

Submodular Maximization Subject to Matroid Intersection on the Fly

Moran Feldman ✉


University of Haifa, Israel

Ashkan Norouzi-Fard ✉

Google Research, Zürich, Switzerland

Ola Svensson ✉

EPFL, Lausanne, Switzerland

Rico Zenklusen ✉ 

ETH Zürich, Switzerland

Abstract

Despite a surge of interest in submodular maximization in the data stream model, there remain significant gaps in our knowledge about what can be achieved in this setting, especially when dealing with multiple constraints. In this work, we nearly close several basic gaps in submodular maximization subject to k matroid constraints in the data stream model. We present a new hardness result showing that super polynomial memory in k is needed to obtain an $o(k/\log k)$ -approximation. This implies near optimality of prior algorithms. For the same setting, we show that one can nevertheless obtain a constant-factor approximation by maintaining a set of elements whose size is independent of the stream size. Finally, for bipartite matching constraints, a well-known special case of matroid intersection, we present a new technique to obtain hardness bounds that are significantly stronger than those obtained with prior approaches. Prior results left it open whether a 2-approximation may exist in this setting, and only a complexity-theoretic hardness of 1.91 was known. We prove an unconditional hardness of 2.69.

2012 ACM Subject Classification Theory of computation → Communication complexity

Keywords and phrases Submodular Maximization, Matroid Intersection, Streaming Algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.52

Related Version *Full Version:* <https://arxiv.org/pdf/2204.05154.pdf>

Funding *Moran Feldman:* Research supported in part by the Israel Science Foundation (ISF) grant no. 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

A set function $f: 2^{\mathcal{N}} \rightarrow \mathbb{R}$ over a ground set \mathcal{N} is *submodular* if

$$f(u \mid A) \geq f(u \mid B) \quad \text{for all } A \subseteq B \subseteq \mathcal{N} \text{ and } u \in \mathcal{N} \setminus B,$$

where, for a subset $S \subseteq \mathcal{N}$ and an element $u \in \mathcal{N}$, we denote by $f(u \mid S) := f(S \cup \{u\}) - f(S)$ the marginal contribution of u with respect to S . We say that f is *monotone* if $f(A) \leq f(B)$ for any $A \subseteq B \subseteq \mathcal{N}$.



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

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 52; pp. 52:1–52:14



Leibniz International Proceedings in Informatics

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

The definition of submodular functions captures the natural property of diminishing returns, and the study of these functions has a rich history in optimization with numerous applications (see, e.g., the book [28]). Already in 1978, Nemhauser, Wolsey, and Fisher showed that a natural greedy algorithm achieves a tight approximation guarantee of $\frac{e}{e-1}$ for selecting the most valuable subset $S \subseteq \mathcal{N}$ of cardinality at most ρ (see [26] for the algorithm’s analysis and [8, 25] for matching hardness results). Since then, significant work has been devoted to extending their result to more general constraints.

A natural generalization of a cardinality constraint is the class of matroid constraints. While matroid constraints are much more expressive than cardinality constraints, both constraints often enjoy the same (or similar) algorithmic guarantees. Indeed, for the problem of maximizing a monotone submodular function subject to a single matroid constraint, Călinescu, Chekuri, Pál, and Vondrák [3] developed the continuous greedy method, and showed that it extends the $\frac{e}{e-1}$ -approximation guarantee to this more general setting. Moreover, for the maximization of a monotone submodular function subject to $k \geq 2$ matroid constraints, there is a $(k+1)$ -approximation guarantee by Fisher, Nemhauser and, Wolsey [11], which was improved to $k + \varepsilon$ by Lee, Sviridenko, and Vondrák [22] when the number k of matroid constraints is considered to be a constant.

While these algorithms are efficient in the traditional sense, i.e., they run in polynomial time in the offline “RAM” model, recent applications in data science and machine learning [21] with very large-scale problem instances have motivated the need for very space-efficient algorithms. In particular, it is interesting to study algorithms whose memory footprint is *independent of the ground set size*.¹ The task of designing such algorithms for (monotone) submodular function maximization has become a very active research area, especially in the context of the popular data stream computational model. Recent progress has resulted in a tight understanding of data stream algorithms² for maximizing monotone submodular functions with a single cardinality constraint: one can obtain a 2-approximation for this problem using a simple “threshold”-based algorithm that requires only $\tilde{O}(\rho)$ memory [2, 19], where ρ is the maximum number of elements allowed in the solution, and this is essentially optimal unless one is willing to have a space complexity that is linear in the size of the ground set [9]. However, our understanding of data stream algorithms for more general constraint families is currently much more limited. Closing this gap for natural settings of multiple matroid constraints is the motivation for our work.

Formally, we term the problem that we study **Submodular Maximization subject to k Matroid Constraints (SMkM)**. In this problem, we are given $k \geq 2$ matroids $M_1 = (\mathcal{N}, \mathcal{I}_1), M_2 = (\mathcal{N}, \mathcal{I}_2), \dots, M_k = (\mathcal{N}, \mathcal{I}_k)$ sharing a common ground set \mathcal{N} , and a non-negative submodular function $f: 2^{\mathcal{N}} \rightarrow \mathbb{R}_{\geq 0}$.³ Our goal is to find a *common independent set* $S \subseteq \mathcal{N}$ (i.e., S is independent in all the matroids) that maximizes $f(S)$. In the data stream

¹ Technically, a logarithmic dependence on the ground set size is unavoidable because, at the very least, the algorithm has to store the indices of the elements in its solution. However, we wish to have a space complexity whose dependence on the ground set size is limited to this unavoidable logarithmic dependence.

² Data stream algorithms are sometimes called “streaming algorithms”; however, in this paper we reserve the term “streaming algorithms” to data streaming algorithms whose space complexity is poly-logarithmic in the natural parameters of the problem.

³ We recall that a matroid $M = (\mathcal{N}, \mathcal{I})$ is a tuple consisting of a finite ground set \mathcal{N} and a nonempty family $\mathcal{I} \subseteq 2^{\mathcal{N}}$ of subsets thereof fulfilling (i) if $I \in \mathcal{I}$ and $J \subseteq I$, then $J \in \mathcal{I}$, and (ii) if $I, J \in \mathcal{I}$ with $|I| < |J|$, then there is an element $e \in J \setminus I$ such that $I \cup \{e\} \in \mathcal{I}$. Moreover, a function $f: 2^{\mathcal{N}} \rightarrow \mathbb{R}_{\geq 0}$, where \mathcal{N} is a finite set, is *submodular* if $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$. This is well-known to be equivalent to the *diminishing returns* property, which states that for $A \subseteq B \subseteq \mathcal{N}$ and $e \in \mathcal{N} \setminus B$, we have $f(A \cup \{e\}) - f(A) \geq f(B \cup \{e\}) - f(B)$.

version of this problem, the elements of the ground set \mathcal{N} appear one by one on a stream, and the algorithm should make a single pass over this stream and construct its output set S . (Some papers allow also algorithms that can do a few sequential passes over the stream; however, we consider the more practical and fundamental single-pass setting.)

The above high-level description of **SMkM** hides some important technical details regarding the way in which the objective function f and the matroid constraints are accessed. In the literature about matroids, it is customary to assume that the access of an algorithm to a matroid is done via an *independence oracle* that, given a set $S \subseteq \mathcal{N}$, indicates whether S is independent in the matroid. This notion can be extended to the intersection of k matroids in two natural ways: (i) having a single *common independence oracle*, which indicates whether S is independent in this intersection (i.e., in all the matroids), or (ii) having k independence oracles, one per matroid. Let us first consider the model with weaker access to the matroids, i.e., the one with a common independence oracle. This model already allows the implementation of a simple algorithm that greedily adds to its solution every element that does not violate feasibility. This natural greedy algorithm is a k -approximation for the special case of **SMkM** in which f is simply the cardinality function (i.e., $f(S) = |S|$) [17, 20] using a space complexity of $\tilde{O}(\rho)$, where ρ is the *common rank* of the k matroids, i.e., the cardinality of a maximum cardinality common independent set in the k matroids of the **SMkM** instance.⁴ We can show that this simple algorithm is almost best possible when access to the matroid is restricted to calls to a common independence oracle, unless one is willing to have a space complexity that is linear in $n := |\mathcal{N}|$.

► **Theorem 1.** *A data stream algorithm for **SMkM**, whose only access to the matroids is via the common independence oracle, and with expected approximation ratio $k - \varepsilon$ (for some $\varepsilon \in [0, k - 1]$), must use $\Omega(\varepsilon^n / k^5 \log k)$ memory. This holds even when the task is to find a maximum size common independent set in k partition matroids, and the common rank of these matroids is k .*

The proof of Theorem 1 is based on carefully defining k matroids such that stream prefixes lead to restricted matroids with many indistinguishable elements. This allows for hiding a large optimal solution. See the full version for details.

Given this inapproximability result, we turn our focus to the model in which we have access to a separate independence oracle for every matroid. A $4k$ -approximation algorithm with space complexity $\tilde{O}(\rho)$ was given for **SMkM** in this model by Chakrabarti and Kale [5] when the objective function f is guaranteed to be monotone, and $O(k)$ -approximation algorithms with similar space complexities were later obtained for the general case by Chekuri et al. [6] and Feldman et al. [10]. Our first main result shows that improving over the approximation guarantees of these algorithms by more than a logarithmic factor requires super polynomial space in k . Specifically, we prove the following theorem.

► **Theorem 2.** *Any data stream algorithm for **SMkM** that finds an α -approximate solution with probability at least $2/3$ uses memory at least $\Omega(e^{k/(8\alpha)} / k^2)$ assuming $\alpha \leq k / (32 \ln k)$. This holds even when the task is to find a maximum size common independent set in k partition matroids, and the common rank of these matroids is $O(\alpha) = O(k / \log k)$.*

The technique to prove Theorem 2 is discussed in Section 3. Interestingly, this technique also implies that any (even preemptive) online algorithm for **SMkM** must also have an approximation ratio of $\Omega(k / \log k)$. We remark that this lower bound is asymptotically the same as

⁴ For general **SMkM**, the state-of-the-art algorithm with a space complexity of $\tilde{O}(\rho)$ obtains a slightly worse approximation ratio of $O(k \log k)$ with a common independence oracle [12]. Moreover, this algorithm is applicable even to the more general class of k -extendible constraints.

the well-known approximation hardness for k -dimensional matching [13], which is a special case of the intersection of k matroids. However, note that this hardness does not carry over to our setting as our model has no restriction on computational power but only on memory.

In light of Theorem 2, it is arguably surprising that one can get essentially a 2-approximation for SMkM with space complexity independent of n .⁵

► **Theorem 3.** *For every $\varepsilon \in (0, 1/\tau)$, there exists a $(2 + O(\varepsilon))$ -approximation data stream algorithm for SMkM with space complexity $\text{poly}(k^{\rho^2 k}, \varepsilon^{-\rho})$.*

For monotone objective functions, Theorem 3 is based on merging ideas that appeared in recent papers by Huang et al. [14] and Huang and Ward [16] (see the full version for details). However, to obtain the same guarantee for non-monotone functions requires an interesting novel guessing scheme. Moreover, this theorem cannot be improved by much. The exponential dependence on k is necessary by Theorem 2, and the approximation ratio cannot be improved, even for a cardinality constraint, without using a linear in n memory because of the inapproximability result of [9]. It is open (even for a single partition matroid constraint) whether the exponential dependence on ρ in Theorem 3 is necessary.

Up to this point, our inapproximability results concentrated on the asymptotic approximation ratio obtainable as a function of k , which is more relevant for large values of k . Small values of k have also been considered extensively in the literature. For example, as aforementioned, the approximation ratio that can be obtained by a data streaming algorithm for a cardinality constraint, which is a special case of SMkM with $k = 1$, was the subject of a long line of research [1, 2, 9, 14, 19, 24] and is now essentially settled. Maximizing a monotone submodular function subject to a bipartite matching constraint is another important special case of SMkM, this time for $k = 2$.

Since ρ is usually polynomially related to n in bipartite matching constraints, the class of algorithms considered interesting for the last problem is more restricted than for general SMkM. Specifically, people are interested in algorithms that use $\tilde{O}(\rho)$ memory. That is, the algorithm does not use more memory (up to logarithmic factors) than what is required to simply store a solution. Such algorithms are known as semi-streaming algorithms. Recently, Levin and Wajc [23] described a semi-streaming algorithm for maximizing a monotone submodular function subject to a bipartite matching constraint which improves over the state-of-the-art for general SMkM with a monotone objective function. They also proved, conditioned on some complexity-theoretic assumption, a lower bound of 1.914 on the approximation ratio that can be obtained by a semi-streaming algorithm for the problem. Our final result improves over this upper bound and is independent of any complexity-theoretic assumption, but does assume that the graph can contain parallel edges (which are distinct elements from the point of view of the submodular objective function); the hardness of [23] applies even when this is not the case.

► **Theorem 4.** *No semi-streaming algorithm can obtain, with probability at least $2/3$, an approximation ratio of 2.692 for maximizing a non-negative monotone submodular function subject to a bipartite matching constraint.*

The last result is obtained by combining known hardness results for semi-streaming algorithms for the maximum cardinality matching problem [18] and submodular function maximization subject to a cardinality constraint [9] in a non-trivial way so that the obtained

⁵ For simplicity, the space complexity stated in Theorem 3 assumes that every element of the ground set can be stored in $O(1)$ space. Without this assumption, we get the unavoidable logarithmic dependence of the space complexity on n . Similarly, we also make the standard assumption that the value of $f(S)$ can be stored in constant space for every set $S \subseteq \mathcal{N}$.

hardness is stronger than what is known for any one of the two problems individually. Moreover, the reduction is general, and another consequence of it is the following: any semi-streaming algorithm for maximizing a monotone submodular function subject to a bipartite matching constraint that has an approximation guarantee better than 3 would yield an improved semi-streaming algorithm for the maximum cardinality bipartite matching problem, which is a longstanding notorious open problem. We refer to reader to the full version for further detail.

2 Preliminaries

In this section we give some additional technical details that are necessary for proving the results stated in Section 1. In Section 2.1 we discuss in more detail the oracles used to access the objective function and constraint matroids; and in Section 2.2 we present a known hard problem from which we often reduce to prove our inapproximability results.

2.1 More details about the access oracles

As mentioned above, the matroid constraints are accessed via either a common independence oracle or k distinct independence oracles, one per matroid. We also need to specify the method used to access the objective function f . In the submodular optimization literature, submodular objective functions such as f are usually accessed via a *value oracle* that, given a set $S \subseteq \mathcal{N}$, returns $f(S)$. In the context of data stream and online algorithms, it is important that the independence and value oracles do not leak information in a way that contradicts our expectations from such algorithms. Accordingly, our algorithms query the oracles only on sets of elements that are explicitly stored in their memory.

The above information leakage issue often makes proving inapproximability results more complicated because such results have to formalize in some way the types of queries that are allowed (i.e., queries that are not considered “leaky”). All our inapproximability results apply to the model used by our algorithmic results; namely, when the algorithm is allowed to query the oracles only on sets of elements that are explicitly stored in its memory – see [15] for a formal statement of this natural model. However, we strive to weaken this assumption, and prove most of our inapproximability results even for algorithms that enjoy a less limited access to the oracles. For example, the proof of Theorem 2 manages to avoid this issue completely using the following technique. The algorithm is given upfront a “super ground set” and fully known objective function and matroids over this super ground set. The real ground set is then chosen as some subset of this super ground set, and only the elements of this real ground set appear in the input stream of the algorithm. Since the challenge that the algorithm has to overcome in this case is to remember which elements belong to the real ground set, the oracles cannot leak important information to the algorithm, and therefore, we allow the algorithm unrestricted access to them.

The situation for Theorem 1 is a bit more involved because of the following observation. If the algorithm is allowed unrestricted access to the common independence oracle, then it can construct k matroids M_1, M_2, \dots, M_k that are consistent with this oracle, which makes the distinction between a single common independence oracle and k independence oracles mute. Therefore, some restriction on the access to the common independence oracle must be used. The (arguably) simplest and most natural restriction of this kind is to allow the algorithm to query the common independence oracle only on subsets that do not include any elements that did not appear in the stream so far; and it turns out that this simple restriction suffices for the proof of Theorem 1 to go through.

It remains to consider our last inapproximability result, namely Theorem 4. Here the elements of the ground set are edges, and the algorithm is given each edge in the form of its two endpoints. Therefore, there is no need for independence oracles. (Formally, this situation is equivalent to the case mentioned above in which there is a “super ground set” that is given upfront to the algorithm, and only part of this super ground set appears in the stream.) Unfortunately, preventing information leakage via the value oracle is more involved. For simplicity, the proof that we give in the full version assumes the same model that we use in our algorithmic results. However, our proof can be extended also to algorithms with a more powerful way to access the objective function f , such as the p -players model described in [9].

2.2 The $\text{CHAIN}_p(n)$ problem

Many of our inapproximability results use reductions to a (hard) problem named $\text{CHAIN}_p(n)$, introduced by Cormode, Dark, and Konrad [7], which is closely related to the Pointer Jumping problem (see [4]). In this problem, there are p players P_1, \dots, P_p . For every $1 \leq i < p$, player P_i is given a bit string $x^i \in \{0, 1\}^n$ of length n , and, for every $2 \leq i \leq p$, player P_i (also) has as input an index $t^i \in \{1, 2, \dots, n\}$. (Note that the convention in this terminology is that the superscript of a string/index indicates the player receiving it.) Furthermore, it is promised that either $x_{t^{i+1}}^i = 0$ for all $1 \leq i < p$ or $x_{t^{i+1}}^i = 1$ for all these i values. We refer to these cases as the 0-case and 1-case, respectively. The objective of the players in $\text{CHAIN}_p(n)$ is to decide whether the input instance belongs to the 0-case or the 1-case. The first player, based on the input bit string x^1 , sends a message M^1 to the second player. Any player $2 \leq i < p$, based on the message it receives from the previous player (i.e., M^{i-1}), the input bit string x^i and index t^i , sends message M^i to the next player. The last player, based on M^{p-1} and t^p , decides if we are in the 0-case or 1-case. Each player has unbounded computational power and can use any (potentially randomized) algorithm. We refer to the collections of the algorithms used by all the players as a protocol. The success probability of a protocol is the probability that its decision is correct, and the communication complexity of a protocol is the size of the maximum message sent (i.e., maximum size of M^1, \dots, M^{p-1}). In [9], the following lower bound was shown for the $\text{CHAIN}_p(n)$ problem, which is very similar to the lower bounds previously proved by [7].

► **Theorem 5** (Theorem 3.3 in [9]). *For any positive integers n and $p \geq 2$, any (potentially randomized) protocol for $\text{CHAIN}_p(n)$ with success probability of at least $2/3$ must have a communication complexity of at least $n/36p^2$. Furthermore, this holds even when instances are drawn from a known distribution $D(p, n)$.*

The distribution $D(p, n)$ referred to by Theorem 5 is simply the uniform distribution over all 0-case and 1-case instances (see the definition of D^p in Appendix C of [9]).

3 Inapproximability for Multiple Independence Oracles

In this section, we prove Theorem 2, which gives a strong inapproximability result for data stream algorithms as a function of the number k of matroids, even in the case when the objective function f is a linear function (unlike in the previous section, here we allow access to the independence oracles of the individual matroids).

► **Theorem 2.** *Any data stream algorithm for SMkM that finds an α -approximate solution with probability at least $2/3$ uses memory at least $\Omega(e^{k/(8\alpha)}/k^2)$ assuming $\alpha \leq k/(32 \ln k)$. This holds even when the task is to find a maximum size common independent set in k partition matroids, and the common rank of these matroids is $O(\alpha) = O(k/\log k)$.*

Note that the above result implies that (i) any data stream algorithm with an approximation guarantee $o(k/\log k)$ requires super polynomial memory in k , and (ii) any data stream algorithm with constant approximation guarantee requires exponential memory in k .

The techniques we use also readily imply hardness for the (preemptive) online version of this problem. In this version, the elements of the ground set \mathcal{N} arrive online, and upon receiving each element the algorithm has to decide either to add this element to the solution it maintains, or to reject the element. If the algorithm accepts an element to its solution, it may remove this element from the solution at a later point; however, a decision to reject an element (or remove it from the solution at a later time) is irrevocable. The algorithm is also required to keep its solution feasible at all times. We have the following hardness in this model.

► **Theorem 6.** *For $k \geq 2$, the competitive ratio of any online algorithm for $SMkM$ against an oblivious adversary is at least $\frac{k}{81 \ln k}$. This holds even when the task is to find a maximum size common independent set in k partition matroids, and the common rank of these matroids is $O(k/\log k)$.*

The key building block for both (streaming and online) hardness results is a “hard” distribution of instances described in Section 3.1. This distribution is then used to prove Theorems 2 and 6 in Sections 3.2 and 3.3, respectively, in a similar way to the proof of Theorem 1.

3.1 Description of hard distribution

Let p be a non-negative integer parameter of the construction. Our instances are subsets of the set $\mathcal{N} = [p]^k$, where we allow multiple elements with the same coordinates (i.e., “multi-subsets”). A set $S \subseteq \mathcal{N}$ is independent in the matroid M_i (for any integer $i \in [k]$) if and only if no two elements of S share the same value in coordinate number i (in other words, $u_i \neq v_i$ for every two distinct elements $u, v \in S$). One can observe that this definition makes M_i a partition matroid. We recall that $S \subseteq \mathcal{N}$ is a *common independent set* if it is independent in all matroids; otherwise, we will refer to it as *dependent*. Note that the common rank of the k matroids M_1, \dots, M_k is p .

In Algorithm 1 we describe a procedure for sampling $p-1$ many subsets $S_1, S_2, \dots, S_{p-1} \subseteq [p]^k$ and $p-1$ “hidden” optimal elements $o^{(1)} \in S_1, o^{(2)} \in S_2, \dots, o^{(p-1)} \in S_{p-1}$. In every iteration $r = 1, \dots, p-1$, the algorithm first forms S_r by sampling m elements independently and uniformly from those elements that form a common independent set with $\{o_1, \dots, o_{r-1}\}$. That is, S_r contains m uniformly random samples with replacements from

$$\{u \in \mathcal{N} \mid \forall 1 \leq i < r, 1 \leq j \leq k \ o_j^{(i)} \neq u_j\} .$$

Then, *after* the selection of S_r , the algorithm samples $o^{(r)}$ uniformly at random among the m elements in S_r .

We remark that the algorithm with small probability may sample the same element more than once when forming the set S_r . When this happens, we consider these samples to be unique elements on the stream (that are dependent). This allows us to simplify the notation in the following as each set S_r is now guaranteed to contain exactly m elements. Formally, this corresponds to extending the ground set \mathcal{N} by making m copies u^1, \dots, u^m of each element $u \in \mathcal{N}$, and whenever an element u is sampled i times, we include the copies u^1, \dots, u^i .

Algorithm 1 HARD INSTANCE GENERATION (p, m) .

- 1: **for** $r = 1$ **to** $p - 1$ **do**
 - 2: Obtain S_r by sampling m elements uniformly and with replacement from

$$\{u \in \mathcal{N} \mid \forall_{1 \leq i < r, 1 \leq j \leq k} o_j^{(i)} \neq u_j\} .$$
 - 3: Select $o^{(r)}$ from S_r uniformly at random.
 - 4: Output S_1, \dots, S_{p-1} and $o^{(1)}, \dots, o^{(p-1)}$.
-

We refer to Algorithm 1 as “HARD INSTANCE GENERATION” as it is the basic building block of our hardness results in both the online and streaming models. More specifically, our hardness result for the online model is based on the (random) stream obtained by first feeding the elements in S_1 (in any order), then S_2 (in any order), and so on until S_{p-1} is fed. The intuition is that when the algorithm has only seen the elements in S_1, \dots, S_i , it has no information about the selection of $o^{(i)}$ and so any online algorithm is unlikely to have saved the element $o^{(i)}$. In addition, while $\{o^{(1)}, \dots, o^{(p-1)}\}$ is a common independent set by construction, we prove (see Lemma 8 below) that any other two elements are likely to be dependent. This creates the “gap” between the values of the solution $\{o^{(1)}, o^{(2)}, \dots, o^{(p-1)}\}$ and a solution with any other elements, which in turn yields the desired hardness result. For our hardness in the data stream model, we forward a subset of the above-mentioned stream, and the difficulty for a low-space streaming algorithm is to “remember” whether the special elements $o^{(1)}, o^{(2)}, \dots, o^{(p-1)}$ appeared in the stream. This is formalized in the next sections.

We complete this section by proving that, with good probability, any large solution must contain the hidden elements $o^{(1)}, \dots, o^{(p-1)}$.

► **Definition 7.** *We say that the output of Algorithm 1 is successful if any two elements $e, f \in S_1 \cup S_2 \cup \dots \cup S_{p-1} \setminus \{o_1, \dots, o_{p-1}\}$ are dependent, i.e., there is a coordinate $i \in [k]$ such that $e_i = f_i$.*

► **Lemma 8.** *The output of Algorithm 1 is successful with probability at least $1 - \binom{pm}{2} e^{-k/p}$.*

Proof. Consider two elements $u \in S_{r_1} \setminus \{o_{r_1}\}$ and $v \in S_{r_2}$ with $r_1 \leq r_2$. As $u \neq o_{r_1}$, each coordinate of u equals that of v with probability at least $1/(p - r_1 + 1) \geq 1/p$. Now, as there are k coordinates, and each coordinate of u is sampled independently at random,

$$\Pr[\{u, v\} \text{ is a common independent set}] \leq (1 - 1/p)^k \leq e^{-k/p} .$$

The lemma now follows by taking the union bound over all possible pairs u, v ; the number of such pairs is upper bounded by $\binom{(p-1)m}{2}$. ◀

3.2 Hardness for online algorithms

Let $p = \lceil k/(27 \ln k) \rceil + 1$ and $m = k^3$. We will prove that the competitive ratio of any online algorithm is at least $k/(81 \ln k)$. We assume throughout that k is such that $k/(81 \ln k) > 1$. This is without loss of generality since the statement is trivial if $k/(81 \ln k) \leq 1$. We shall consider the following distribution of instances. Run Algorithm 1 with the parameters p and m to obtain sets S_1, \dots, S_{p-1} (and hidden elements $o^{(1)}, \dots, o^{(p-1)}$), and construct an input stream in which the elements of S_1 appear first (in any order) followed by those in S_2 and so on until the elements in S_{p-1} appear. We will show that any deterministic online algorithm ALG cannot be $((p-1)/3)$ -competitive on this distribution of instances. Theorem 6 then follows via Yao’s principle.

To analyze the competitive ratio of ALG, let O be the event that the output of ALG contains some element of $\{o^{(1)}, o^{(2)}, \dots, o^{(p-1)}\}$, and let S be the event that the instance is successful. Then, if we use $|ALG|$ to denote the size of the independent set outputted by ALG,

$$\mathbb{E}[|ALG|] \leq (1 - \Pr[\neg O, S]) \cdot k + 1 ,$$

where the inequality holds because any solution has size at most k , and if the algorithm fails to identify any element in $\{o^{(1)}, \dots, o^{(p-1)}\}$, then it can produce solution of size at most 1 for a successful instance.

Note that since $k/(81 \ln k) > 1$ we have that $p \leq 3 \cdot k/(27 \ln k) = k/(9 \ln k)$. Now, by Lemma 8 and the selection of p and m , we have

$$\Pr[S] \geq 1 - \binom{pm}{2} e^{-k/p} \geq 1 - k^8 \cdot e^{-9 \ln k} = 1 - 1/k .$$

To bound $\Pr[\neg O]$, note that the set S_r contains no information about $o^{(r)}$ because $o^{(r)}$ is selected uniformly at random from S_r . Moreover, when all m elements of S_r have been inspected in the stream, ALG keeps at most p independent elements. Any one of these elements is $o^{(r)}$ with probability at most p/m . In other words, the algorithm selects $o^{(r)}$ with probability at most p/m . Hence, by the union bound, the algorithm has selected any of the $p-1$ elements $\{o^{(1)}, \dots, o^{(p-1)}\}$ with probability at most $p/m \cdot (p-1) \leq 1/k$. We thus have $\Pr[\neg O] \geq 1 - 1/k$, and together with the above proved inequality $\Pr[S] \geq 1 - 1/k$, we get via the union bound

$$\mathbb{E}[|ALG|] \leq (1 - \Pr[\neg O, S]) \cdot k + 1 \leq (2/k) \cdot k + 1 = 3 .$$

Since the stream contains a solution $\{o^{(1)}, \dots, o^{(p-1)}\}$ of size $p-1$, this implies that ALG is not better than $((p-1)/3)$ -competitive, which in turn implies Theorem 6.

3.3 Hardness for streaming algorithms

Let ALG be a data stream algorithm for finding a set of maximum cardinality subject to k partition matroid constraints. Further suppose that ALG has the following properties:

- ALG uses memory M ;
- ALG outputs an α -approximate solution with probability at least $2/3$, where $\alpha \leq \frac{k}{32 \ln k}$ (note that α is also lower bounded by 1 since it is an approximation ratio).

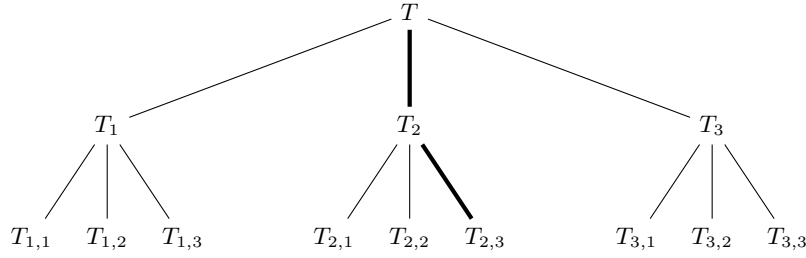
Select $p = \lceil 3 \cdot \alpha \rceil$, and let m be the smallest power of two such that $m \geq e^{k/(8\alpha)}$. Note that this selection satisfies

$$p \in [3 \cdot \alpha, 4 \cdot \alpha] , \quad m \in [e^{k/(8\alpha)}, 2e^{k/(8\alpha)}] , \quad \text{and} \quad 8 \cdot p \leq k \leq m .$$

We will use ALG to devise a protocol for the $\text{CHAIN}_p(m)$ problem that succeeds with probability at least $2/3$ and has communication complexity at most $M + p \log_2 m$. Combining this reduction with Theorem 5 then yields Theorem 2, i.e., that any such algorithm ALG must have a memory footprint M that is at least $\Omega(e^{k/(8\alpha)}/k^2)$.

3.3.1 Description of protocol

We use ALG to obtain Protocol 2 for $\text{CHAIN}_p(m)$. The protocol consists of two phases: a precomputation phase that is independent of the $\text{CHAIN}_p(m)$ instance, followed by a description of the messages of the players.



■ **Figure 1** A tree representation of the sets computed during precomputation for $m = p = 3$.

3.3.1.1 Precomputation phase

In the precomputation phase, the players use shared random coins⁶ to generate instances from the same distribution produced by Algorithm 1 for all possible values of $t^2, \dots, t^p \in [m]$ in an instance of $\text{CHAIN}_p(m)$. Specifically, first a set T is sampled from the same distribution as the set S_1 produced by Algorithm 1. The elements of T are then randomly permuted. For $j \in [m]$, we let $T(j)$ denote the j -th element in the obtained ordered set. The reason that the elements in T are randomly permuted is to make sure that for any fixed $t^2 \in [m]$, the element $T(t^2)$ is uniformly random, and thus, has the same distribution as $o^{(1)}$ in Algorithm 1. Following the choice of $S_1 = T$ and $o^{(1)} \in S_1$, Algorithm 1 proceeds to sample S_2 to be m random elements that are independent with respect to $o^{(1)}$. In the precomputation phase we do so for each possible element in T , i.e., we sample sets T_1, T_2, \dots, T_m , one for each possible choice of $t^2 \in [m]$. Then, for each such T_{t^2} we sample m sets $T_{t^2,1}, T_{t^2,2}, \dots, T_{t^2,m}$ for all possible choices of $t^3 \in [m]$ and so on. The sets constructed in the precomputation phase can thus naturally be represented by a tree, where each path from the root T to a leaf corresponds to a particular choice of $t^2, t^3, \dots, t^p \in [m]$. For $m = 3$ and $p = 3$, this tree is depicted in Figure 1. The thick path corresponds to the case of $t_2 = 2$ and $t_3 = 3$.

As described above, we randomly permute the sets so as to make sure that, for fixed $t^2, t^3, \dots, t^r \in [m]$, the distribution of $o^{(1)}$ is the same as that of $T(t^2)$ and, in general, the distribution of $o^{(i)}$ is the same as that of $T_{t^2, \dots, t^i}(t^{i+1})$. This gives us the following observation.

► **Observation 9.** Fix $t^2, t^3, \dots, t^r \in [m]$. Over the randomness of the precomputation phase, the elements $o^{(1)} = T(t^2), o^{(2)} = T_{t^2}(t^3), \dots, o^{(r-1)} = T_{t^2, \dots, t^{r-1}}(t^r)$ and the sets $S_1 = T, S_2 = T_{t^2}, \dots, S_{r-1} = T_{t^2, \dots, t^{r-1}}$ have the same distribution as the output of Algorithm 1.

The reason the players do this precomputation is that, after they have commonly agreed on the tree-structure of sets (which can be generated using the public coins), it requires little communication to decide on a “hard” instance generated from the same distribution as Algorithm 1. Indeed, Player r only needs to know t^2, \dots, t^r ($r \log_2 m$ bits of information) in-order to know the set T_{t^2, \dots, t^r} .

⁶ We note that the hardness result of $\text{CHAIN}_p(m)$ (Theorem 5) holds when the players have access to public coins, i.e., shared randomness. This is proved, e.g., in Theorem 3.3 of [9]. In general, Newman’s theorem [27] says that we can turn any public coin protocol into a private coin protocol with little (logarithmic) increase in communication.

■ **Protocol 2** Reduction from $\text{CHAIN}_p(m)$ to SMkM .

Precomputation

- 1: Let T be a uniformly random subset of $\mathcal{N} = [p]^k$ of size m .
- 2: Order the elements of T randomly, and let $T(j)$ denote the j -th element.
- 3: **for** $r = 2, \dots, p - 1$ and $t^2, \dots, t^r \in [m]$ **do**
- 4: Identify $o^{(1)} = T(t^2), o^{(2)} = T_{t^2}(t^3), \dots, o^{(r-1)} = T_{t^2, \dots, t^{r-1}}(t^r)$.
- 5: Let T_{t^2, \dots, t^r} be a uniformly random subset of $\{u \in \mathcal{N} \mid \forall 1 \leq i < r, 1 \leq j \leq k \ o_j^{(i)} \neq u_j\}$ of size m .
- 6: Order the elements of T_{t^2, \dots, t^r} randomly, and let $T_{t^2, \dots, t^r}(j)$ denote the j -th element.

Player P_r 's Algorithm for $r = 1, \dots, p - 1$

- 1: Initialize ALG with the received memory state (or initial state if first player).
- 2: Simulate ALG on the elements $T_r = \{T_{t^2, \dots, t^r}(j) \mid j \in [n] \text{ with } x_j^t = 1\}$ given in any order.
- 3: Send to P_{r+1} the values t^2, t^3, \dots, t^r and the memory state of ALG .

Player P_p 's Algorithm

- 1: If ALG with the received memory state returns an independent set of size at least 2, output “1-case”; otherwise, output “0-case”.
-

3.3.1.2 The messages of the players

After generating the (common) sets in the precomputation phase using the public coins, the players now proceed as follows. The first player receives as input x^1 and simulates ALG on the subset $\{T(j) \mid x_j^1 = 1\}$ of T corresponding to the 1-bits. These elements are given to ALG as a stream in any order. Player 1 then sends to Player 2 the message containing the state of ALG after processing this stream of elements.

The second player receives input x^2, t^2 and initializes ALG with the state received from the first player. Then, the elements $\{T_{t^2}(j) \mid x_j^2 = 1\}$ of T_{t^2} that correspond to 1-bits of x^2 are streamed to ALG in any order. Player 2 sends to Player 3 a message containing the state of ALG after processing these elements and the index t^2 . Player r , for $r = 3, \dots, p - 1$, proceeds similarly to Player 2: given input x^r, t^r , ALG is first initialized with the state received from the previous player, and then the elements $\{T_{t^2, \dots, t^r}(j) \mid x_j^r = 1\}$ are streamed to ALG in any order. Notice that Player r knows t^2, \dots, t^{r-1} from the message of the previous player, t^r, x^r from the input, and T_{t^2, \dots, t^r} from the precomputation phase, and so the set $\{T_{t^2, \dots, t^r}(j) \mid x_j^r = 1\}$ can be computed. Finally, Player r sends to Player $r + 1$ a message consisting of the indices t^2, \dots, t^r and the memory state of ALG .

The final player initializes ALG with the received state and asks ALG to return an independent set. If the independent set consists of at least two elements, Player p outputs “1-case”, and otherwise, the output is “0-case”.

3.3.2 Analysis

The messages sent by the players contain the memory state of ALG and at most $p - 2$ indices $t^2, \dots, t^{p-1} \in [m]$. The memory state of ALG is at most M bits by assumption, and each index requires $\log_2 m$ bits. The communication complexity of the protocol is, therefore, upper bounded by $M + p \log_2 m$.

To analyze the success probability of the protocol, we have the following lemma.

- **Lemma 10.** *The instance that the players stream to ALG satisfies the following:*
- In the 1-case, the stream contains $p - 1$ independent elements.
 - In the 0-case, with probability at least $2/3$, any two elements in the stream are dependent.

Proof. In the 1-case, we have $x_{t^2}^1 = x_{t^3}^2 = \dots = x_{t^p}^{p-1} = 1$ and so the elements $T(t^2)$, $T_{t^2}(t^3)$, \dots , $T_{t^2, \dots, t^{p-1}}(t^p)$ are part of the stream. By definition, they form an independent set consisting of $p - 1$ elements.

In the 0-case, we have that $x_{t^2}^1 = x_{t^3}^2 = \dots = x_{t^p}^{p-1} = 0$ and so any two elements e, f in the stream belong to the set $S_1 \cup S_2 \dots, S_{p-1} \setminus \{o^{(1)}, o^{(2)}, \dots, o^{(p-1)}\}$, where $S_1 = T$, $o^{(1)} = T(t^2)$ and $S_j = T_{t^2, \dots, t^j}, o^{(j)} = T_{t^2, \dots, t^j}(t^{j+1})$ for $j = 2, \dots, p - 1$. By Observation 9, we can apply Lemma 8 to obtain that, with probability at least $1 - \binom{pm}{2} e^{-k/p}$, any two elements in the stream are dependent. The statement now follows since the selection of our parameters p, m and k implies

$$\binom{pm}{2} e^{-k/p} \leq m^4 e^{-k/p} \leq 16e^{k/(8\alpha)} \cdot e^{-k/(4\alpha)} = 16e^{-k/(8\alpha)} \leq 1/3 ,$$

where for the first inequality we used $p \leq m$, and for the second inequality we used that $p \leq 4 \cdot \alpha$ and $m \leq 2e^{k/(8\alpha)}$. The last inequality holds for $k \geq 3$ (the case of $k = 2$ can be ignored because it implies $k/(32 \ln k) < 1$, which makes Theorem 2 trivial). \blacktriangleleft

We now argue how the above lemma implies that Protocol 2 has a success probability of $2/3$ in both the 0-case and 1-case. For 0-case instances, we have with probability $2/3$ that any two elements are dependent. Hence, with that probability, there is no way for ALG to return an independent set with more than one element. Thus, the output of Player p is correct with probability at least $2/3$ in the 0-case. In the 1-case, the stream always contains a solution of value $p - 1$. By the assumption that ALG returns an α -approximate solution with probability at least $2/3$, ALG returns an independent set of size at least $(p - 1)/\alpha$ with probability at least $2/3$. This implies that Player p is correct in this case with probability $2/3$ since

$$(p - 1)/\alpha \geq (3 \cdot \alpha - 1)/\alpha \geq 2 .$$

Using ALG we have, thus, devised a protocol for $\text{CHAIN}_p(m)$ that is correct with probability $2/3$ and has a communication complexity that is upper bounded by $M + p \log_2 m$. By Theorem 5, we thus must have $(M + p \log_2 m) \geq m/(36p^2)$. Now using that $p \leq k/8$ and $e^{k/(8\alpha)} \leq m$, we get

$$M \geq \frac{m}{36p^2} - p \log_2 m \geq \frac{64}{36} \frac{m}{k^2} - k^2 \geq \frac{64}{36} \frac{e^{k/(8\alpha)}}{k^2} - k^2 ,$$

which is $\Omega(e^{k/(8\alpha)}/k^2)$ since by assumption on α we have $e^{k/(8\alpha)} \geq k^4$. We have thus proved that the memory usage M of ALG must be at least $\Omega(e^{k/(8\alpha)}/k^2)$ as required by Theorem 2.

References

- 1 Naor Alaluf, Alina Ene, Moran Feldman, Huy L. Nguyen, and Andrew Suh. Optimal streaming algorithms for submodular maximization with cardinality constraints. In *ICALP*, pages 6:1–6:19, 2020. doi:10.4230/LIPIcs.ICALP.2020.6.
- 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 Gruiă 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.

- 4 A. Chakrabarti. Lower bounds for multi-player pointer jumping. *Electronic Colloquium on Computational Complexity*, 14, 2007.
- 5 Amit Chakrabarti and Sagar Kale. Submodular maximization meets streaming: matchings, matroids, and more. *Mathematical Programming*, 154(1):225–247, December 2015.
- 6 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.
- 7 G. Cormode, J. Dark, and C. Konrad. Independent sets in vertex-arrival streams. In *Proceedings of 46th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 45:1–45:14, 2019.
- 8 Uriel Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998.
- 9 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.
- 10 Moran Feldman, Amin Karbasi, and Ehsan Kazemi. Do less, get more: Streaming submodular maximization with subsampling. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 730–740, 2018. URL: <https://proceedings.neurips.cc/paper/2018/hash/d1f255a373a3cef72e03aa9d980c7eca-Abstract.html>.
- 11 Marshall L. Fisher, George L. Nemhauser, and Laurence A. Wolsey. An analysis of approximations for maximizing submodular set functions–II. *Mathematical Programming*, 8:73–87, 1978.
- 12 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>.
- 13 E. Hazan, S. Safra, and O. Schwarz. On the complexity of approximating k -set packing. *Computational Complexity*, 15(1):20–39, 2006. 2006.
- 14 Chien-Chung Huang, Naonori Kakimura, Simon Mauras, and Yuichi Yoshida. Approximability of monotone submodular function maximization under cardinality and matroid constraints in the streaming model. *CoRR*, abs/2002.05477, 2020. [arXiv:2002.05477](https://arxiv.org/abs/2002.05477).
- 15 Chien-Chung Huang, Theophile Thiery, and Justin Ward. Improved multi-pass streaming algorithms for submodular maximization with matroid constraints. *CoRR*, abs/2102.09679, 2021. [arXiv:2102.09679](https://arxiv.org/abs/2102.09679).
- 16 Chien-Chung Huang and Justin Ward. Fpt-algorithms for the l -matchoid problem with linear and submodular objectives. *CoRR*, abs/2011.06268, 2020. [arXiv:2011.06268](https://arxiv.org/abs/2011.06268).
- 17 Tom A. Jenkyns. The efficacy of the “greedy” algorithm. In *South Eastern Conference on Combinatorics, Graph Theory and Computing*, pages pages 341–350, 1976.
- 18 Michael Kapralov. Space lower bounds for approximating maximum matching in the edge arrival model. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 1874–1893. SIAM, 2021. doi:10.1137/1.9781611976465.112.
- 19 Ehsan Kazemi, Marko Mitrovic, Morteza Zadimoghaddam, Silvio Lattanzi, and Amin Karbasi. Submodular streaming in all its glory: Tight approximation, minimum memory and low adaptive complexity. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pages 3311–3320, 2019. URL: <http://proceedings.mlr.press/v97/kazemi19a.html>.
- 20 Bernhard Korte and Dirk Hausmann. An analysis of the greedy heuristic for independence systems. *Annals of Discrete Math.*, 2:65–74, 1978.
- 21 Andreas Krause. Submodularity in machine learning. <http://submodularity.org/>.

- 22 Jon Lee, Maxim Sviridenko, and Jan Vondrák. Submodular maximization over multiple matroids via generalized exchange properties. *Math. Oper. Res.*, 35(4):795–806, 2010. doi: 10.1287/moor.1100.0463.
- 23 Roie Levin and David Wajc. Streaming submodular matching meets the primal-dual method. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1914–1933, 2021. doi: 10.1137/1.9781611976465.114.
- 24 Andrew McGregor and Hoa T. Vu. Better streaming algorithms for the maximum coverage problem. *Theory Comput. Syst.*, 63(7):1595–1619, 2019.
- 25 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.
- 26 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.
- 27 Ilan Newman. Private vs. common random bits in communication complexity. *Inf. Process. Lett.*, 39(2):67–71, 1991.
- 28 Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Algorithms and Combinatorics. Springer Berlin Heidelberg, 1 edition, 2003.