically, we can also get a result for adversarial order streams in this way, but we omit this result

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. [7] 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 [13, 29]. The corresponding problem with a non-monotone objective is not as well understood. A long line of work [12, 17, 24] on this problem culminated in a 0.385-approximation due to Buchbinder and Feldman [4] and an upper bound by Oveis Gharan and Vondrák [19] 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 [8], 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 [9] 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 [9] 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 [10], and achieved a $(1/(4+e)-\varepsilon) \approx 0.1488$-approximation. This was later improved to 0.1715-approximation by Feldman, Karbasi, and Kazemi [16].[2]

Outline of the paper. In Section 2, we introduce notations and definitions used throughout this paper. Afterwords, 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 6 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.

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.

since it is weaker than a known result of [22].

[2] Mirzasoleiman et al. [28] claimed another approximation ratio for the problem (weaker than the one given later by [16]), but some problems were found in their analysis (see [20] for details).

2.1 Problem Statement

Submodular Functions. Given a ground set $\mathcal{N}$, a *set function* $f\colon 2^{\mathcal{N}} \to \mathbb{R}$ 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 $f(u \mid S)$ the marginal contribution of u to S with respect to f, i.e., $f(u \mid S) \coloneqq f(S \cup \{u\}) - f(S)$. Similarly, we denote the marginal contribution of a set $T \subseteq \mathcal{N}$ to S with respect to f by $f(T \mid S) \coloneqq f(S \cup T) - f(S)$.

A set function $f\colon 2^{\mathcal{N}} \to \mathbb{R}$ 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 $f(u \mid S) \geq f(u \mid T)$. 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., $\varnothing \in \mathcal{I}$. 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 $M_1 = (\mathcal{N}_1, \mathcal{I}_1), M_2 = (\mathcal{N}_2, \mathcal{I}_2), \ldots, M_q = (\mathcal{N}_q, \mathcal{I}_q)$ be q matroids, and let $\mathcal{N} = \mathcal{N}_1 \cup \cdots \cup \mathcal{N}_q$ 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 $\mathcal{N}_\ell$ 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 as simply as matchoids (without a parameter p).

Problem. In the `Submodular Maximization subject to a Matroid Constraint` problem (`SMMatroid`), we are given a non-negative[4] submodular function $f\colon 2^{\mathcal{N}} \to \mathbb{R}_{\geq 0}$ 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,

[3] The last property is often referred to as the *exchange axiom* of matroids.
[4] The assumption of non-negativity is necessary to allow multiplicative approximation guarantees.

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 $f\colon 2^{\mathcal{N}} \to \mathbb{R}$ 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 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 [7].

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.

[5] The similar term *streaming* algorithms often refers to algorithms whose space complexity is polylogarithmic 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.

Additional Definitions from Matroid Theory. Matroid theory is extensive, and we refer the reader to [32] 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 $\text{span}_M(S) \coloneqq \{u \in \mathcal{N} \mid \text{rank}(S) = \text{rank}(S + u)\}$ 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 $\widetilde{O}(k)$ 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 Appendix C.

▶ **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 $\widetilde{O}(k)$ additional memory, and achieves an approximation guarantee of* 0.1921.

Throughout this section, we denote by $P_M \coloneqq \{x \in \mathbb{R}^{\mathcal{N}}_{\geq 0} \colon x(S) \leq \text{rank}(S)\ \forall S \subseteq \mathcal{N}\}$ 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, [7]). 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 $\varnothing$ 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 $f(\mathrm{OPT})$ 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 $P_{\mathcal{M}}$.

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 $i(u)$ where updated. However, to obtain an algorithm running in finite (even polynomial) time and linear memory, we do not consider indices below $\max\{b, i(u) - \mathrm{rank}(M) - L\}$ 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 [7], namely that a point in the matroid polytope can be rounded losslessly to an independent set. More formally, given any point $y \in P_{\mathcal{M}}$ in the 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 Appendix B.

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

Algorithm 1 Single-Pass Semi-Streaming Algorithm for MSMMatroid

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

theorems (using the framework) can be found in Sections 5 and 6, 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

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

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

Our objective in Sections 5 and 6 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. Due to space constraints, the formal proof of this proposition is deferred to Appendix D; however, in a nutshell, the proposition is an implication of the properties of "Accelerated Continuous Greedy" described above.

▶ **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 $1-1/e-\varepsilon$.*

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

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

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 $g(\mathrm{OPT}_g)$. We defer the proof of the following observation to Appendix D.

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

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 $\mathbb{E}[g(D \mid \varnothing)] \geq (1-\varepsilon) \cdot g(D' \mid \varnothing)$ 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 $g(\mathrm{OPT}_g \mid \varnothing) \geq 2\varepsilon \cdot f(\mathrm{OPT})$, 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

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

where the second inequality holds by Observation 7.

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

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

where the second inequality holds since D' is an ε-approximate local maximum (by plugging $B = \mathrm{OPT}$ 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

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

Combining the last two inequalities yields

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

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 $(\mathbf{1}_{\mathcal{N}} - y) \wedge \mathbf{1}_{\mathrm{OPT}}$) and the last inequality holds by the motonicity of f and the observation that

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

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 6 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 [8]. 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 [22] (based on ideas of Chekuri et al. [10]). 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 Appendix E.

▶ **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 $(c-1) \cdot f(S_n \mid \varnothing) + \frac{3c-2}{c-1}[f(S_n) - f(S_0)] \geq f(B \mid S_0 \setminus B) - f(S_0 \mid \varnothing) \geq f(B \mid S_0) + \sum_{u \in B \cap S_0} f(u \mid S_0 - u) - f(S_0 \mid \varnothing)$ 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** $f(T_i) - f(T_{i-1}) \leq \varepsilon^2/10 \cdot f(T_1 \mid \varnothing)$ **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 states this formally. Due to space constraints, the proof of this lemma and the next one are deferred to Appendix F.

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

One could image 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 $f(\mathrm{OPT})$, which implies a bound on the number of times this value can be increased significantly. This idea is formalized by the next lemma.

▶ **Lemma 12.** *Algorithm 2 never indicates failure.*

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 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. [25] for optimizing f under a cardinality constraint. We begin with simplifying and reanalyzing the single-pass local search algorithm of Shadravan [34]. By applying this algorithm multiple times (in multiple passes), we are able to prove the following proposition. Proposition 13 implies Theorem 3 by Lemma 8 and Proposition 6.

▶ **Proposition 13.** *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 $\mathbb{E}[f(D \mid \varnothing)] \geq (1-\varepsilon)\cdot f(D' \mid \varnothing)$, where D' is the ε-approximate local maximum whose value with respect to f is the smallest.*

In Appendix I we observe that our single-pass algorithm can naturally be extented 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)

```
Draw |N| integers uniformly and independently from 1, 2, ..., αk.
for i = 1 to αk do
    Let n_i ← # of integers equal to i.
    Let t_i ← Σ_{j=1}^{i-1} n_i.
    Let w_i ← elements t_i + 1 to t_i + n_i in N.
return {w_1, w_2, ..., w_αk}.
```

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, 25]. 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 $w_1, w_2, \ldots, w_{\alpha k}$.
2: Let $H \leftarrow \varnothing$.
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 $u^\star$ and $u_r^\star$ be elements maximizing $f(L_i - u_r^\star + u^\star)$ subject to the constraints: $u^\star \in C_i$, $u_r^\star \in L_i$ and $L_i - u_r^\star + u^\star \in \mathcal{I}$.
7:     **if** $f(L_i) < f(L_i - u_r^\star + u^\star)$ **then**
8:         Update $H \leftarrow H + u^\star$.
9:         Let $L_{i+1} \leftarrow L_i - u_r^\star + u^\star$.
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 14.** *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 [25]. 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 Appendix G.

▶ **Lemma 15.** *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 16.** *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 16 is valid as Lemma 15 guarantees that $1/(\alpha k p_e^i)$ 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 17.** *For every integer $0 \leq i < \alpha k$,*

$$\begin{aligned}\mathbb{E}[f(L_{i+1}) - f(L_i) \mid \mathcal{H}_i, \mathcal{A}_{i+1}] &\geq \tfrac{1}{k}\mathbb{E}\Big[f(B \mid L_i) + \textstyle\sum_{u \in B \cap L_i} f(u \mid L_i - u) - f(L_i \mid \varnothing) \;\Big|\; \mathcal{H}_i\Big] \\ &\geq \tfrac{1}{k}\mathbb{E}[f(B) - 2f(L_i) \mid \mathcal{H}_i] \enspace .\end{aligned}$$

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 17 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 13), and the rightmost side of the lemma is used to prove Theorem 4. Due to space constraints, the proof of Lemma 17, and the use of this lemma to prove Proposition 13 are deferred to Appendix H.

References

1. Shipra Agrawal, Mohammad Shadravan, and Cliff Stein. Submodular secretary problem with shortlists. *CoRR*, abs/1809.05082:1:1–1:19, 2018. URL: http://arxiv.org/abs/1809.05082, arXiv:1809.05082, 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 *Proceedings 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 non-symmetric technique. *Mathematics of Operations Research*, 44(3):988–1005, 2019. doi:10.1287/moor.2018.0955.
5. Niv Buchbinder, Moran Feldman, Joseph Naor, and Roy Schwartz. Submodular maximization with cardinality constraints. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1433–1452. SIAM, 2014. doi:10.1137/1.9781611973402.106.
6. Niv Buchbinder, Moran Feldman, Joseph Naor, and Roy Schwartz. Submodular Maximization with Cardinality Constraints. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1433–1452. SIAM, 2014. doi:10.1137/1.9781611973402.106.
7. Gruia Călinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM Journal on Computing*, 40(6):1740–1766, 2011. doi:10.1137/080733991.
8. Amit Chakrabarti and Sagar Kale. Submodular maximization meets streaming: matchings, matroids, and more. *Mathematical Programming*, 154(1-2):225–247, 2015. doi:10.1007/s10107-015-0900-7.

9 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)*, SODA '17, pages 1204–1223, Philadelphia, PA, USA, 2017. Society for Industrial and Applied Mathematics.

10 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.

11 Chandra Chekuri, T. S. Jayram, and Jan Vondrák. On multiplicative weight updates for concave and submodular function maximization. In Tim Roughgarden, editor, *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 201–210. ACM, 2015. doi:10.1145/2688073.2688086.

12 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. doi:10.1109/FOCS.2016.34.

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

14 Uriel Feige, Vahab S. Mirrokni, and Jan Vondrák. Maximizing non-monotone submodular functions. *SIAM Journal on Computing*, 40(4):1133–1153, 2011. doi:10.1137/090779346.

15 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.

16 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.

17 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. doi:10.1109/FOCS.2011.46.

18 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.

19 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. doi:10.1137/1.9781611973082.83.

20 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. URL: http://proceedings.mlr.press/v119/haba20a.html.

21 Chien-Chung Huang and Naonori Kakimura. Multi-pass streaming algorithms for monotone submodular function maximization. *CoRR*, abs/1802.06212, 2018.

22 Chien-Chung Huang, Theophile Thiery, and Justin Ward. Improved multi-pass streaming algorithms for submodular maximization with matroid constraints, 2021. URL: https://arxiv.org/abs/2102.09679, arXiv:2102.09679, doi:10.4230/LIPIcs.APPROX/RANDOM.2020.62.

23 Andreas Krause. Submodularity in machine learning. http://submodularity.org/.

24 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.

25 Paul Liu, Aviad Rubinstein, Jan Vondrák, and Junyao Zhao. Cardinality constrained submodular maximization for random streams. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. URL: https://arxiv.org/abs/2111.07217.

26 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.

27 Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, and Amin Karbasi. Fast constrained submodular maximization: Personalized data summarization. In *Proceedings of the 33nd International Conference on Machine Learning, ICML*, pages 1358–1367, 2016.

28 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. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17014.

29 George L. Nemhauser and Laurence A. Wolsey. Best algorithms for approximating the maximum of a submodular set function. *Mathematics of Operations Research*, 3(3):177–188, 1978.

30 George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. An analysis of approximations for maximizing submodular set functions—–I. *Mathematical Programming*, 14(1):265–294, 1978.

31 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.

32 Alexander Schrijver. *Combinatorial Optimization : Polyhedra and Efficiency*. Springer-Verlag Berlin Heidelberg, 2003.

33 Mohammad Shadravan. Improved submodular secretary problem with shortlists, 2020. URL: https://arxiv.org/abs/2010.01901, arXiv:2010.01901.

34 Mohammad Shadravan. Submodular Matroid Secretary Problem with Shortlists, January 2020. URL: http://arxiv.org/abs/2001.00894.

35 Ashwinkumar Badanidiyuru Varadaraja. Buyback problem - approximate matroid intersection with cancellation costs. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 379–390, 2011. doi:10.1007/978-3-642-22006-7_32.

36 J. Vondrák. Symmetry and approximability of submodular maximization problems. *SIAM Journal on Computing*, 42(1):265–304, 2013.

A Discussion of a Lower Bound by McGregor and Vu [26]

McGregor and Vu [26] 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 [26] 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 [26] 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 [26] 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 18** (Corollary of McGregor and Vu [26]). *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.*

B Analysis of Algorithm 1

In this section we show that Algorithm 1 from Section 3 implies Theorem 1. Let $\varepsilon \in (0,1]$ in what follows. Additionally, we highlight that we can assume in what follows that there is at least one element $u \in \mathcal{N}$ which gets considered in the for-loop on Line 3 of the algorithm, i.e., it fulfills $\partial_u F(a) > 0$ when appearing in the for-loop. Note that if this does not happen, then we are in a trivial special case where a remains the zero vector and $\partial_u F(a) = 0$ for all $u \in \mathcal{N}$, which corresponds to $f(\mathcal{N}) = f(\varnothing)$. In this case, all sets S_k for $k \in \{0,\dots,m-1\}$ are empty, which implies that s is the zero vector, and one can simply return $R = \varnothing$, which fulfills $f(R) \geq F(s)$, and is even a global maximizer of $f(S)$ over all sets $S \subseteq \mathcal{N}$.

As mentioned in Section 3, we sometimes restrict the considered index range for i in the algorithm to make sure that the algorithm has a finite running time and only uses limited memory. This happens in particular in Line 10 when updating a, where we only consider indices starting from b. However, for the analysis, it is convenient to look at the vector $a_{\text{all}} = \sum_{i=-\infty}^{\infty} \overline{a}_i$ obtained without this lower bound, where $\overline{a}_i$ is the vector a_i when the algorithm terminates. In the definition of a_{all}, we also consider indices $i \in \mathbb{Z}$ together with corresponding vectors $\overline{a}_i$ that have been removed from memory in Line 11. Here, the vector $\overline{a}_i$ is simply the last vector a_i before it got removed from memory in Line 11. Similarly, we let $\overline{A}_i \subseteq \mathcal{N}$ be the set A_i at the end of the algorithm or, in case A_i got removed from memory at some point, $\overline{A}_i$ is the set A_i right before it got removed from memory.

Note that because the coordinates of the vectors a_i never decrease throughout the algorithm, every vector a encountered throughout Algorithm 1 is upper bounded, coordinate-wise, by a_{all}. In the following, we compare both the value of an optimal solution and the value $f(R) \geq F(s)$ of the returned set to $F(a_{\text{all}})$. We start by making sure that the different steps of the algorithm are well defined. For this, we first show that a_{all}, and therefore also any vector a encountered through Algorithm 1, is contained in the box $[0,1]^{\mathcal{N}}$, which implies that the computations of partial derivatives $\partial_u F(a)$ are well defined.

▶ **Observation 19.** *It holds that $a_{\text{all}} \in [0,1]^{\mathcal{N}}$. Consequently, throughout the algorithm, the vector a is also contained in $[0,1]^{\mathcal{N}}$.*

Proof. Consider an element $u \in \mathcal{N}$ and the moment when u was considered in the for-loop at Line 3 of Algorithm 1. Let $i(u)$ be the index computed at Line 4 of the algorithm. Hence, for the vector a at that moment we have $c^{i(u)} \leq \partial_u F(a)$. Thus,

$$a_{\text{all}}(u) \leq \sum_{j=-\infty}^{i(u)} \frac{c^j}{m \cdot \partial_u F(a)} \leq \frac{1}{m} \sum_{j=-\infty}^{0} c^j = \frac{1}{m} \frac{c}{c-1} = \frac{1}{\alpha} \leq 1 \ ,$$

where the second inequality follows from $c^{i(u)} \leq \partial_u F(a)$, and the second equality holds by the definition of c, i.e., $c = \frac{m}{m-\alpha}$. ◀

Moreover, we highlight that the fractional point s rounded at the end of Algorithm 1 at Line 17 is indeed in the matroid polytope P_M. This holds because it is a convex combination of the sets S_k for $k \in \{0, \ldots m-1\}$, each of which is an independent set by construction. Hence, the rounding performed in Line 17 is indeed possible, as discussed.

We now bound the memory used by the algorithm. Note that, for any constant ε (which implies that c is also a constant), the guarantee in the next lemma becomes $O(\text{rank}(M))$, which is the guarantee we need in order to prove Theorem 1. One can also observe that Algorithm 1 stores one non-zero entry in its a_i vectors for every element stored in the sets A_i; and thus, the next lemma also implies that Algorithm 1 is a semi-streaming algorithm using space $\widetilde{O}(\text{rank}(M))$.

▶ **Lemma 20.** *At any point in time, the sum of the cardinalities of all sets A_i that Algorithm 1 has in memory is $O(L \cdot \text{rank}(M)) = O\left(\frac{\log\left(\frac{2c}{\varepsilon(c-1)}\right)}{\log c} \text{rank}(M)\right)$.*

Proof. It suffices to bound the number of elements $\sum_{i=b}^{\infty} |A_i|$ after Line 11. Indeed, we never have more than that many elements in memory plus the number of elements added in a single iteration of the for-loop at Line 3, which is at most $\text{rank}(M) + L = O(L \cdot \text{rank}(M))$. Hence, consider the state of the algorithm at any moment right after the executing of Line 11. We have

$$\sum_{i=b}^{\infty} |A_i| = \sum_{i=b}^{b+L} |A_i| + \sum_{i=b+L+1}^{\infty} |A_i| < (L+1)\,\text{rank}(\mathcal{M}) + \text{rank}(\mathcal{M}) = O(L \cdot \text{rank}(\mathcal{M})) \ ,$$

where the inequality follows from the fact that the first sum has $L+1$ terms, each is the cardinality of an independent set, which is upper bounded by $\text{rank}(M)$; moreover, the second term in the sum is strictly less than $\text{rank}(M)$ by the definition of h in Algorithm 1 (note that $h = b + L$). ◀

We now start to relate the different relevant quantities to $F(a_{\text{all}})$. We start by upper bounding the value of $F(a_{\text{all}})$ as a function of the sets $\overline{A}_i$.

▶ **Lemma 21.**

$$F(a_{\text{all}}) \leq f(\varnothing) + \frac{1}{m}\sum_{i\in\mathbb{Z}} |\overline{A}_i| \cdot c^i \ .$$

Proof. One can think of the vector a_{all} as being constructed iteratively starting with the zero vector $w = 0$ as follows. Whenever Algorithm 1 is at Line 8, we update w by $w \leftarrow w + \frac{c^i}{m\cdot\partial F(a)}\mathbf{1}_u$, where $a \in [0,1]^{\mathcal{N}}$ is the current vector a of the algorithm at that moment in the execution. Note that we have $w \geq a$ because $a = \sum_{j=b}^{\infty} a_j$, for the current value of b and the current vectors a_j, whereas $w = \sum_{j\in\mathbb{Z}} a_j$. Hence, by submodularity of f, we have that the increase of $F(w)$ in this iteration is upper bounded by

$$F\left(q + \frac{c^i}{m\cdot\partial_u F(a)}\mathbf{1}_u\right) - F(q) \leq F\left(a + \frac{c^i}{m\cdot\partial_u F(a)}\mathbf{1}_u\right) - F(a) = \frac{c^i}{m} \ .$$

Hence, the total change in $F(w)$ starting from $F(0) = f(\varnothing)$ to $F(a_{\text{all}})$ is therefore obtained by summing the above left-hand side over all occurrences when algorithm is at Line 8, which leads to

$$F(a_{\text{all}}) - F(0) \leq \frac{1}{m}\sum_{i\in\mathbb{Z}} |\overline{A}_i| \cdot c^i \ ,$$

thus completing the proof. ◀

Let $\bar{b}$ be the value of b at the end of the algorithm. A key difference between the fractional point s, which is constructed during the algorithm, and the point a_{all}, is that sets $\overline{A}_i$ for indices below $\bar{b}$ have an impact on the value of $F(a_{\text{all}})$ (but not on $F(s)$), as reflected in the upper bound on $F(a_{\text{all}})$ in Lemma 21. The following lemma shows that this difference in index range is essentially negligible because the impact of the sets $\overline{A}_i$ in these bounds decreases exponentially fast with decreasing index i.

▶ **Lemma 22.**

$$\frac{1}{m}\sum_{i=-\infty}^{\bar{b}-1} c^i \cdot |\overline{A}_i| \leq \frac{c^{\bar{b}}}{m(c-1)}\operatorname{rank}(M) \leq \frac{\varepsilon}{2c}\cdot\frac{1}{m}\sum_{i=\bar{b}}^{q} c^i \cdot |\overline{A}_i| \ .$$

Proof. The first inequality of the statement follows from $|\overline{A}_i| \leq \operatorname{rank}(M)$ for $i \in \mathbb{Z}$, which holds because $\overline{A}_i \in \mathcal{I}$. The second one follows from

$$\frac{1}{m}\sum_{i=\bar{b}}^{q} c^i \cdot |\overline{A}_j| \geq \frac{1}{m}\sum_{i=\bar{b}+L}^{q} c^i \cdot |\overline{A}_j| \geq \frac{1}{m}\operatorname{rank}(M)\cdot c^{\bar{b}+L} \geq \frac{1}{m}\operatorname{rank}(M)\cdot c^{\bar{b}}\frac{2c}{\varepsilon(c-1)} \ , \tag{2}$$

where the second inequality follows by the fact that $\bar{b}+L$ is the value of h at the end of the algorithm, which fulfills by definition $\sum_{i=h}^{\infty}|\overline{A_j}| \geq \operatorname{rank}(M)$, and the third inequality follows by our definition of L. ◀

Combining Lemma 22 with Lemma 21 now leads to the following lower bound on $F(a_{\text{all}})$, described only in terms of sets $|\overline{A}_i|$ that have not been deleted from memory when the algorithm terminates.

▶ **Corollary 23.**

$$F(a_{\text{all}}) - f(\varnothing) \le \left(1 + \frac{\varepsilon}{2c}\right) \cdot \frac{1}{m} \sum_{i=\bar{b}}^{q} c^i \cdot |\overline{A}_i| \ .$$

Proof. The statement follows from

$$\left(1 + \frac{\varepsilon}{2c}\right) \cdot \frac{1}{m} \sum_{i=\bar{b}}^{q} c^i \cdot |\overline{A}_i| \ge \frac{1}{m} \sum_{i=-\infty}^{q} c^i \cdot |\overline{A}_i| \ge F(a_{\text{all}}) - f(\varnothing) \ ,$$

where the first inequality is due to Lemma 22, and the second one follows from Lemma 21. ◀

Before relating $F(s)$ to $F(a_{\text{all}})$, we need the following structural property on the sets $\overline{A}_i$, which will be exploited to show that the sets S_k, chosen at the end of the algorithm, lead to a point s of high multilinear value.

▶ **Lemma 24.**

$$\overline{A}_i \subseteq \operatorname{span}\left(\overline{A}_{i-1}\right) \quad \forall i \in \{\bar{b}+1, \bar{b}+2, \dots, q\} \ .$$

Proof. Let $u \in \overline{A}_i$, and we show the statement by proving that $u \in \operatorname{span}(\overline{A}_{i-1})$. Consider the state of Algorithm 1 when it performs the for-loop at Line 4 when the outer for-loop is considering the element u (this is the for-loop that adds the element u to sets A_j). Note that we have

$$\bar{b} \ge \max\{b, i(u) - \operatorname{rank}(M) - L\} \ ,$$

due to the following. We clearly have $\bar{b} \ge b$ because the value of b is non-decreasing throughout the algorithm. Moreover, if $i(u) - \operatorname{rank}(M) - L \ge b$, then after the execution of the for-loop at Line 4, we have $A_j \neq \varnothing$ for each $j \in \{i(u) - \operatorname{rank}(M) - L, \dots, i(u)\}$. Hence, right after this execution of the for-loop, the value of b will be increased to at least $i(u) - \operatorname{rank}(M) - L$.

Thus, because u got added to A_i for some $i \in \mathbb{Z}_{>\bar{b}}$, the algorithm will also add u to A_{i-1} if $A_{i-1} + u \in \mathcal{I}$. Hence, after the execution of this for-loop, we have $u \in \operatorname{span}(A_{i-1})$. Finally, because $\overline{A}_{i-1} \supseteq A_{i-1}$, we also have $u \in \operatorname{span}(\overline{A}_{i-1})$. ◀

We are now ready to lower bound the value of $F(s)$ in terms of $F(a_{\text{all}})$.

▶ **Lemma 25.**

$$F(s) \ge f(\varnothing) + \frac{1}{m}(1 - c^{-m}) \sum_{i=\bar{b}}^{q} c^i \cdot |\overline{A}_i| \ge \left(1 - c^{-m} - \frac{\varepsilon}{2c}\right) \cdot F(a_{\text{all}}) \ .$$

Proof. For $k \in \{0, \dots, m-1\}$, we partition S_k into

$$S_k = S_k^{\bar{b}} \cup S_k^{\bar{b}+1} \cup \ldots \cup S_k^q \ ,$$

where S_k^i are the elements that got added to S_k in iteration i of Line 14. Note that because S_k only gets updated in every m-th iteration, we have $S_k^i = \varnothing$ for any $i \not\equiv k \pmod m$. Moreover, we have

$$|S_k^i| \ge |\overline{A}_i| - |\overline{A}_{i+m}| \qquad \forall i \in \mathbb{Z} \text{ with } \bar{b} \le i \le q \text{ and } i \equiv k \pmod m \tag{3}$$

because of the following. We recall that the sets S_k are constructed by adding elements from sets $\overline{A}_j$ from higher indices j to lower ones. Thus, when elements of $\overline{A}_i$ are considered to be added to S_k, the current set S_k only contains elements from sets A_j with $j \geq i+m$ (recall that only elements from every m-th set A_j can be added to S_k). However, by Lemma 24, we have that all those elements are spanned by $\overline{A}_{i+m}$. Hence, when elements of $\overline{A}_i$ are considered to be added to S_k, the set S_k has at most $\operatorname{rank}(\overline{A}_{i+m}) = |\overline{A}_{i+m}|$ many elements since $S_k \in \mathcal{I}$ by construction. Moreover, when elements of the set $\overline{A}_i$ are added to S_k, this is done in a greedy way, which implies that the size of S_k after adding elements from $\overline{A}_i$ will be equal to $\operatorname{rank}(\overline{A}_i) = |\overline{A}_i|$. This implies Equation (3).

The desired relation now follow from

$$
\begin{aligned}
F(s) &\geq f(\varnothing) + \frac{1}{m}\sum_{i=\bar{b}}^{q} c^i \sum_{k=0}^{m-1} |S_k^i| \\
&\geq f(\varnothing) + \frac{1}{m}\sum_{i=\bar{b}}^{q} c^i \cdot \left(|\overline{A}_i| - |\overline{A}_{i+m}|\right) \\
&\geq f(\varnothing) + \frac{1}{m}(1-c^{-m})\sum_{i=\bar{b}}^{q} c^i \cdot |\overline{A}_i| \\
&\geq f(\varnothing) + \frac{1}{m}(1-c^{-m})\left(1+\frac{\varepsilon}{2c}\right)^{-1}(F(a_{\text{all}}) - f(\varnothing)) \\
&\geq \frac{1}{m}(1-c^{-m})\left(1-\frac{\varepsilon}{2c}\right)F(a_{\text{all}}) \ , \\
&\geq \frac{1}{m}\left(1-c^{-m}-\frac{\varepsilon}{2c}\right)F(a_{\text{all}}) \ ,
\end{aligned}
$$

where the first inequality follows from a reasoning analogous to the one used in the proof of Lemma 21,[6] the second one is due to Equation (3), and the fourth one uses Corollary 23 . ◀

Let OPT be an arbitrary (but fixed) optimal solution for our problem. To relate $f(\text{OPT})$ to $F(a_{\text{all}})$, we analyze by how much $f(\text{OPT})$ can be bigger than $F(a_{\text{all}})$. This difference can be bounded through the derivatives $\partial_u F(a_{\text{all}})$, which we analyze first. To this end, for any $u \in \mathcal{N}$, we denote by $\ell(u)$ the largest index $i \in \mathbb{Z}$ such that $u \in \operatorname{span}(\overline{A}_i)$. If no such index exists, we set $\ell(u) = -\infty$.

▶ **Observation 26.**

$$\partial_u F(a_{\text{all}}) \leq c^{\ell(u)+1} \quad \forall u \in \mathcal{N} \ .$$

Proof. Because $u \notin \operatorname{span}(\overline{A}_{\ell(u)+1})$, this implies that u did not get added to the set $A_{\ell(u)+1}$ in Algorithm 1, even though $A_{\ell(u)+1} + u \in \mathcal{I}$, which holds because $A_{\ell(u)+1} + u \subseteq \overline{A}_{\ell(u)+1} + u \in \mathcal{I}$. Hence, when u got considered in Line 3 of Algorithm 1, we had $\partial_u F(a) < c^{\ell(u)+1}$. Finally, by submodularity of f and because $a \leq a_{\text{all}}$ (coordinate-wise), we have $\partial_u F(a_{\text{all}}) \leq \partial_u F(a) \leq c^{\ell(u)+1}$. ◀

We are now ready to bound the difference between $f(\text{OPT})$ and $F(a_{\text{all}})$. Lemma 27 is the first statement in our analysis that exploits monotonicity of f.

[6] More precisely, we can think of s as being constructed iteratively starting from $w = 0$. Whenever the algorithm adds an element $u \in \mathcal{N}$ to some set A_i with $i \in \{\bar{b}, \dots, q\}$, then, if u is also part of the set S_k^i for $k \in \{0, \dots, m-1\}$ with $i \equiv k \pmod{m}$, we update w by setting it to $w + \frac{c^i}{m \cdot \partial_u F(a)} \mathbf{1}_u$. The increase $F(w + \frac{c^i}{m \cdot \partial_u F(a)} \mathbf{1}_u) - F(w)$ is at least as big as $F(a + \frac{c^i}{m \cdot \partial_u F(a)} \mathbf{1}_u) - F(a)$, which is $\frac{c^i}{m}$.

▶ **Lemma 27.**

$$f(\mathrm{OPT}) - F(a_{\mathrm{all}}) \leq \sum_{u \in \mathrm{OPT}} c^{\ell(u)+1} \ .$$

Proof. The result follows from

$$\begin{aligned} f(\mathrm{OPT}) - F(a_{\mathrm{all}}) &\leq F(a_{\mathrm{all}} \vee \mathbf{1}_{\mathrm{OPT}}) - F(a_{\mathrm{all}}) \\ &\leq \nabla F(a_{\mathrm{all}})^T((a_{\mathrm{all}} \vee \mathbf{1}_{\mathrm{OPT}}) - a_{\mathrm{all}}) \\ &\leq \nabla F(a_{\mathrm{all}})^T \mathbf{1}_{\mathrm{OPT}} \\ &= \sum_{u \in \mathrm{OPT}} \partial_u F(a_{\mathrm{all}}) \\ &\leq \sum_{u \in \mathrm{OPT}} c^{\ell(u)+1} \ , \end{aligned}$$

where the first inequality follows from monotonicity of F, the second one because F is concave along non-negative directions, the third one uses again monotonicity of F which implies $\nabla F(a_{\mathrm{all}}) \geq 0$, and the last one follows from Observation 26. ◀

The following lemma allows us to express the bound on the difference between $f(\mathrm{OPT})$ and $F(a_{\mathrm{all}})$ in terms of $F(a_{\mathrm{all}})$, which, combined with the previously derived results, will later allow us to compare $F(s)$ to $f(\mathrm{OPT})$ via the quantity $F(a_{\mathrm{all}})$.

▶ **Lemma 28.**

$$\sum_{u \in \mathrm{OPT}} c^{\ell(u)+1} \leq (c-1)\left(1 + \frac{\varepsilon}{2c}\right) \frac{m}{1 - c^{-m}} F(s) \ .$$

Proof. We start by expanding the left-hand side of the inequality to be shown:

$$\begin{aligned} \sum_{u \in \mathrm{OPT}} c^{\ell(u)+1} &= c \cdot \sum_{u \in \mathrm{OPT}} c^{\ell(u)} \\ &= c \cdot \sum_{i \in \mathbb{Z}} c^i \cdot |\{u \in \mathrm{OPT} \colon \ell(u) = i\}| \\ &= (c-1) \sum_{i \in \mathbb{Z}} |\{u \in \mathrm{OPT} \colon \ell(u) \geq i\}| \\ &= (c-1) \left[\sum_{i=-\infty}^{\bar{b}-1} c^i \cdot |\{u \in \mathrm{OPT} \colon \ell(u) \geq i\}| + \sum_{i=\bar{b}}^{q} c^i \cdot |\{u \in \mathrm{OPT} \colon \ell(u) \geq i\}| \right] \ . \end{aligned} \tag{4}$$

To upper bound the terms in the first sum, we use

$$|\{u \in \mathrm{OPT} \colon \ell(u) \geq i\}| \leq \mathrm{rank}(M) \qquad \forall i \in \mathbb{Z} \ , \tag{5}$$

which holds because $\{u \in \mathrm{OPT} \colon \ell(u) \geq i\} \subseteq \mathrm{OPT}$ and $\mathrm{OPT} \in \mathcal{I}$. Moreover, for the second sum, we use

$$|\{u \in \mathrm{OPT} \colon \ell(u) \geq i\}| \leq |\overline{A}_i| \qquad \forall i \in \mathbb{Z}_{\geq \bar{b}} \ , \tag{6}$$

which holds due to the following. By the definition of $\ell(u)$ and Lemma 24, we have $\{u \in \mathrm{OPT} : \ell(u) \geq i\} \subseteq \mathrm{span}(\overline{A}_i)$. Equation (6) now follows from

$$|\{u \in \mathrm{OPT} \colon \ell(u) \geq i\}| = \mathrm{rank}(\{u \in \mathrm{OPT} \colon \ell(u) \geq i\}) \leq \mathrm{rank}(\mathrm{span}(\overline{A}_i)) = \mathrm{rank}(\overline{A}_i) = |\overline{A}_i| \ ,$$

where the first equality holds because $\{u \in \mathrm{OPT} \colon \ell(u) \geq i\} \subseteq \mathrm{OPT} \in \mathcal{I}$, the inequality holds because $\{u \in \mathrm{OPT} \colon \ell(u) \geq i\} \subseteq \mathrm{span}(\overline{A}_i)$, and the last equation follows from $\overline{A}_i \in \mathcal{I}$.

We now combine the above-proved inequalities to obtain the desired result:

$$\begin{aligned}
\sum_{u \in \mathrm{OPT}} c^{\ell(u)+1} &\leq (c-1)\left[\sum_{i=-\infty}^{\bar{b}-1} c^i \cdot \mathrm{rank}(M) + \sum_{i=\bar{b}}^{q} c^i \cdot |\overline{A}_i|\right] \\
&= (c-1)\left[\mathrm{rank}(M) \cdot \frac{c^{\bar{b}}}{c-1} + \sum_{i=\bar{b}}^{q} c^i \cdot |\overline{A}_i|\right] \\
&\leq (c-1)\left(1 + \frac{\varepsilon}{2c}\right) \sum_{i=\bar{b}}^{q} c^i \cdot |\overline{A}_i| \\
&\leq (c-1)\left(1 + \frac{\varepsilon}{2c}\right) \frac{m}{1-c^{-m}} F(s) \ ,
\end{aligned}$$

where the first inequality follows by Equation (4), Equation (5), and Equation (6), the second inequality follows by Lemma 22, and the last one is a consequence of Lemma 25. ◀

Combining Lemmas 27 and 28 we obtain the following lower bound on $F(s)$ in terms of $f(\mathrm{OPT})$.

▶ **Corollary 29.**

$$F(s) \geq \frac{1 - c^{-m} - \frac{\varepsilon}{2c}}{(c-1)\left(1 + \frac{\varepsilon}{2c}\right) m + 1} \cdot f(\mathrm{OPT}) \ .$$

Proof.

$$\begin{aligned}
f(\mathrm{OPT}) &\leq (c-1)\left(1 + \frac{\varepsilon}{2c}\right) \frac{m}{1-c^{-m}} \cdot F(s) + F(a_{\mathrm{all}}) \\
&\leq (c-1)\left(1 + \frac{\varepsilon}{2c}\right) \frac{m}{1-c^{-m}} \cdot F(s) + \left(1 - c^{-m} - \frac{\varepsilon}{2c}\right)^{-1} F(s) \\
&\leq \frac{(c-1)\left(1 + \frac{\varepsilon}{2c}\right) m + 1}{1 - c^{-m} - \frac{\varepsilon}{2c}} \cdot F(s) \ ,
\end{aligned}$$

where the first inequality follows from Lemmas 27 and 28, and the second one from Lemma 25. ◀

Our result for `MSMMatroid`, i.e., Theorem 1, now follows from Corollary 29 and our choice of parameters α, m, and c, which have been chosen to optimize the ratio. This leads to a lower bound on $F(s)$ in terms of $f(\mathrm{OPT})$, which in turn leads to a lower bound on $f(R)$ because $f(R) \geq F(s)$.

Proof of Theorem 1. We have

$$\begin{aligned}
f(R) &\geq F(s) \qquad (7) \\
&\geq \frac{1 - c^{-m} - \frac{\varepsilon}{2c}}{(c-1)\left(1 + \frac{\varepsilon}{2c}\right) m + 1} f(\mathrm{OPT}) \\
&\geq \frac{1 - e^{-\alpha} - \frac{\varepsilon}{2c}}{(c-1) \cdot \left(1 + \frac{\varepsilon}{2c}\right) m + 1} \cdot f(\mathrm{OPT})
\end{aligned}$$

$$
\begin{aligned}
&= \frac{1 - e^{-\alpha} - \frac{\varepsilon}{2c}}{\alpha\left(c + \frac{\varepsilon}{2}\right) + 1} \cdot f(\mathrm{OPT}) \\
&\geq \left(\frac{1 - e^{-\alpha}}{\alpha + 1} \cdot \frac{1}{c + \frac{\varepsilon}{2}} - \frac{\varepsilon}{2c}\right) \cdot f(\mathrm{OPT}) \\
&\geq \left(\frac{1}{\alpha + 2} \cdot \frac{1}{c + \frac{\varepsilon}{2}} - \frac{\varepsilon}{2c}\right) \cdot f(\mathrm{OPT}) \ ,
\end{aligned}
$$

where the second inequality is due to Corollary 29, the third one follows from $c^{-m} = (1 - \alpha/m)^m \leq e^{-\alpha}$, the equality uses again $c = \frac{m}{m-\alpha}$, and the last inequality uses our choice of value for α (note the inequality would have held as an equality if α was obeying $e^\alpha = \alpha + 2$, and we chose a value that is close).

By our choice of $m = \lceil 3\alpha/\varepsilon \rceil$, we obtain the following bound on c:

$$
c = \frac{m}{m - \alpha} = \frac{1}{1 - \frac{\alpha}{m}} \leq \frac{1}{1 - \frac{\varepsilon}{3}} \leq 1 + \frac{\varepsilon}{2} \ .
$$

Plugging this bound into Equation (7) and using $c \geq 1$, we get

$$
\begin{aligned}
F(R) &\geq \left(\frac{1}{\alpha + 2} \cdot \frac{1}{1 + \varepsilon} - \frac{\varepsilon}{2}\right) \cdot f(\mathrm{OPT}) \\
&\geq \left(\frac{1}{\alpha + 2} \cdot (1 - \varepsilon) - \frac{\varepsilon}{2}\right) \cdot f(\mathrm{OPT}) \\
&\geq \left(\frac{1}{\alpha + 2} - \varepsilon\right) \cdot f(\mathrm{OPT}) \ ,
\end{aligned}
$$

as desired. ◀

C Single-Pass for Non-Monotone Submodular Functions

We now show how our single-pass algorithm for `MSMMatroid`, i.e., Algorithm 1, can be extended to the non-monotone case, i.e., to `SMMatroid`, to obtain Theorem 5. Only minor modifications are needed to the algorithm. The modified algorithm appears as Algorithm 5. There are two changes compared to Algorithm 1, our algorithm for the monotone case. First, the parameter α is chosen differently, and is set to be (approximately) the solution to $e^\alpha = \frac{\alpha^2+2\alpha-1}{\alpha-1}$; more precisely, we set $\alpha = 1.9532$. Second, when updating coordinates of the vectors a_i, we only increase a coordinate, corresponding to some element $u \in \mathcal{N}$, if the total increase of the coordinate u so far does not exceed some target value $p \in (0, 1)$, which is set to $\frac{1}{m(c-1)+1}$.

We recall that most results shown in Section 3 did not rely on monotonicity of f. More precisely, the first result needing monotonicity of f was Lemma 27. Moreover, also Observation 26 does not hold anymore, because it used the fact that we add u to sets A_i as long as the marginal contribution is large enough and $A_i + u$ is independent. However, this is not always the case in Algorithm 5 because we will stop adding u to sets A_i if the condition $\sum_{i=\beta}^{i(u)} a_i(u) \leq p$ on Line 7 of Algorithm 5 fails.

To circumvent this issue, we provide two bounds on the partial derivative $\partial_u F(a_{\mathrm{all}})$, one for elements $u \in \mathcal{N}$ with $a_{\mathrm{all}}(u) \leq p$ and one for elements $u \in \mathcal{N}$ with $a_{\mathrm{all}}(u) \geq p$. In the first case, the condition $\sum_{i=\beta}^{i(u)} \leq p$ was always fulfilled, and we can replicate the same analysis as in the monotone case. In the second case, we exploit that $a_{\mathrm{all}}(u) \geq p$ is large to provide a bound on $\partial_u F(a_{\mathrm{all}})$ that depends on p.

Recall that $\ell(u)$ was defined, in Section 3, as largest index $i \in \mathbb{Z}$ such that $u \in \mathrm{span}(\overline{A}_i)$ (or $-\infty$, if no such index exists).

Algorithm 5 Single-Pass Semi-Streaming Algorithm for `MSMMatroid`

1: Set $\alpha = 1.9532$, $m = \lceil \frac{3\alpha}{\varepsilon} \rceil$, $c = \frac{m}{m-\alpha}$, $p = \frac{1}{m(c-1)+1}$, and $L = \left\lceil \log_c(\frac{2c}{\varepsilon(c-1)}) \right\rceil$.
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 $i(u) = \lfloor \log_c(\partial_u F(a)) \rfloor$. $\triangleright$ Thus, $i(u)$ is largest index $i \in \mathbb{Z}$ with $c^i \leq \partial_u F(a)$.
5:     $\beta \coloneqq \max\{b, i(u) - \text{rank}(M) - L\}$.
6:     **for** $i = \beta$ **to** $i(u)$ **do**
7:         **if** $A_i + u \in \mathcal{I}$ **and** $\sum_{i=\beta}^{i(u)} a_i(u) \leq p$ **then**
8:             $A_i \leftarrow A_i + u$.
9:             $a_i \leftarrow a_i + \frac{c^i}{m \cdot \partial_u F(a)} \mathbf{1}_u$.
10:     Set $b \leftarrow h - L$, where h is largest index $i \in \mathbb{Z}$ satisfying $\sum_{j=i}^{\infty} |A_j| \geq \text{rank}(M)$.
11:     $a \leftarrow \sum_{i=b}^{\infty} a_i$.
12:     Delete from memory all sets A_i and vectors a_i with $i \in \mathbb{Z}_{<b}$.
13: Set $S_k \leftarrow \varnothing$ for $k \in \{0, \ldots, m-1\}$.
14: Let q be largest index $i \in \mathbb{Z}$ with $A_i \neq \varnothing$.
15: **for** $i = q$ **to** b (stepping down by 1 at each iteration) **do**
16:     **while** $\exists u \in A_i \setminus S_{(i \bmod m)}$ with $S_{(i \bmod m)} + u \in \mathcal{I}$ **do**
17:         $S_{(i \bmod m)} \leftarrow S_{(i \bmod m)} + u$.
18: **return** a rounding $R \in \mathcal{I}$ of the fractional solution $s \coloneqq \frac{1}{m} \sum_{k=0}^{m-1} \mathbf{1}_{S_k}$ with $f(R) \geq F(s)$.

▶ **Observation 30.**

$$\partial_u F(a_{\text{all}}) \leq c^{\ell(u)+1} \qquad \forall u \in \mathcal{N} \text{ with } a_{\text{all}}(u) \leq p \ .$$

Proof. Since $u \notin \text{span}(\overline{A}_{\ell(u)+1})$ and $a_{\text{all}}(u) \leq p$, we get that u did not get added to the set $A_{\ell(u)+1}$ in Algorithm 1, even though it fulfilled both $A_{\ell(u)+1} + u \in \mathcal{I}$—because $A_{\ell(u)+1} + u \subseteq \overline{A}_{\ell(u)+1} + u \in \mathcal{I}$—and $\sum_{i=b}^{i(u)} a_i(u) \leq a_{\text{all}}(u) \leq p$ when it got considered. Hence, when u got considered in Line 3 of Algorithm 5, we had $\partial_u F(a) < c^{\ell(u)+1}$. Finally, by submodularity of f and because $a \leq a_{\text{all}}$ (coordinate-wise), we have $\partial_u F(a_{\text{all}}) \leq \partial_u F(a) \leq c^{\ell(u)+1}$. ◀

▶ **Lemma 31.**

$$\partial_u F(a_{\text{all}}) \leq \frac{1}{1-p} \cdot c^{\ell(u)+1} \qquad \forall u \in \mathcal{N} \text{ with } a_{\text{all}}(u) \geq p \ .$$

Proof. Let $u \in \mathcal{N}$ with $a_{\text{all}}(u) \geq p$, and let a be the vector at the beginning of the iteration of the for loop in Line 3 of Algorithm 5 when u got considered. Moreover, let $\beta \coloneqq \max\{b, i(u) - \text{rank}(M) - L\}$ be the β computed and used at that iteration. Because the multilinear extension F is linear in each single coordinate, and in particular the one corresponding to u,

$$F\left(a + \sum_{i=\beta}^{\ell(u)} \frac{c^i}{m \cdot \partial_u F(a)} \mathbf{1}_u\right) - F(a) = \frac{1}{m} \sum_{i=\beta}^{\ell(u)} c^i \ . \tag{8}$$

Note that because the values of $a_i(u)$ are only increased during the iteration of the for loop in Line 3 that considers u, we have

$$a_{\text{all}}(u) = \sum_{i=\beta}^{\ell(u)} \frac{c^i}{m \cdot \partial_u F(a)} \ . \tag{9}$$

Due to the same reason, we have $a(u) = 0$. Hence, $a_{\text{all}} \wedge \mathbf{1}_{\mathcal{N}-u} \geq a$ (coordinate-wise). We thus obtain

$$\begin{aligned}
\frac{1}{m}\sum_{i=\beta}^{\ell(u)} c^i &= F\left(a + a_{\text{all}}(u)\cdot \mathbf{1}_u\right) - F(a) \\
&\geq F\left((a_{\text{all}} \wedge \mathbf{1}_{\mathcal{N}-u}) + a_{\text{all}}(u)\cdot \mathbf{1}_u\right) - F(a_{\text{all}} \wedge \mathbf{1}_{\mathcal{N}-u}) \\
&= a_{\text{all}}(u)\cdot \partial_u F(a_{\text{all}}) \\
&\geq p \cdot \partial_u F(a_{\text{all}}) \enspace ,
\end{aligned} \tag{10}$$

where the first equality is due to Equations (8) and (9), the first inequality follows from submodularity of f and $a_{\text{all}} \wedge \mathbf{1}_{\mathcal{N}-u} \geq a$, the second equality is due to multilinearity of F, and the last inequality holds because we are considering an element $u \in \mathcal{N}$ with $a_{\text{all}}(u) \geq p$. The result now follows due to

$$\partial_u F(a_{\text{all}}) \leq \frac{1}{mp}\sum_{i=\beta}^{\ell(u)} c^i \leq \frac{1}{mp}\sum_{i=-\infty}^{\ell(u)} c^i = \frac{1}{mp(c-1)}\cdot c^{\ell(u)+1} = \frac{1}{1-p}\cdot c^{\ell(u)+1} \enspace ,$$

where the first inequality is due to Equation (10), and the last equality follows from our definition of p, i.e., $p \coloneqq \frac{1}{m(c-1)+1}$. ◀

We now combine Observation 30 and Lemma 31 to obtain a result analogous to Lemma 27 (which we had in the monotone case).

▶ **Lemma 32.**

$$F(a_{\text{all}} \vee \mathbf{1}_{\text{OPT}}) - F(a_{\text{all}}) \leq \sum_{u \in \text{OPT}} c^{\ell(u)+1} \enspace .$$

Proof. Let

$$\text{OPT}_{\text{big}} \coloneqq \{u \in \text{OPT} \colon a_{\text{all}}(u) \geq p\} \enspace .$$

The result now follows from

$$\begin{aligned}
F(a_{\text{all}} \vee \mathbf{1}_{\text{OPT}}) - F(a_{\text{all}}) &\leq \sum_{u \in \text{OPT}} \partial_u F(a_{\text{all}}) \cdot (1 - a_{\text{all}}(u)) \\
&\leq \frac{1}{1-p}\sum_{u \in \text{OPT}_{\text{big}}} c^{\ell(u)+1}\cdot(1 - a_{\text{all}}(u)) + \sum_{u \in \text{OPT}\setminus\text{OPT}_{\text{big}}} c^{\ell(u)+1}\cdot(1 - a_{\text{all}}(u)) \\
&\leq \sum_{u \in \text{OPT}} c^{\ell(u)+1} \enspace ,
\end{aligned}$$

where the first inequality uses concavity of F along non-negative directions, the second one is due to Observation 30 and Lemma 31, and the last one holds since $1 - a_{\text{all}}(u) \leq 1 - p$ in the first sum (because $u \in \text{OPT}_{\text{big}}$) and $1 - a_{\text{all}}(u) \leq 1$ in the second sum. ◀

As in the monotone case, we now would like to relate the difference between $f(\text{OPT})$ and $F(a_{\text{all}})$ to the above-derived bounds on $\partial_u F(a_{\text{all}})$. Lemma 27, which we used in the monotone case, does not hold for non-monotone functions f. To avoid the need for monotonicity, we bound the difference $F(a_{\text{all}} \vee \mathbf{1}_{\text{OPT}}) - F(a_{\text{all}})$ instead. To relate $F(a_{\text{all}} \vee \mathbf{1}_{\text{OPT}})$ to $f(\text{OPT})$, we exploit that a_{all} has small coordinates, through the following known lemma (for completeness, we prove this lemma here, however, we note that it can also be viewed as an immediate corollary of either Lemma 7 of [11] or Lemma 2.2 of [5]).

▶ **Lemma 33.** *Let $f : 2^{\mathcal{N}} \to \mathbb{R}_{\geq 0}$ be a non-negative submodular function with multilinear extension F, and let $p \in [0,1]$, $x \in [0,p]^{\mathcal{N}}$, and $S \subseteq \mathcal{N}$. Then $F(x \vee \mathbf{1}_S) \geq (1-p)f(S)$.*

Proof. We use the fact that the multilinear extension is lower bounded by the Lovász extension $f_L \colon [0,1]^{\mathcal{N}} \to \mathbb{R}_{\geq 0}$, which is given by

$$f_L(y) \coloneqq \int_{t=0}^{1} f\left(\{u \in \mathcal{N} \colon y(u) > t\}\right) dt \qquad \forall y \in [0,1]^{\mathcal{N}} \ .$$

Hence, $F(y) \geq f_L(y)$ for all $y \in [0,1]$ (see, e.g., [36] for a formal proof of this well-known fact). The result now follows from

$$\begin{aligned} F(x \vee \mathbf{1}_S) \geq f_L(x \vee \mathbf{1}_S) \\ &= \int_{t=0}^{1} f(S \cup \{u \in \mathcal{N} \colon x(u) > t\}) dt \\ &\geq \int_{t=p}^{1} f(S \cup \{u \in \mathcal{N} \colon x(u) > t\}) dt \\ &= (1-p) f(S) \ , \end{aligned}$$

where the last equality uses that $x \in [0,p]^{\mathcal{N}}$. ◀

By applying Lemma 33 in our context, we get the following lower bound on $F(a_{\text{all}} \vee \mathbf{1}_{\text{OPT}})$ in terms of $f(\text{OPT})$.

▶ **Corollary 34.**

$$\left(1 - p - \frac{1}{m}\right) \cdot f(\text{OPT}) \leq F(a_{\text{all}} \vee \mathbf{1}_{\text{OPT}})$$

Proof. This is an immediate consequence of Lemma 33 and the fact that $a_{\text{all}} \in [0, p + \frac{1}{m}]$, which holds due to the following. For any element $u \in \mathcal{N}$, Algorithm 5 does not continue to increase coordinates $a_i(u)$ if the sum of the $a_j(u)$ already surpasses p. Moreover, every increase of u happens through an update of one of the a_i vectors by increasing $a_i(u)$ by $\frac{c^i}{m \cdot \partial_u F(a)} \leq \frac{1}{m}$, because $c^i \leq c^{i(u)} \leq \partial_u F(a)$ by choice of $i(u)$. ◀

Finally, by combining the above obtained relations, and using our choices of the parameters α, c, p, and m, we obtain the desired result. Note that the bound on the memory requirement of Algorithm 1 also holds for Algorithm 5, as it is unrelated to monotonicity of f or to the minor differences between the two algorithms.

Proof of Theorem 5. Because $f(R) \geq F(s)$, it suffices to show that $F(s) \geq 0.1921 \cdot f(\text{OPT})$. The value of OPT and $F(s)$ can be related as follows:

$$\begin{aligned} \left(1 - p - \frac{1}{m}\right) \cdot f(\text{OPT}) &\leq F(a_{\text{all}} \vee \mathbf{1}_{\text{OPT}}) \\ &\leq F(a_{\text{all}}) + \sum_{u \in \text{OPT}} c^{\ell(u)+1} \\ &\leq \frac{1}{1 - c^{-m} - \frac{\varepsilon}{2c}} \cdot F(s) + (c-1) \cdot \left(1 + \frac{\varepsilon}{2c}\right) \cdot \frac{m}{1 - c^{-m}} \cdot F(s) \\ &\leq \left(m \cdot (c-1) \cdot \left(1 + \frac{\varepsilon}{2c}\right) + 1\right) \cdot \frac{1}{1 - c^{-m} - \frac{\varepsilon}{2c}} \cdot F(s) \ , \end{aligned}$$

where the first inequality is due to Corollary 34, the second one follows from Lemma 32, and the third one is implied by Lemmas 25 and 28 (we recall that these results did not need monotonicity of f).

Regrouping terms in the above inequality and simplifying, we obtain the following:[7]

$$\begin{aligned} F(s) &\geq \frac{1 - c^{-m} - \frac{\varepsilon}{2c}}{m \cdot (c-1) \cdot \left(1 + \frac{\varepsilon}{2c}\right) + 1} \cdot \left(1 - p - \frac{1}{m}\right) \cdot f(\mathrm{OPT}) \qquad (11) \\ &\geq \frac{1 - e^{-\alpha} - \frac{\varepsilon}{2c}}{\alpha \cdot \left(c + \frac{\varepsilon}{2}\right) + 1} \cdot \left(1 - p - \frac{1}{m}\right) \cdot f(\mathrm{OPT}) \\ &\geq \left(\frac{1 - e^{-\alpha}}{(\alpha+1) \cdot \left(c + \frac{\varepsilon}{2}\right)} - \frac{\varepsilon}{2c}\right) \cdot \left(1 - p - \frac{1}{m}\right) \cdot f(\mathrm{OPT}) \\ &\geq \left(\frac{1 - e^{-\alpha}}{\alpha + 1} \cdot \frac{1}{1+\varepsilon} - \frac{\varepsilon}{2c}\right) \cdot \left(1 - p - \frac{1}{m}\right) \cdot f(\mathrm{OPT}) \\ &\geq \left(\frac{1 - e^{-\alpha}}{\alpha + 1} - \varepsilon\right) \cdot \left(1 - p - \frac{1}{m}\right) \cdot f(\mathrm{OPT}) \\ &\geq \left(\frac{1 - e^{-\alpha}}{\alpha + 1} \cdot (1-p) - \varepsilon\right) \cdot f(\mathrm{OPT}) \\ &= \left(\frac{1 - e^{-\alpha}}{\alpha + 1} \cdot \frac{\alpha c}{\alpha c + 1} - \varepsilon\right) \cdot f(\mathrm{OPT}) \\ &\geq \left(\frac{(1 - e^{-\alpha}) \cdot \alpha}{(\alpha+1)^2} - \varepsilon\right) \cdot f(\mathrm{OPT}) \ , \end{aligned}$$

where the different inequalities hold due to the following. The second inequality uses that $c = \frac{m}{m-\alpha}$, which implies $c^{-m} = (1 - \alpha/m)^m \leq e^{-\alpha}$ and $m(c-1) = \alpha c$. The third inequality holds because $c + \varepsilon/2 \geq 1$ and $\alpha \cdot (c + \varepsilon/2) + 1 \geq 1$. The forth one follows from $c = \frac{m}{m-\alpha} = (1 - \alpha/m)^{-1} \leq (1 - \varepsilon/3)^{-1} \leq 1 + \varepsilon/2$ by using our definitions of c and m. The fifth inequality uses that $(1+\varepsilon)^{-1} \geq 1 - \varepsilon$ and $\frac{1-e^{-\alpha}}{\alpha+1} \leq \frac{1}{2}$. The sixth inequality holds because $\frac{1-e^{-\alpha}}{\alpha+1} \frac{1}{m} \leq \frac{1}{3m} \leq \frac{\varepsilon}{9\alpha}$ and $p \cdot \varepsilon = \frac{\varepsilon}{\alpha c+1} \geq \frac{\varepsilon}{9\alpha}$. The requality uses that $p = \frac{1}{\alpha c+1}$. Finally, the last inequality follows from $c \geq 1$.

The claimed approximation factor of 1.921 is obtained by plugging in our value of $\alpha = 1.9532$ (for a small enough $\varepsilon > 0$). ◀

D Omitted Proofs of Section 4

This section includes the proofs that are omitted from Section 4.

D.1 Proof of Proposition 6

▶ **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 $1 - 1/e - \varepsilon$.*

[7] The first steps of the derivation are analogous to the ones performed in the proof of Theorem 1. The only difference is the additional term of $(1 - p - 1/m)$.

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 $\widetilde{O}(k/\varepsilon)$. As this term is nearly linear in k for any constant ε, we get that `DSCG` has a low enough space 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. ◀

D.2 Proof of Observation 7

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

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

$$g(D) \geq g(D \mid \varnothing) \geq g(OPT_g \mid D) + \sum_{u \in \mathrm{OPT}_g \cap D} g(u \mid D - u) - \varepsilon \cdot g(\mathrm{OPT}_g \mid \varnothing)$$
$$\geq g(\mathrm{OPT}_g \mid D) - \varepsilon \cdot g(OPT_g \mid \varnothing) \geq (1 - \varepsilon) \cdot g(\mathrm{OPT}_g) - g(D) \ ,$$

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 $B = \mathrm{OPT}_g$), 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. ◀

E Alternative Analysis for a Known Single Pass Algorithm

The first data stream algorithm for `MSMMatroid` was described by Chakrabarti and Kale [8]. In this section we consider a variant of their algorithm. This variant is a special case of an algorithm that was described by Huang, Thiery and Ward [22] (based on ideas of Chekuri et al. [10]), and it appears as Algorithm 6. Algorithm 6 gets a parameter $c > 1$ and a base S_0 of M that it starts from, and intuitively, it inserts every arriving element into its solution (at the expense of an appropriate existing element) whenever such a swap is beneficial enough in some sense. In the pseudocode of Algorithm 6, we denote by $u_1, u_2, \dots, u_n$ the elements of $\mathcal{N} \setminus S_0$ in the order of their arrival. Similarly, we denote by S_i the solution of the algorithm immediately after it processes element u_i for every integer $1 \leq i \leq n$. Finally, we denote the elements of the base S_0 by $u_{1-|S_0|}, u_{2-|S_0|}, \dots, u_0$ in an arbitrary order. This notation allows us to define, for every integer $1 - |S_0| \leq i \leq n$ and set $T \subseteq \mathcal{N}$,

$$f(u_i : T) = f(u_i \mid \{u_j \in T \mid 1 - |S_0| \leq j < i\}) \enspace .$$

In other words, $f(u_i : T)$ is the marginal contribution of u_i with respect to the elements of T that appear in the input stream of the algorithm before u_i.

Algorithm 6 Single Local Search Pass (S_0, c)

1: **for** every element $u_i \in \mathcal{N} \setminus S_0$ that arrives **do**
2: Let C_i be the single cycle in $S_{i-1} + u_i$.
3: Let u'_i be the element in $C_i - u_i$ minimizing $f(u'_i : S_{i-1})$.
4: ▷ Note that u'_i is equal to u_j for some $j < i$.
5: **if** $f(u_i \mid S_{i-1}) \geq c \cdot f(u'_i : S_{i-1})$ **then**
6: Set $S_i \leftarrow S_{i-1} - u'_i + u_i$.
7: **else**
8: Set $S_i \leftarrow S_{i-1}$.
9: **return** S_n.

One can verify that the solution of Algorithm 6 remains a base of M throughout the execution of the algorithm. Furthermore, it is known that Algorithm 6 achieves 4-approximation for `MSMMatroid` when $c = 2$. However, we need to prove a slightly different property of it. Specifically, we show below that when S_0 is not an approximation local maximum, the value of the final solution S_n of Algorithm 6 is much larger than the value of the initial solution S_0. In Section 5 we show how this property of Algorithm 6 can be used to find an ε-approximate local maximum in $O(\varepsilon^{-2})$ passes.

Let B be an arbitrary base of M (intuitively, one can think of B as the optimal solution, although this will not always be the case). We begin the analysis of Algorithm 6 by showing a lower bound on the sum of the marginal contributions of the elements of $B \setminus S_0$ with respect to the solutions held by Algorithm 6 when these elements arrive. Let A be the set of all elements that belong to the solution of Algorithm 6 at some point (formally, $A = \bigcup_{i=0}^{n} S_i$).

▶ **Lemma 35.** $\sum_{u_i \in B \setminus S_0} f(u_i \mid S_{i-1}) \geq f(S_0 \cup B) + \frac{1}{c-1} \cdot f(S_0) - \frac{c}{c-1} \cdot f(S_n)$.

Proof. By the submodularity of f,

$$\begin{aligned}\sum_{u_i \in B \setminus S_0} f(u_i \mid S_{i-1}) &\geq \sum_{u_i \in B \setminus S_0} f(u_i \mid A) \geq f(B \mid A) = f(B \cup (A \setminus S_0) \mid S_0) - f(A \setminus S_0 \mid S_0)\\ &\geq f(B \mid S_0) - \sum_{u_i \in A \setminus S_0} f(u_i \mid S_0 \cup \{u_j \in A \mid 1 \leq j < i\})\\ &\geq f(B \mid S_0) - \sum_{u_i \in A \setminus S_0} f(u_i \mid S_{i-1}) \enspace ,\end{aligned}$$

where the third inequality holds by the monotonicity of f.

Let us now upper bound the second term in the rightmost side. Since all the elements of $A \setminus S_0$ were accepted by Algorithm 6 into its solution,

$$\begin{aligned}
\sum_{u_i \in A \setminus S_0} f(u_i \mid S_{i-1}) &\leq \tfrac{c}{c-1} \cdot \sum_{u_i \in A \setminus S_0} [f(u_i \mid S_{i-1}) - f(u'_i : S_{i-1})] \\
&\leq \tfrac{c}{c-1} \cdot \sum_{u_i \in A \setminus S_0} [f(u_i \mid S_{i-1}) - f(u'_i \mid S_{i-1} + u_i - u'_i)] \\
&= \tfrac{c}{c-1} \cdot \sum_{u_i \in A \setminus S_0} [f(S_i) - f(S_{i-1})] = \tfrac{c}{c-1} \cdot [f(S_n) - f(S_0)] \enspace ,
\end{aligned}$$

where the second inequality holds by the submodularity of f, and the last equality holds since $S_i = S_{i-1}$ for every integer $1 \leq i \leq n$ for which $u_i \notin A$. The lemma now follows by combining the two above inequalities. ◀

To complement the last lemma, we need to upper bound the marginal contributions of the elements u'_i corresponding to the elements $u_i \in A$ with respect to the solutions of Algorithm 6 when the last elements arrive. We prove such an upper bound in Corollary 40 below. However, proving this upper bound requires us to present a few additional definitions as well as properties of the objects defined. We begin by constructing an auxilary directed graph G whose vertices are the elements of $\mathcal{N}$. Furthermore, for every element $u_i \in \mathcal{N} \setminus S_0$, we create edges for the graph G in the following way. Note that there is a single element $c_i \in C_i$ that does not belong to S_i. The graph G includes edges from c_i to every other element of C_i. Let us now prove some properties of the graph G.

▶ **Observation 36.** *For every element $u \in \mathcal{N}$, let us define*

$$\mathrm{Val}(u) = \begin{cases} f(u : S_n) & \text{if } u \in S_n \enspace , \\ f(u : S_i) & \begin{array}{l}\text{if } u \in A \setminus S_n, \text{ and } u \text{ was removed from} \\ \text{the solution of Algorithm 6 when } u_i \text{ arrived} \end{array} \enspace , \\ f(u \mid S_i) & \text{if } u \notin A \text{ and } u = u_i \enspace . \end{cases}$$

Then, for every edge uv of G such that $u \in A$, $\mathrm{Val}(u) \leq \mathrm{Val}(v)$.

Proof. Since there is an edge from u to v, u must have been removed from A when some element u_i arrived, and v was another element of the cycle C_i. If $v \neq u_i$, then the fact that u was removed (rather than v) implies

$$\mathrm{Val}(u) = f(u : S_i) \leq f(v : S_i) \leq \mathrm{Val}(v) \enspace ,$$

where the second inequality holds since $\mathrm{Val}(v)$ is equal to $f(v : S_j)$ for some $j \geq i$. Otherwise, if $v = u_i$, then the fact that u was removed following the arrival of v implies

$$\mathrm{Val}(u) = f(u : S_i) \leq \frac{f(v \mid S_i)}{c} \leq \frac{\mathrm{Val}(v)}{c} \leq \mathrm{Val}(v) \enspace ,$$

where the last inequality holds since the monotonicity of f guarantees that $\mathrm{Val}(v)$ is non-negative. ◀

▶ **Corollary 37.** *If u and v are two elements of A such that v is reachable from u in G, then $\mathrm{Val}(u) \leq \mathrm{Val}(v)$.*

Proof. The corollary follows from Observation 36 because the construction of G guarantees that the vertices of $\mathcal{N} \setminus A$ are all sources of G (i.e., vertices that do not have any edge entering them). ◀

▶ **Observation 38.** *G is acyclic; and every element $u \in \mathcal{N}$ that is not a sink of G is spanned by the elements of $\delta^+(u)$, where $\delta^+(u) = \{v \mid uv \text{ is an edge of } G\}$.*

Proof. Every edge e of G was created due to some cycle C_i. Furthermore, the edge e goes from a vertex that does not appear in S_i or any solution that Algorithm 6 has at a later time point to a vertex that does belong to S_i. Therefore, if we sort the vertices of G by the largest index i for which they belong to S_i (a vertex that does not belong to S_i for any i is placed before all the vertices that do belong to S_i for some i), then we obtain a topological order of G, which implies that G is acyclic.

To prove the second part of the observation, we note that whenever the construction of G includes edges leaving a node u, this implies that these edges go to all the vertices of $C - u$ for some cycle C that includes u. Therefore, $\delta^+(u) \supseteq C - u$ spans u. ◀

To use the last observation, we need the following known lemma (a similar lemma appeared earlier in [35] in an implicit form, and was made explicit in [10]).

▶ **Lemma 39** (Lemma 13 of [16])**.** *Consider an arbitrary directed acyclic graph $G = (V, E)$ whose vertices are elements of some matroid M'. If every non-sink vertex u of G is spanned by $\delta^+(u)$ in M', then for every set S of vertices of G which is independent in M' there must exist an injective function ψ_S such that, for every vertex $u \in S$, $\psi_S(u)$ is a sink of G which is reachable from u.*

▶ **Corollary 40.** *$\sum_{u_i \in B \setminus A} f(u'_i : S_{i-1}) + c \cdot \sum_{u_i \in B \cap (A \setminus S_0)} f(u'_i : S_{i-1}) \leq f(S_n \mid \varnothing) - f(S_0 \mid S_0 \setminus B)$.*

Proof. Let ψ_B be the function whose existence is guaranteed by Lemma 39 (recall that B is a base of M, and therefore, is independent in M). Consider now an element $u_i \in B \setminus A$, and let P_i be the path in G from u_i to $\psi_B(u_i)$ whose existence is guaranteed by Lemma 39. If we denote by u''_i the element that appears in this path immediately after u_i (there must be such an element because $u_i \notin A \supseteq S_n$, and therefore, is not a sink of G), then $\text{Val}(u''_i) \leq \text{Val}(\psi_B(u_i))$ according to Corollary 37. Additionally, since u_i was rejected by Algorithm 6 immediately upon arrival, both u'_i and u''_i are elements of $C_i - u_i$, and thus, due to the way in which Algorithm 6 selects u'_i,

$$f(u'_i : S_{i-1}) \leq f(u''_i : S_{i-1}) \leq \text{Val}(u''_i) \leq \text{Val}(\psi_B(u_i)) = f(\psi_B(u_i) : S_n) \ ,$$

where the last equality holds since $\psi_B(u_i)$ is a sink of G, and therefore, belongs to S_n.

Consider now an element $u_i \in B \cap (A \setminus S_0)$. Since $\psi_B(u_i)$ is reachable in G from u_i, $\text{Val}(u_i) \leq \text{Val}(\psi_B(u_i))$. Therefore, the fact that u_i was added upon arrival to the solution of Algorithm 6 implies

$$f(u'_i : S_{i-1}) \leq \frac{f(u_i \mid S_{i-1})}{c} \leq \frac{\text{Val}(u_i)}{c} \leq \frac{\text{Val}(\psi_B(u_i))}{c} = \frac{f(\psi_B(u_i) : S_n)}{c} \ .$$

Combining both the above inequalities, we get

$$\begin{aligned}
\sum_{u_i \in B \setminus A} f(u'_i : S_{i-1}) + c \cdot \sum_{u_i \in B \cap (A \setminus S_0)} f(u'_i : S_{i-1}) \leq \sum_{u_i \in B \setminus S_0} f(\psi_B(u_i) : S_n) \\
= \sum_{u_i \in S_n} f(\psi_B(u_i) : S_n) - \sum_{u_i \in B \cap S_0} f(\psi_B(u_i) : S_n) \leq f(S_n \mid \varnothing) - \sum_{u_i \in B \cap S_0} \text{Val}(u_i) \\
\leq f(S_n \mid \varnothing) - f(S_0 \cap B \mid S_0 \setminus B) = f(S_n \mid \varnothing) - f(S_0 \mid S_0 \setminus B) \ ,
\end{aligned}$$

where the first equality holds because ψ_B is a bijection from B to S_n by Lemma 39, and the third inequality holds since $\sum_{u_i \in B \cap S_0} \text{Val}(u_i) \geq \sum_{u_i \in B \cap S_0} f(u_i : S_0) \geq \sum_{u_i \in B \cap S_0} f(u_i \mid (S_0 \setminus B) \cup (S_0 \cap \{u_{1-k}, u_{2-k}, \ldots, u_{i-1}\})) = f(S_0 \cap B \mid S_0 \setminus B)$. ◀

We are now ready to prove our main result regarding Algorithm 6, namely that it has all the properties guaranteed by Proposition 10. In the beginning of the section, we claimed that this result shows that the difference $f(S_n) - f(S_0)$ is large whenever S_0 is not an approximate local maximum. To see why this is the case, note that the rightmost side in the next proposition is guaranteed to be large (for some base B) when S_0 is not an approximate local maximum.

▶ **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 $(c-1) \cdot f(S_n \mid \varnothing) + \frac{3c-2}{c-1}[f(S_n) - f(S_0)] \geq f(B \mid S_0 \setminus B) - f(S_0 \mid \varnothing) \geq f(B \mid S_0) + \sum_{u \in B \cap S_0} f(u \mid S_0 - u) - f(S_0 \mid \varnothing)$ for every base B of M. Furthermore, this algorithm stores $O(k)$ elements at any point during its execution.*

Proof. The first part of the proposition holds because implementing Algorithm 6 requires us to maintain only two bases of the matroid M, the input base S_0 and the current solution of the algorithm. The rest of this proof is devoted to proving the second part of the proposition.

Observe that whenever Algorithm 6 changes its solution while processing element u_i, the value of this solution changes by

$$\begin{aligned} f(u_i \mid S_{i-1}) - f(u'_i \mid S_{i-1} + u_i - u'_i) &\geq c \cdot f(u'_i : S_{i-1}) - f(u'_i \mid S_{i-1} + u_i - u'_i) \\ &\geq (c-1) \cdot f(u'_i \mid S_{i-1} + u_i - u'_i) \geq 0 \ , \end{aligned}$$

where the second inequality holds by the submodularity of f and the last inequality follows from the monotonicity of f. This implies that $f(S_i)$ is a non-decreasing function of i, and therefore,

$$\begin{aligned} f(S_n) - f(S_0) &\geq \sum_{u_i \in B \cap (A \setminus S_0)} [f(u_i \mid S_{i-1}) - f(u'_i \mid S_{i-1} + u_i - u'_i)] \\ &\geq \sum_{u_i \in B \cap (A \setminus S_0)} [f(u_i \mid S_{i-1}) - f(u'_i \mid S_{i-1} - u'_i)] \geq \sum_{u_i \in B \cap (A \setminus S_0)} [f(u_i \mid S_{i-1}) - f(u'_i : S_{i-1})] \\ &\geq \sum_{u_i \in B \setminus S_0} f(u_i \mid S_{i-1}) - \sum_{u_i \in B \cap (A \setminus S_0)} f(u'_i : S_{i-1}) - c \cdot \sum_{u_i \in B \setminus A} f(u'_i : S_{i-1}) \\ &\geq \sum_{u_i \in B \setminus S_0} f(u_i \mid S_{i-1}) - c^2 \cdot \sum_{u_i \in B \cap (A \setminus S_0)} f(u'_i : S_{i-1}) - c \cdot \sum_{u_i \in B \setminus A} f(u'_i : S_{i-1}) \ , \end{aligned}$$

where the second and third inequalities hold by the submodularity of f, the penultimate inequality holds since the elements of $B \setminus A$ where not added by Algorithm 6 to its solution, and the last inequality holds by the monotonicity of f.

Using Lemma 35 and Corollary 40, the previous inequality implies

$$\begin{aligned} &f(S_n) - f(S_0) \\ &\quad \geq \{f(S_0 \cup B) + \tfrac{1}{c-1} \cdot f(S_0) - \tfrac{c}{c-1} \cdot f(S_n)\} - c \cdot \{f(S_n \mid \varnothing) - f(S_0 \mid S_0 \setminus B)\} \\ &\quad = f(S_0 \cup B) + c \cdot f(\varnothing) + \tfrac{1}{c-1} \cdot f(S_0) - \tfrac{c^2}{c-1} \cdot f(S_n) + c \cdot f(S_0 \mid S_0 \setminus B) \\ &\quad \geq f(S_0 \cup B) + c \cdot f(\varnothing) + \tfrac{1}{c-1} \cdot f(S_0) - \tfrac{c^2}{c-1} \cdot f(S_n) + f(S_0 \mid S_0 \setminus B) \ , \end{aligned}$$

where the second inequality follows from monotonicity of f and the fact that $c > 1$. The first inequality of the proposition now follows by rearranging the last inequality (this can be verified by checking term by term).

To see that the second inequality of the proposition holds as well, we note that, by the submodulary of f,

$$f(B \mid S_0 \setminus B) = f(B \mid S_0) + f(B \cap S_0 \mid S_0 \setminus B) \geq f(B \mid S_0) + \sum_{u \in B \cap S_0} f(u \mid S_0 - u) \ . \quad \blacktriangleleft$$

F Omitted Proofs of Section 5

This section includes the proofs that are omitted from Section 5 (except for the proof of Proposition 10, which appears in Appendix E).

F.1 Proof of Lemma 11

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

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

$$f(T_i) - f(T_{i-1}) \leq (\varepsilon^2/10) \cdot f(\mathrm{OPT} \mid \varnothing) \ .$$

Plugging this inequality and the fact that $f(T_i \mid \varnothing) \leq f(\mathrm{OPT} \mid \varnothing)$ (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

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

F.2 Proof of Lemma 12

The proof of Lemma 12 uses the following observation.

▶ **Observation 41.** $f(T_1 \mid \varnothing) \geq \frac{1}{5} f(\mathrm{OPT} \mid \varnothing)$.

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

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

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

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

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

Using the last observation, we can now prove Lemma 12.

▶ **Lemma 12.** *Algorithm 2 never indicates failure.*

Proof. If $f(\mathrm{OPT} \mid \varnothing) = 0$, then the value of every base of M according to f is $f(\varnothing)$, 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 $f(\mathrm{OPT} \mid \varnothing) > 0$. Furthermore, assume towards a contradiction that Algorithm 2 indicates failure. By Observation 41, this assumption implies that the value of the solution maintained by Algorithm 2 increases by at least $(\varepsilon^2/10) \cdot f(T_1 \mid \varnothing) \geq \frac{\varepsilon^2}{50} f(\mathrm{OPT} \mid \varnothing)$ 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

$$f(T_1) + (1 + \lceil 40\varepsilon^{-2} \rceil) \cdot \tfrac{\varepsilon^2}{50} f(\mathrm{OPT} \mid \varnothing) > f(\varnothing) + \tfrac{1}{5} f(\mathrm{OPT} \mid \varnothing) + \tfrac{4}{5} f(\mathrm{OPT} \mid \varnothing) = f(\mathrm{OPT}) \enspace,$$

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

G Proof of Lemma 15

In this section we prove Lemma 15. We begin with the following helper lemma.

▶ **Lemma 42.** *Suppose the element of $\mathcal{N}$ appear in the stream in a uniformly random order, and we partition $\mathcal{N}$ by Algorithm 3 into αk windows, then this is equivalent to assigning each $u \in \mathcal{N}$ to one of αk different buckets uniformly and independently at random.*

Proof. The way we define the window sizes $n_1, n_2, \ldots,$ is equivalent to placing each element independently into a random bucket, and then letting n_i be the number of elements that ended up in bucket i. Hence, the distribution of the window sizes is correct. Furthermore, conditioned on the window sizes, each window is simply assigned the elements in some positions of the random stream, and therefore, the set of elements it gets is a uniformly random subset of $\mathcal{N}$ of the right size which is independent of the partitioning of the remaining elements between the other windows. ◀

Let us denote now by R the random coins used in Algorithm 4 of Algorithm 4. Below, we prove Lemma 15 conditioned in a fixed choice of R, which implies that the lemma holds also unconditionally due to the law of total probability. Let $J_u(\mathcal{H}_{i-1}, R)$ be the set of indices j where there exists some partition P that implies the history $\mathcal{H}_{i-1}$ given R[8] and has $P(u) = j$.

▶ **Lemma 43.** *Conditioned on history $\mathcal{H}_{i-1}$ and random coins R, the probability of an element $u \in \mathcal{N} \setminus \mathcal{H}_{i-1}$ to end up in every window corresponding to the indices of $J_u(\mathcal{H}_{i-1})$ is equal. Furthermore, $J_u(\mathcal{H}_{i-1}, R)$ includes every integer $i \leq j \leq \alpha k$*

Proof. Choose any $j, j' \in J_u(\mathcal{H}_{i-1})$. We would like to show that for each partition P that implies the history $\mathcal{H}_{i-1}$ given R and has $P(u) = j$ we can create another partition $\tilde{P}$ that implies $\mathcal{H}_{i-1}$ given R by setting $\tilde{P}(u) = j'$ and keeping all other values of $\tilde{P}$ as in P. Since $\tilde{P}$ is equal to P everywhere except on u, this maps each such partition P to a unique partition $\tilde{P}$, establishing that the number of partitions that imply $\mathcal{H}_{i-1}$ given R and map u to w_j is

[8] Observe that once R and P are fixed, Algorithm 4 becomes deterministic, and therefore, P and R determine $\mathcal{H}_{i-1}$.

not larger than the number of such partitions mapping u to $w_{j'}$. Since this is true for every $j, j \in J_u(\mathcal{H}_{i-1}, R)$, and all the partitions have equal probability by Lemma 42, the first part of the lemma follows once we show the above.

Since $u \notin \mathcal{H}_{i-1}$, for an index j to be in $J_u(\mathcal{H}_{i-1}, R)$, one of two things must happen. The first option is that $j \geq i$, in which case trivially Algorithm 4 could not add j to H while processing the first $i-1$ windows (note that the existence of this option already implies the second part of the lemma). The second option is that $j < i$, but u was not selected by Algorithm 4 when it arrived because either u was never the maximum element found in Line 6, or if it was, its marginal value was not sufficient to replace the current solution. In all these cases, removing or adding u to window w_j does not change the history $\mathcal{H}_{i-1}$. Thus, given that P implies the history $\mathcal{H}_{i-1}$ given R, changing $P(u)$ from one index of $J_u(\mathcal{H}_{i-1}, R)$ to another does not change this history. ◀

We are now ready to prove Lemma 15, which we repeat here for convenience.

▶ **Lemma 15.** *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 $Pr[u \in w_j \mid \mathcal{H}_{i-1}] \geq 1/(\alpha k)$.*

Proof. Since u must appear in some window, and it can appear only in windows whose indices appear in $J_u(\mathcal{H}_{i-1}, R)$, Lemma 43 implies that conditioned on R we have

$$1 = \sum_{j' \in J_u(\mathcal{H}_{i-1}, R)} \Pr[u \in w_{j'} \mid \mathcal{H}_{i-1}] = |J_u(\mathcal{H}_{i-1}, R)| \cdot \Pr[u \in w_j \mid \mathcal{H}_{i-1}] \leq \alpha k \cdot \Pr[u \in w_j \mid \mathcal{H}_{i-1}] \enspace .$$

As mentioned above, the conditioning on R can be dropped by the law of totol probability, which implies the lemma. ◀

H Continuing the Proof of Proposition 13

We note that that this section highly depends on Section 4 (starting after the proof of Theorem 46), and should not be read before that section. Section 6 concluded by presenting Lemma 17, which we would like to prove. However, we first need to introduce the following known lemmata.

▶ **Lemma 44** (Lemma 2.2 of [14]). *Let $g\colon 2^{\mathcal{N}} \to \mathbb{R}$ be a submodular function. Further, let R be a random subset of $T \subseteq \mathcal{N}$ in which every element occurs with probability p (not necessarily independently). Then, $\mathbb{E}[g(R)] \geq p \cdot g(T) + (1-p) \cdot g(\varnothing)$.*

▶ **Lemma 45** (Follows, for example, from Corollary 39.12a of [32]). *If S and T are two bases of a matroid $\mathcal{M} = (\mathcal{N}, \mathcal{I})$, then there exists a bijection $h\colon T \to S$ such that for every $u \in T$, $T - h(u) + u \in \mathcal{I}$. Furthermore, for every element $u \in S \cap T$, $h(u) = u$.*

We are now ready to prove Lemma 17.

▶ **Lemma 17.** *For every integer $0 \leq i < \alpha k$,*

$$\begin{aligned}\mathbb{E}[f(L_{i+1}) - f(L_i) \mid \mathcal{H}_i, \mathcal{A}_{i+1}] &\geq \tfrac{1}{k}\mathbb{E}\Big[f(B \mid L_i) + \textstyle\sum_{u \in B \cap L_i} f(u \mid L_i - u) - f(L_i \mid \varnothing) \;\Big|\; \mathcal{H}_i\Big] \\ &\geq \tfrac{1}{k}\mathbb{E}[f(B) - 2f(L_i) \mid \mathcal{H}_i] \enspace .\end{aligned}$$

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

Proof. Let us apply Lemma 45 with $S = L_i$ and $T = B$ to get a bijection $h\colon B \to L_i$ with the properties specified in the lemma. Let b denote a uniformly random element of $A_{i+1} \cap B$. Since every element of B belongs to A_{i+1} with probability $1/(\alpha k)$, independently, even conditioned on $\mathcal{H}_i$, and the event $\mathcal{A}_{i+1}$ simply excludes the possibility that $A_{i+1} \cap B$ is empty, we get that b is a uniformly random element of B when conditioned on $\mathcal{H}_{i+1}$ and $\mathcal{A}_{i+1}$. Furthermore, since h is a bijection, $h(b)$ is a uniformly random element of L_i under the same conditioning, which implies that every element of L_i appears in $L_i - h(b)$ with probability $1 - 1/k$.

Given the above observations, we get

$$\begin{aligned}
\mathbb{E}[f(L_{i+1}) \mid \mathcal{H}_i, \mathcal{A}_{i+1}] &\geq \mathbb{E}[f(L_i - h(b) + b) \mid \mathcal{H}_i, \mathcal{A}_{i+1}] \\
&= \mathbb{E}[f(L_i - h(b)) \mid \mathcal{H}_i, \mathcal{A}_{i+1}] + \mathbb{E}[f(b \mid L_i - h(b)) \mid \mathcal{H}_i, \mathcal{A}_{i+1}] \\
&= \mathbb{E}[f(L_i - h(b)) \mid \mathcal{H}_i, \mathcal{A}_{i+1}] + \mathbb{E}[\tfrac{1}{k}\textstyle\sum_{u \in B} f(u \mid L_i - h(u)) \mid \mathcal{H}_i, \mathcal{A}_{i+1}] \\
&\geq (1 - 1/k) \cdot \mathbb{E}[f(L_i) \mid \mathcal{H}_i, \mathcal{A}_{i+1}] + \tfrac{1}{k} f(\varnothing) + \mathbb{E}[\tfrac{1}{k}\textstyle\sum_{u \in B} f(u \mid L_i - h(u)) \mid \mathcal{H}_i, \mathcal{A}_{i+1}] \\
&= (1 - 1/k) \cdot \mathbb{E}[f(L_i) \mid \mathcal{H}_i] + \tfrac{1}{k} f(\varnothing) + \tfrac{1}{k} \cdot \mathbb{E}[\textstyle\sum_{u \in B} f(u \mid L_i - h(u)) \mid \mathcal{H}_i]
\end{aligned}$$

where the second inequality follows from Lemma 44, and the last equality holds since L_i and B are deterministic given $\mathcal{H}_i$. Using the submodularity of f, and recalling that $h(u) = u$ for $u \in L_i \cap B$, we can now lower bound the argument of the second expectation on the rightmost side of the last inequality as follows.

$$\begin{aligned}
\sum_{u \in B} f(u \mid L_i - h(u)) &\geq \sum_{u \in B \setminus L_i} f(u \mid L_i) + \sum_{u \in B \cap L_i} f(u \mid L_i - u) \\
&\geq f(B \mid L_i) + \sum_{u \in B \cap L_i} f(u \mid L_i - u) \enspace .
\end{aligned}$$

The first inequality of the lemma follows by plugging the last inequality into the previous one, and rearranging. Furthermore, the second inequality of the lemma holds since $f(L_i \mid \varnothing) \leq f(L_i)$ and the monotonicity of f implies

$$f(B \mid L_i) + \sum_{u \in B \cap L_i} f(u \mid L_i - u) \geq f(B \mid L_i) \geq f(B) - f(L_i) \enspace . \qquad \blacktriangleleft$$

Technically, Lemma 17 suffices to prove our results. However, it is useful to also prove the following theorem, which reproves a result due to [33]. To understand this theorem we need to make two observations.

- We would like to chose α on the order of $1/\varepsilon$ to guarantee that the error term diminishes with ε. However, we also need to guarantee that αk is integral (which is necessary for Algorithm 4). The value we choose for α in Theorem 46 is designed to satisfy these two requirements.
- As given, Algorithm 4 requires a base L_0 of $\mathcal{M}$ as input. Since we do not care about the value of this base in Theorem 46 (we care about it in the next section), we can mimic having such a base using the following idea. First, we pretend to add k dummy elements to the ground set such that (i) the dummy elements do not affect the value of any set according to f; and (ii) a set that includes dummy elements is independent in the matroid constraint if it is independent when the dummy elements are removed, and its original size before the removal is at most k (see [5] for a proof that adding such dummy elements does not affect the properties we assume for the objective function and constraint). Once the dummy element are added, we can choose L_0 to simply be the base consisting of the k dummy elements.

▶ **Theorem 46.** *For every $\varepsilon \in [0,1]$, setting $\alpha = \lceil k/\varepsilon \rceil / k$ and initializing L_0 to be the set of dummy elements as described above makes Algorithm 4 a semi-streaming algorithm guaranteeing $\frac{1}{2}(1-1/e^2) - O(\varepsilon)$ approximation and storing $O(k/\varepsilon)$ elements.*

Proof. Recall that $1 - e^{-1/\alpha} \leq \mathbb{P}[\mathcal{A}_{i+1}] = 1 - (1-1/(\alpha k))^k \leq 1/\alpha$. Thus, for every integer $0 \leq i < \alpha k$,

$$\begin{aligned}
\mathbb{E}[f(L_{i+1}) \mid \mathcal{H}_i] &= \Pr[\mathcal{A}_{i+1}] \cdot \mathbb{E}[f(L_{i+1}) \mid \mathcal{H}_i, \mathcal{A}_{i+1}] + \Pr[\neg\mathcal{A}_{i+1}] \cdot \mathbb{E}[f(L_{i+1}) \mid \mathcal{H}_i, \neg\mathcal{A}_{i+1}] \\
&\geq \Pr[\mathcal{A}_{i+1}] \cdot \mathbb{E}[f(L_{i+1}) \mid \mathcal{H}_i, \mathcal{A}_{i+1}] + \Pr[\neg\mathcal{A}_{i+1}] \cdot \mathbb{E}[f(L_i) \mid \mathcal{H}_i] \\
&\geq \Pr[\mathcal{A}_{i+1}] \cdot \left((1-\tfrac{2}{k}) \cdot \mathbb{E}[f(L_i) \mid \mathcal{H}_i] + \tfrac{1}{k} f(\mathrm{OPT})\right) + \Pr[\neg\mathcal{A}_{i+1}] \cdot \mathbb{E}[f(L_i) \mid \mathcal{H}_i] \\
&\geq \left(1 - \frac{2}{\alpha k}\right) \cdot \mathbb{E}[f(L_i) \mid \mathcal{H}_i] + \tfrac{1}{k}(1-e^{-1/\alpha}) \cdot f(\mathrm{OPT})
\end{aligned}$$

where the first inequality follows from the facts that the algorithm only increases the value of its solution and $\mathcal{H}_i$ completely determines L_i, and the second inequality follows from Lemma 17 by choosing $B = \mathrm{OPT}$. The law of total expectation allows us to remove the conditioning on $\mathcal{H}_i$ from both sides of the last inequality, which yields (by repeated applications of the last inequality and using the fact that $f(L_0) \geq 0$) the inequality

$$\begin{aligned}
\mathbb{E}[f(L_i)] &\geq \sum_{j=1}^{i} \left(1 - \frac{2}{\alpha k}\right)^{i-j} \cdot \tfrac{1}{k}(1-e^{-1/\alpha}) \cdot f(\mathrm{OPT}) \\
&\geq \frac{1-(1-2/(\alpha k))^i}{1-(1-2/(\alpha k))} \cdot \tfrac{1}{k}(1-e^{-1/\alpha}) \cdot f(\mathrm{OPT}) \\
&= \tfrac{\alpha}{2}(1-e^{-1/\alpha}) \cdot \left(1-(1-2/(\alpha k))^i\right) \cdot f(\mathrm{OPT}) \ .
\end{aligned}$$

To simplify this inequality, we observe that $1 - 2/(\alpha k) \leq e^{-2/(\alpha k)}$ and $\alpha(1-e^{-1/\alpha}) \geq \alpha(1/\alpha - 1/\alpha^2) = 1 - 1/\alpha$, which yields

$$\mathbb{E}[f(L_i)] \geq \tfrac{1}{2}(1-e^{-2i/(\alpha k)} - 1/\alpha) \cdot f(\mathrm{OPT}) = \tfrac{1}{2}(1-e^{-2i/(\alpha k)} - O(\varepsilon)) \cdot f(\mathrm{OPT}) \ .$$

The theorem now follows by plugging $i = \alpha k$ into this inequality. ◀

Let us now consider a multi-pass algorithm (given as Algorithm 7) obtained by running Algorithm 4 $\Theta(\varepsilon^{-1}\log\varepsilon^{-1})$ times, feeding the output of each execution as the input for the next execution. As promised above (in Section 6), we show that the algorithm obtained in this way outputs a solution whose expected value is almost as good as some ε-approximate local maximum. Algorithm 7 gets $\varepsilon \in (0, 1/2]$ as a parameter.

Algorithm 7 MULTIPLE LOCAL SEARCH PASSES FOR RANDOM STREAMS (ε)

1: Let $r = \lceil 2\varepsilon^{-1} \ln \varepsilon^{-1} \rceil$.
2: Find a base T_0 of M using a single pass.
3: **for** $j = 1$ **to** r **do**
4: Let T_j be the output of Algorithm 6 when given $L_0 = T_{j-1}$ and $\alpha = \lceil k/\varepsilon \rceil / k$.
5: **return** T_r.

We begin the analysis of Algorithm 7 by showing that the expected value of the solution of Algorithm 4 increases significantly in every window as long as this solution is not an approximate local maximum.

▶ **Observation 47.** *Consider some window w_i in an execution of Algorithm 4 done within Algorithm 7, and let $\mathcal{E}$ be the event that the solution L_i of the algorithm at the beginning this window is not an ε-approximate local maximum, then*

$$\mathbb{E}[f(L_{i+1}) - f(L_i) \mid \mathcal{E}] \geq \frac{\varepsilon}{2\alpha k} \cdot f(\mathrm{OPT} \mid \varnothing) \ .$$

Proof. Fix an history $\mathcal{H}_i$ that implies $\mathcal{E}$ (since $\mathcal{H}_i$ completely determines L_i, it also determines $\mathcal{E}$). By the law of total expectation, the lemma will follow if we can prove

$$\mathbb{E}[f(L_{i+1}) - f(L_i) \mid \mathcal{H}_i] \geq \varepsilon \cdot f(\mathrm{OPT} \mid \varnothing) \ .$$

Therefore, in the rest of this proof we concentrate on proving this inequality.

Since L_i is not an ε-local maximum under $\mathcal{H}_i$, we can choose B to be a base such that

$$f(L_i \mid \varnothing) \leq f(B \mid L_i) + \sum_{B \cap L_i} f(u \mid L_i - u) - \varepsilon \cdot f(OPT \mid \varnothing) \ .$$

Then, Lemma 17 implies

$$\begin{aligned}\mathbb{E}[f(L_{i+1}) - f(L_i) \mid \mathcal{H}_i, \mathcal{A}_{i+1}] &\geq \tfrac{1}{k}\mathbb{E}\left[f(B \mid L_i) + \sum_{u \in B \cap L_i} f(u \mid L_i - u) - f(L_i \mid \varnothing) \;\middle|\; \mathcal{H}_i \right] \\ &\geq \tfrac{1}{k}\mathbb{E}[\varepsilon \cdot f(\mathrm{OPT} \mid \varnothing) \mid \mathcal{H}_i] = \frac{\varepsilon}{k} \cdot f(\mathrm{OPT} \mid \varnothing) \ .\end{aligned}$$

We are now ready to prove the observation. By the law of total expectation and the fact that the value of the solution of Algorithm 4 never decreases,

$$\begin{aligned}\mathbb{E}[f(L_{i+1}) - f(L_i) \mid \mathcal{H}_i] \geq{} & \Pr[A_i \mid \mathcal{H}_i] \cdot \mathbb{E}[f(L_{i+1}) - f(L_i) \mid \mathcal{H}_i, \mathcal{A}_i] \\ &\geq (1 - e^{-1/\alpha}) \cdot (\varepsilon/k) \cdot f(\mathrm{OPT} \mid \varnothing) \geq \frac{\varepsilon(1 - 1/\alpha)}{\alpha k} \cdot f(\mathrm{OPT} \mid \varnothing) \\ &\geq \frac{\varepsilon(1-\varepsilon)}{\alpha k} \cdot f(\mathrm{OPT} \mid \varnothing) \geq \frac{\varepsilon}{2\alpha k} \cdot f(\mathrm{OPT} \mid \varnothing) \ ,\end{aligned}$$

where the last inequality holds since $\varepsilon \leq 1/2$. ◀

Let D' be an ε-approximate local maximum whose value according to f is minimal among all ε-approximation local maxima.

▶ **Lemma 48.** *The output T_r of Algorithm 7 obeys $\mathbb{E}[f(T_r \mid \varnothing)] \geq (1 - \varepsilon) \cdot f(D' \mid \varnothing)$.*

Proof. Consider the setting described in Observation 47. By a Markov like argument, the probability of the event $\mathcal{E}$ is at least $1 - \mathbb{E}[f(L_i \mid \varnothing)]/f(D' \mid \varnothing)$ because the event $\mathcal{E}$ happens whenever $f(L_i \mid \varnothing) < f(D' \mid \varnothing)$. Therefore,

$$\begin{aligned}\mathbb{E}[f(L_{i+1}) - f(L_i)] \geq{} & \Pr[\mathcal{E}] \cdot \mathbb{E}[f(L_{i+1}) - f(L_i) \mid \mathcal{E}] \\ &\geq \max\left\{0, \left(1 - \frac{\mathbb{E}[f(L_i \mid \varnothing)]}{f(D' \mid \varnothing)}\right)\right\} \cdot \frac{\varepsilon}{2\alpha k} \cdot f(OPT \mid \varnothing) \\ &\geq \max\left\{0, \left(1 - \frac{\mathbb{E}[f(L_i \mid \varnothing)]}{f(D' \mid \varnothing)}\right)\right\} \cdot \frac{\varepsilon}{2\alpha k} \cdot f(D' \mid \varnothing) \\ &\geq \frac{\varepsilon}{2\alpha k} \cdot \{f(D' \mid \varnothing) - \mathbb{E}[f(L_i \mid \varnothing)]\} \ ,\end{aligned}$$

where the penultimate inequality holds since the inequality $f(\mathrm{OPT}) > f(D')$ follows from the fact that OPT is an optimal solution with respect to f. Rearranging this inequality yields,

$$f(D' \mid \varnothing) - \mathbb{E}[f(L_{i+1} \mid \varnothing)] \leq \left(1 - \frac{\varepsilon}{2\alpha k}\right) \cdot \{f(D' \mid \varnothing) - \mathbb{E}[f(L_i \mid \varnothing)]\} \enspace .$$

The above inequality applies to every window in every one of the r executions of Algorithm 4 that are used by Algorithm 7. Since there are $r\alpha k$ such windows in all these executions of Algorithm 4, combining the inequalities corresponding to all them yields

$$\begin{aligned} f(D' \mid \varnothing) - \mathbb{E}[f(T_r \mid \varnothing)] &\leq \left(1 - \frac{\varepsilon}{2\alpha k}\right)^{rk\alpha} \cdot \{f(D' \mid \varnothing) - \mathbb{E}[f(T_0) \mid \varnothing]\} \\ &\leq e^{-\varepsilon r/2} \cdot \{f(D' \mid \varnothing) - \mathbb{E}[f(T_0) \mid \varnothing]\} \leq e^{-\varepsilon r/2} \cdot f(D' \mid \varnothing) \\ &\leq e^{-\ln \varepsilon^{-1}} \cdot f(D' \mid \varnothing) = \varepsilon \cdot f(D' \mid \varnothing) \enspace , \end{aligned}$$

where the penultimate inequality follows from the monotonicity of f. The lemma now follows by rearranging the last inequality. ◀

The last lemma completes the proof of Proposition 13 since Algorithm 7 uses $O(\varepsilon^{-1} \log \varepsilon^{-1})$ passes (one pass for each execution of Algorithm 4) and stores only a single solution in addition to the $O(\alpha k) = O(k/\varepsilon)$ elements stored by each execution of Algorithm 4.[9]

I Extending Algorithm 7 to p-Matchoid Constraints

In this section we prove Theorem 4, which we repeat here for convenience.

▶ **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 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 $O(p^{-1} \log \varepsilon^{-1})$.

Algorithm 8 is a generalization of Algorithm 4 for a matchoid constraint. The one key difference between the algorithms is that adding an element u to a solution L_i may cause the removal of up to p elements because u can conflict with at most one element in each one of the p matroids it is a member of. Additionally, while Algorithm 8 requires the input set L_0 to be independent in the matchoid constraint, it does not require it to be a base (unlike Algorithm 4, which does require that).

The analysis of Algorithm 8 is identical to the analysis of Algorithm 4 up to (but excluding) Lemma 17. Therefore, we begin by proving the analog of the last lemma given below as Lemma 50.

▶ **Lemma 49** (Lemma 2.2 of [6]). *Let $g\colon 2^{\mathcal{N}} \to \mathbb{R}_{\geq 0}$ be a non-negative submodular function. Further, let R be a random subset of $T \subseteq \mathcal{N}$ in which every element occurs with probability at least p (not necessarily independently). Then, $\mathbb{E}[g(R)] \geq p \cdot g(T)$.*

[9] A technical issue is that we assume in the analysis of Algorithm 7 that $\varepsilon \leq 1/2$. However, this assumption can be dropped by simply replacing ε with $1/2$ at the beginning of the algorithm if ε happens to be larger.

Algorithm 8 p-MatchoidStream(α, L_0)

1: Partition $\mathcal{N}$ into windows $w_1, w_2, \ldots, w_{\alpha k}$.
2: Let $H \leftarrow \varnothing$.
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 $u^\star$ and $S_r^\star$ be element and set, respectively, maximizing $f(L_i \setminus S_r^\star + u_r)$ subject to the constraints: $u^\star \in C_i$, $S_r^\star \subseteq L_i$ and $L_i \setminus S_r^\star + u^\star \in \mathcal{I}$.
7: **if** $f(L_i) < f(L_i \setminus S_r^\star + u^\star)$ **then**
8: Update $H \leftarrow H + u^\star$.
9: Let $L_{i+1} \leftarrow L_i \setminus S_r^\star + u^\star$.
10: **return** $L_{\alpha k}$.

▶ **Lemma 50.** *For every integer $0 \leq i < \alpha k$,*

$$\mathbb{E}[f(L_{i+1}) - f(L_i) \mid \mathcal{H}_i, \mathcal{A}_{i+1}] \geq \tfrac{1}{k}\mathbb{E}[f(B) - (p+1) \cdot f(L_i) \mid \mathcal{H}_i] \enspace .$$

Proof. For every integer $1 \leq \ell \leq q$, we would like to apply Lemma 45 with $S = B \cap \mathcal{N}_\ell$ and $T = L_i \cap \mathcal{N}_\ell$ to get a bijection $h_\ell \colon B \to L_i$ with the properties specified in the lemma with respect to the matroid M_i. This cannot be immediately done because $L_i \cap \mathcal{N}_\ell$ and $B \cap \mathcal{N}_\ell$. However, if we extend $L_i \cap \mathcal{N}_\ell$ and $B \cap \mathcal{N}_\ell$ to bases of M_i in an arbitrary way, and then apply Lemma 45, then we can get an injective function $h_\ell \colon (B \cap \mathcal{N}_\ell) \to \mathcal{N}_\ell$ such that $L_i \cap \mathcal{N}_\ell - h(u) + u \in \mathcal{I}$ for every element $u \in B \cap \mathcal{N}_\ell$.

Let b denote now a uniformly random element of $A_{i+1} \cap B$. Since every element of B belongs to A_{i+1} with probability $1/(\alpha k)$, independently, even conditioned on $\mathcal{H}_i$, and the event $\mathcal{A}_{i+1}$ simply excludes the possibility that $A_{i+1} \cap B$ is empty, we get that b is a uniformly random element of B when conditioned on $\mathcal{H}_{i+1}$ and $\mathcal{A}_{i+1}$. Furthermore, since h_ℓ is an injective function for every $\ell \in [q]$ and every element $u \in \mathcal{N}$ belongs to $\mathcal{N}_\ell$ for at most p different values of ℓ, the probability that $\bigcup_{\ell : b \in \mathcal{N}_\ell}\{h(b)\}$ contains some element $u \in \mathcal{N}$ is at most $p/|B|$. Therefore, if we denote $U(b) \coloneqq \bigcup_{\ell : b \in \mathcal{N}_\ell}$, then every element of L_i appears in $L_i \setminus U(b)$ with probability at least $1 - p/|B|$.

Since $L_i \setminus U(b) + b$ is independent in the matchoid constraint, the above observations imply

$$\begin{aligned}
\mathbb{E}[f(L_{i+1}) \mid \mathcal{H}_i, \mathcal{A}_{i+1}] &\geq \mathbb{E}[f(L_i \setminus U(b) + b) \mid \mathcal{H}_i, \mathcal{A}_{i+1}] \\
&= \mathbb{E}[f(L_i \setminus U(b)) \mid \mathcal{H}_i, \mathcal{A}_{i+1}] + \mathbb{E}[f(b \mid L_i \setminus U(b)) \mid \mathcal{H}_i, \mathcal{A}_{i+1}] \\
&= \mathbb{E}[f(L_i \setminus U(b)) \mid \mathcal{H}_i, \mathcal{A}_{i+1}] + \mathbb{E}[\tfrac{1}{|B|}\textstyle\sum_{u \in B} f(u \mid L_i \setminus U(u)) \mid \mathcal{H}_i, \mathcal{A}_{i+1}] \\
&\geq (1 - p/|B|) \cdot \mathbb{E}[f(L_i) \mid \mathcal{H}_i, \mathcal{A}_{i+1}] + \mathbb{E}[\tfrac{1}{|B|}\textstyle\sum_{u \in B} f(u \mid L_i \setminus U(u)) \mid \mathcal{H}_i, \mathcal{A}_{i+1}] \\
&= (1 - p/|B|) \cdot \mathbb{E}[f(L_i) \mid \mathcal{H}_i] + \tfrac{1}{|B|} \cdot \mathbb{E}[\textstyle\sum_{u \in B} f(u \mid L_i \setminus U(u)) \mid \mathcal{H}_i]
\end{aligned}$$

where the second inequality follows from Lemma 49, and the last equality holds since L_i and B are deterministic given $\mathcal{H}_i$. By the monotonicity and submodularity of f, we can lower bound the argument of the second expectation on the rightmost side of the last inequality as follows.

$$\sum_{u \in B} f(u \mid L_i \backslash U(u)) \geq \sum_{u \in B \backslash L_i} f(u \mid L_i \backslash U(u)) \geq \sum_{u \in B \backslash L_i} f(u \mid L_i) \geq f(B \mid L_i) \geq f(B) - f(L_i) \enspace .$$

Plugging the last inequality into the previous one, we get

$$\mathbb{E}[f(L_{i+1}) - f(L_i) \mid \mathcal{H}_i, \mathcal{A}_{i+1}] \geq \tfrac{1}{|B|}\mathbb{E}[f(B) - (p+1)f(L_i) \mid \mathcal{H}_i] \ .$$

If $\mathbb{E}[f(B) - (p+1)f(L_i) \mid \mathcal{H}_i] \geq 0$ then this inequality implies the lemma since the size of the independent set B cannot exceed the rank k of the matchoid. Otherwise, the lemma holds since Algorithm 8 guarantees $f(L_{i+1}) \geq f(L_i)$. ◀

The last lemma allows us to reprove also the following result due to [33], which is a generalisation of Theorem 46.

▶ **Theorem 51.** *For every $\varepsilon \in [0, 1]$, setting $\alpha = \lceil k/\varepsilon \rceil / k$ and initializing L_0 to be the empty set makes Algorithm 8 a semi-streaming algorithm guaranteeing $\frac{1}{p+1}(1 - 1/e^{p+1}) - O(\varepsilon)$ approximation and storing $O(k/\varepsilon)$ elements.*

Proof. Repeating the initial stages of the proof of Theorem 46, but using Lemma 50 instead of Lemma 17, we can get, for every integer $0 \leq i < \alpha k$,

$$\mathbb{E}[f(L_{i+1}) \mid \mathcal{H}_i] \geq \left(1 - \frac{p+1}{\alpha k}\right) \cdot \mathbb{E}[f(L_i) \mid \mathcal{H}_i] + \tfrac{1}{k}(1 - e^{-1/\alpha}) \cdot f(\mathrm{OPT}) \ .$$

The law of total expectation allows us to remove the conditioning on $\mathcal{H}_i$ from both sides of the last inequality, which yields (by repeated applications of the inequality and observing that $f(L_0) \geq 0$) the inequality

$$\begin{aligned}\mathbb{E}[f(L_i)] &\geq \sum_{j=1}^{i} \left(1 - \frac{p+1}{\alpha k}\right)^{i-j} \cdot \tfrac{1}{k}(1 - e^{-1/\alpha}) \cdot f(\mathrm{OPT}) \\ &\geq \frac{1 - (1 - (p+1)/(\alpha k))^i}{1 - (1 - (p+1)/(\alpha k))} \cdot \tfrac{1}{k}(1 - e^{-1/\alpha}) \cdot f(\mathrm{OPT}) \\ &= \tfrac{\alpha}{p+1}(1 - e^{-1/\alpha}) \cdot \left(1 - (1 - (p+1)/(\alpha k))^i\right) \cdot f(\mathrm{OPT}) \ .\end{aligned}$$

To simplify this inequality, we observe that $1 - (p+1)/(\alpha k) \leq e^{-(p+1)/(\alpha k)}$ and $\alpha(1 - e^{-1/\alpha}) \geq \alpha(1/\alpha - 1/\alpha^2) = 1 - 1/\alpha$, which yields

$$\mathbb{E}[f(L_i)] \geq \tfrac{1}{p+1}(1 - e^{-i(p+1)/(\alpha k)} - 1/\alpha) \cdot f(\mathrm{OPT}) = \tfrac{1}{p+1}(1 - e^{-i(p+1)/(\alpha k)} - O(\varepsilon)) \cdot f(\mathrm{OPT}) \ . \tag{12}$$

The theorem now follows by plugging $i = \alpha k$ into this inequality. ◀

We can extend Algorithm 8 into a multi-pass algorithm in the same way in which Algorithm 4 is extended into the multi-pass algorithm Algorithm 7 in Appendix H.[10] If this is done for r passes, then the resulting algorithm has $r\alpha k$ windows instead of the αk windows of a single pass. Therefore, the expected value of the solution obtained at the end of r passes is given by plugging $r\alpha k$ into Inequality (12), yielding a value of

$$\tfrac{1}{p+1}(1 - e^{-r(p+1)} - O(\varepsilon)) \cdot f(\mathrm{OPT}) \ .$$

[10] A technicality to consider is that, for a matchoid constraint, one cannot use the first pass to construct a base T_0. However, this is not an issue since Algorithm 8 can get any independent set as L_0, which means that it is fine to simply set $T_0 \leftarrow \varnothing$.

Therefore, if we choose the number of passes r to be $\lceil (p+1)^{-1} \log \varepsilon^{-1} \rceil$, the approximation ratio of the multi-pass algorithm becomes

$$\tfrac{1}{p+1}(1 - e^{-\log \varepsilon^{-1}} - O(\varepsilon)) = \frac{1-\varepsilon}{p+1} - O(\varepsilon) = \frac{1}{p+1} - O(\varepsilon) \ ,$$

which completes the proof of the second part of Theorem 4. The first part of the theorem then follows because every matroid is a 1-matchoid, and vice versa.