Streaming Submodular Maximization under a k-Set System Constraint

Ran Haba [1] **Ehsan Kazemi** [2 3] **Moran Feldman** [4] **Amin Karbasi** [2]

Abstract

In this paper, we propose a novel framework that converts streaming algorithms for monotone submodular maximization into streaming algorithms for non-monotone submodular maximization. This reduction readily leads to the currently tightest deterministic approximation ratio for submodular maximization subject to a k-matchoid constraint. Moreover, we propose the first streaming algorithm for monotone submodular maximization subject to k-extendible and k-set system constraints. Together with our proposed reduction, we obtain $O(k \log k)$ and $O(k^2 \log k)$ approximation ratio for submodular maximization subject to the above constraints, respectively. We extensively evaluate the empirical performance of our algorithm against the existing work in a series of experiments including finding the maximum independent set in randomly generated graphs, maximizing linear functions over social networks, movie recommendation, Yelp location summarization, and Twitter data summarization.

1. Introduction

Submodularity captures an intuitive diminishing returns property where the benefit of an item decreases as the context in which it is considered grows. This property naturally occurs in many applications where items may represent data points, features, actions, etc. Moreover, submodularity is a sufficient condition that leads to an efficient optimization procedure for many discrete optimization problems. The above reasons have led to a surge of applications in machine learning where the gain of discrete choices shows diminishing returns and the optimization can be handled efficiently. Novel examples include non-parametric learning (Mirzasoleiman et al., 2016b), dictionary learning (Das & Kempe, 2011), crowd teaching (Singla et al., 2014), regression under human assistance (De et al., 2019), sequence selection (Tschiatschek et al., 2017; Mitrovic et al., 2019) interpreting neural networks (Elenberg et al., 2017), adversarial attacks (Lei et al., 2019), fairness (Kazemi et al., 2018), social graph analysis (Norouzi-Fard et al., 2018), data summarization (Dasgupta et al., 2013; Tschiatschek et al., 2014; Elhamifar & Clara De Paolis Kaluza, 2017; Kirchhoff & Bilmes, 2014; Mitrovic et al., 2018; Kazemi et al., 2020), fMRI parcellation (Salehi et al., 2017), and DNA sequencing (Libbrecht et al., 2018).

More formally, a set function $f\colon 2^{\mathcal{N}} \to \mathbb{R}_{\geq 0}$ is called **submodular** if for all sets $A \subseteq B \subseteq \mathcal{N}$ and element $u \notin B$ we have $f(A \cup \{u\}) - f(A) \geq f(B \cup \{u\}) - f(B)$. Moreover, a set function is called monotone if $f(A) \leq f(B)$ whenever $A \subseteq B$. The focus of this paper is on maximizing a general submodular function (not necessarily monotone). More concretely, we consider a very general form of constrained submodular maximization, i.e.,

$$\texttt{OPT} = \arg\max_{A \in \mathcal{I}} f(A) \ , \tag{1}$$

where $\mathcal{I}$ represents the set of feasible solutions. For instance, when f is monotone and $\mathcal{I}$ represents a cardinality/size constraint,[1] the celebrated result of (Nemhauser et al., 1978) states that the greedy algorithm achieves a $e/(e-1)$-approximation for this problem, which is known to be optimal (Nemhauser & Wolsey, 1978). In the recent years, there has been a large body of literature aiming at solving Problem (1) in the offline/centralized setting under various types of feasibility constraints such as matroid, k-matchoid, k-extendible system, and k-set system (formal definitions of some of these terms appear in Section 3).

In the offline/centralized setting, the problem of maximizing a (non-monotone) submodular function subject to the above types of constraints is fairly well understood and easy-to-implement algorithms have been proposed. For instance, for maximization under a k-set system constraint, one obtains an approximation ratio of $k + O(\sqrt{k})$ using roughly $\sqrt{k}$ invocations of the natural greedy algorithm and an algorithm for unconstrained submodular maximization. Or, when

[1]Depart. of Mathematics and Computer Science, The Open University of Israel [2]Yale Institute for Network Science, Yale University [3]Now at Google, Zürich, Switzerland [4]Department of Computer Science, University of Haifa, Israel. Correspondence to: Ehsan Kazemi <ehsankazemi@google.com>.

Proceedings of the 37^{th} International Conference on Machine Learning, Online, PMLR 119, 2020.

[1]Formally, $\mathcal{I}$ contains in this case all subsets of $\mathcal{N}$ of size at most ρ for some value ρ.

the constraint is a k-extendible system, running the greedy algorithm only once over a carefully subsampled ground set achieves a $k + 3$ approximation ratio (Feldman et al., 2017). It should also be noted that as is the greedy algorithm fails to provide any constant factor approximation guarantee when the submodular function is non-monotone, and thus, the above modifications of it are necessary.

In the streaming setting, when the elements arrive one at a time and the memory footprint is not allowed to grow significantly with the size of the data, the landscape of constrained submodular maximization is much less understood. In particular, even for the simple problem of monotone submodular maximization subject to a cardinality constraint, the best known approximation guarantee is 2 (Badanidiyuru et al., 2014) (as opposed to $e/(e-1)$ in the offline setting). Moreover, no algorithm is currently known to achieve a non-trivial guarantee for more complicated constraints such as k-extendible or k-set system in the streaming setting even when the submodular objective function is monotone.

In this paper, we propose the first streaming algorithm for maximizing a general submodular function (not necessarily monotone) subject to a general k-set system constraint. Our algorithm achieves an $O(k^2 \log k)$ approximation ratio for this problem. Moreover, when the constraint reduces to a k-extendible system the approximation guarantee of our streaming method improves to a better $O(k \log k)$ approximation ratio, nearly matching a lower bound of roughly k due to (Feldman et al., 2017) that applies even for the offline version of the problem. Interestingly, the last approximation ratio is a significant improvement even compared to the best approximation ratio previously known for the very special case of this problem in which the objective function is linear. The current state-of-the-art algorithm for this special case, due to (Crouch & Stubbs, 2014), guarantees only an $O(k^2)$-approximation.

With the exception some algorithms designed for the simple cardinality constraint (Alaluf & Feldman, 2019; Badanidiyuru et al., 2014; Ene et al., 2019; Kazemi et al., 2019), all the streaming algorithms previously suggested for submodular maximization (see (Buchbinder et al., 2014; Chekuri et al., 2015; Chakrabarti & Kale, 2015; Feldman et al., 2018)) have been based on the same basic technique. These algorithms maintain a feasible solution, and update it in the following way. When an element u arrives, the algorithm (1) determines a set of elements that have to be removed from the current feasible solution to allow u to be added without violating feasibility, and then (2) decides using some algorithm specific rule whether it is beneficial to make this trade (i.e., add u and remove the necessary elements to recover feasibility). Our algorithm uses a very different technique of maintaining multiple feasible solutions to which elements can be added (but can never be removed), which is inspired by the technique of (Crouch & Stubbs, 2014) for maximization of linear functions subject to k-set systems. Intuitively, each one of the solutions maintained by our algorithm is associated with a particular importance of elements, and the role of this solution is to collect enough elements of this importance. Since we collect elements from each level of importance, once the stream ends, the union of the solutions we maintain is a good enough summary of the stream, and our algorithm is able to pick a feasible subset of this union which is competitive with respect to the optimal solution.

One component of our algorithm is a general framework that is able to convert many streaming algorithms for monotone submodular maximization to similar algorithms for non-monotone submodular maximization. As an immediate consequence of this framework, we get a deterministic streaming algorithm for maximizing a general (not necessarily monotone) submodular function subject to a k-matchoid constraint, which is an improvement over the state-of-the-art deterministic approximation ratio for this problem due to (Chekuri et al., 2015). We also compare the empirical performance of our algorithm with the existing work and natural baselines in a set of experiments including independent set over randomly generated graphs, maximizing a linear function over edges of a graph, movie recommendation, and Yelp location data summarization. In all these applications, the various constraints are modeled as an instance of a k-extendible or a k-set system.

Before concluding this section, we need to highlight a technical issue. The standard definition of streaming algorithms requires them to use poly-logarithmic amount of space, which is less than the space necessary for keeping a solution to our problem. Thus, no algorithm for this problem aiming to produce a solution (rather than just estimate the value of the optimal solution) can be a true streaming algorithm. This is true also for all the above mentioned streaming algorithms, which are in fact semi-streaming algorithms—a semi-streaming algorithm is an algorithm that processes the data as a sequence of elements using an amount of space which is nearly linear in the maximum size of a feasible solution and typically makes only a single pass over the entire data stream. Since true streaming algorithms are almost irrelevant to our setting, we ignore the distinction between streaming and semi-streaming algorithms in this paper and often use the term "streaming algorithm" to refer to a semi-streaming algorithm.

Paper Structure. In Section 2, we review the related work. In Section 3, we formally define different types of constraints and the notation we use, and then formally state the technical results that we need. In Section 4, we describe our above mentioned framework for converting streaming algorithms for monotone submodular maximization into streaming algorithms for non-monotone submodular maximization. Then, in Section 5, we describe and formally analyze our algorithm, and in Section 6 we describe the experiments we conducted to study the empirical performance of this algorithm. **Most of the proofs for the theoretical results are deferred to the Supplementary Material.** An earlier version of this paper, in which the result applies only to non-negative linear functions subject to k-extendible constraints (Feldman & Haba, 2019), appeared on arXiv at the past under a different title.

2. Related Work

The study of submodular maximization in the streaming setting was initialized by the works of Badanidiyuru et al. (2014) and Chakrabarti & Kale (2015). As discussed above, the work of (Chakrabarti & Kale, 2015) was based on a technique allowing the removal of elements from the solution (also known as preemption). Originally, (Chakrabarti & Kale, 2015) suggested this technique only for constraints formed by the intersection of k-matroids and a monotone submodular objective function, but later works extended the use of the technique to the more general class of k-matchoid constraints as well as non-monotone submodular functions (Buchbinder et al., 2014; Chekuri et al., 2015; Feldman et al., 2018).

The above mentioned algorithm of Badanidiyuru et al. (2014) works only for the simple cardinality constraint and monotone submodular objective functions, but provides an improved approximation ratio of 2 for this setting (recently, Feldman et al. (2020) proved the optimality of this approximation factor). The technique at the heart of this algorithm is based on growing a set to which elements can only be added, which becomes the output solution of the algorithm by the end of the stream (unlike the case in the technique of (Crouch & Stubbs, 2014) on which we base our results, in which the final solution is obtained by combining multiple sets grown by the algorithm). More recent works improved the algorithm of (Badanidiyuru et al., 2014) by improving its space complexity (Kazemi et al., 2019) and extending its technique to non-monotone submodular functions (Alaluf & Feldman, 2019; Ene et al., 2019).

The study of submodular maximization in the offline (centralized) setting is very vast, and thus, we concentrate here only on results for general k-extendible or k-set system constraints. Already in 1978, Fisher et al. (1978) proved that the natural greedy algorithm obtains $k + 1$ approximation for the problem of maximizing a monotone submodular function subject to a k-set system constraint (some of their proof was given implicitly, and the details were filled in by (Călinescu et al., 2011)). This was recently proved to be almost optimal. Specifically, Badanidiyuru & Vondrák (2014) proved that no polynomial time algorithm can obtain $k - \varepsilon$ approximation for this problem for any constant $\varepsilon > 0$, and the same inapproximability result was later shown to apply also to k-extendible constraints by (Feldman et al., 2017). As mentioned in Section 1, Feldman et al. (2017) presented the state-of-the-art algorithms for maximizing a (not necessarily monotone) submodular function subject to k-set system and k-extendible constraints. Both algorithms obtain $k + o(k)$ approximation, which improves over two previous results due to (Gupta et al., 2010) and (Mirzasoleiman et al., 2016a) that obtained roughly $3k$ and $2k$ approximation, respectively, for the more general case of a k-set system constraint.

This hierarchy of independence systems (including matroids, k-matchoids, k-extendible systems, and k-set system) is quite rich and expressive. Several recent works have used these general constraints for modeling real-world applications. Badanidiyuru et al. (2020) cast text, location, and video data summarization tasks to the problem of maximizing a submodular function subject to the interaction of k-matroids and extend their results to k-matchoids. Mirzasoleiman et al. (2018) and Feldman et al. (2018) studied the video and location data summarization applications subject to k-matchoid constraints in the streaming setting. Feldman et al. (2017) used a k-extendible constraint to model a movie recommendation application. Mirzasoleiman et al. (2016a) used the intersection of k matroids and ℓ knapsacks to model several machine learning applications, including recommendation systems, image summarization, and revenue maximization tasks.

3. Preliminaries and Notation

We begin this section by presenting some notation that we use in this paper. Then, we formally define some types of constraints mentioned in Section 1, and discuss the guarantee of a simple greedy algorithm for these constraints.

Given an element u and a set A, we use $A+u$ as a shorthand for the union $A \cup \{u\}$. We also denote the marginal gain of adding u to A with respect to a set function $f\colon 2^{\mathcal{N}} \to \mathbb{R}$ using $f(u \mid A) \triangleq f(A+u) - f(A)$. Similarly, the marginal gain of adding a set $B \subseteq \mathcal{N}$ to another set $A \subseteq \mathcal{N}$ is denoted by $f(B \mid A) \triangleq f(B \cup A) - f(A)$. Note that this notation allows us, for example, to rewrite the definition of submodularity as the requirement that $f(u \mid A) \geq f(u \mid B)$ for every two sets $A \subseteq B \subseteq \mathcal{N}$ and element $u \notin B$.

A constraint is defined, for our purposes, as a pair $(\mathcal{N}, \mathcal{I})$, where $\mathcal{N}$ is a ground set and $\mathcal{I}$ is the collection of all feasible subsets of $\mathcal{N}$. All the types of constraints discussed in Section 1 are independence systems according to the following definition.

Definition 1. *Given a ground set $\mathcal{N}$ and a collection of sets $\mathcal{I} \subseteq 2^{\mathcal{N}}$, the pair $(\mathcal{N}, \mathcal{I})$ is an independence system if (i) $\varnothing \in \mathcal{I}$ and (ii) for $B \in \mathcal{I}$ and any $A \subseteq B$ we have $A \in \mathcal{I}$.*

It is customary to call a set $A \subseteq \mathcal{N}$ *independent* if it belongs to $\mathcal{I}$ and *dependent* if it does not (i.e., it is infeasible). An independent set $B \in \mathcal{I}$ which is maximal with respect to inclusion is called a *base*; that is, $B \in \mathcal{I}$ is a base if $A \in \mathcal{I}$ and $B \subseteq A$ imply that $B = A$. Furthermore, an independent set $B \in \mathcal{I}$ which is a subset of some set $E \subseteq \mathcal{N}$ is called a base of E if it is a base of the independence system $(E, 2^E \cap \mathcal{I})$. Note that this means that a set B is a base of $(\mathcal{N}, \mathcal{I})$ if and only if it is a base of $\mathcal{N}$.

With this terminology, we can now define k-set systems.

Definition 2. *An independence system $(\mathcal{N}, \mathcal{I})$ is a k-set system for an integer $k \geq 1$ if for every set $E \subseteq \mathcal{N}$, all the bases of E have the same size up to a factor of k (in other words, the ratio between the sizes of the largest and smallest bases of E is at most k).*

An immediate consequence of the definition of k-set systems is that any base of such a system is a maximum size independent set up to an approximation ratio of k. Thus, one can get a k-approximation for the problem of finding a maximum size set subject to a k-set system constraint by outputting an arbitrary base of the k-set system, which can be done using the following simple strategy. Start with the empty solution, and consider the elements of the ground set $\mathcal{N}$ in an arbitrary order. When considering an element, add it to the current solution, unless this will make the solution dependent. We refer to this procedure as the unweighted greedy algorithm.

Let us now define k-extendible systems. We remind the reader that k-extendible systems are well-known to be a restricted class of k-set systems.

Definition 3. *An independence system $(\mathcal{N}, \mathcal{I})$ is a k-extendible system for an integer $k \geq 1$ if for any two independent sets $S \subseteq T \subseteq \mathcal{N}$, and an element $u \notin T$ such that $S + u \in \mathcal{I}$, there is a subset $Y \subseteq T \setminus S$ of size at most k such that $T \setminus Y + u \in \mathcal{I}$.*

Since k-extendible systems are, in particular, k-set systems, the above discussion already implies that the unweighted greedy algorithm obtains k-approximation for the problem of finding a maximum size independent set in such a system. The following lemma strengthens this observation, and is the key technical reason that our algorithm has a better approximation guarantee for k-extendible system constraints than for k-set system constraints. The proof of the lemma can be found in Appendix A.1.

Lemma 4. *Given a k-extendible set system $(\mathcal{N}, \mathcal{I})$, the unweighted greedy algorithm is guaranteed to produce an independent set B such that $k \cdot |B \setminus A| \geq |A \setminus B|$ for any independent set $A \in \mathcal{I}$.*

4. A Framework: From Monotone to Non-Monotone Streaming Maximization

Mirzasoleiman et al. (2018) proposed a framework for the following task. Given a streaming[2] algorithm for maximizing monotone submodular functions, the framework produces a similar algorithm that works also for non-monotone submodular objectives. Unfortunately, however, this framework applies only to algorithms satisfying a property which, to the best of our knowledge, is not satisfied by any streaming algorithm from the literature (except algorithms that work for non-monotone functions by design). In particular, this is the case for the algorithm of Chekuri et al. (2015) explicitly mentioned by (Mirzasoleiman et al., 2018) as a natural fit for their framework. In the rest of this section we discuss this issue in more detail, and then introduce a different framework which achieves the same goal (converting algorithms for monotone submodular maximization into algorithms for non-monotone submodular maximization), but requires a different property from the input algorithms which is satisfied by both existing algorithms from the literature and the new algorithm we suggest in this paper.

The algorithm of (Chekuri et al., 2015) discussed above is a streaming algorithm for maximizing monotone submodular functions under a k-matchoid constraint, and Mirzasoleiman et al. (2018) applied their framework to it in order to get such an algorithm for non-monotone function. Formally, this framework requires the input streaming algorithm to satisfy the inequality

$$f(S) \geq \alpha \cdot f(S \cup T) \ , \tag{2}$$

where S as the output of the algorithm, T is an arbitrary feasible solution and α is a positive value. Unfortunately, the algorithm of (Chekuri et al., 2015) fails to satisfy Eq. (2) for any constant α, so does the algorithms of Buchbinder et al. (2019) and Chakrabarti & Kale (2015). In Appendix B we provide examples showing that this is the case for all these algorithms even under a simple cardinality constraint.

Interestingly, Chekuri et al. (2015) presented, prior to the work of (Mirzasoleiman et al., 2018), an alternative method to convert their algorithm into a deterministic algorithm for non-monotone functions based on a technique due to Gupta et al. (2010). The framework we suggest in this paper, can be viewed as a formalization and generalization of this technique. As an alternative to the property (2), which does not have counterparts in most of the existing streaming algorithms, our framework uses the property described by Definition 5.

[2] Recall that in this paper we use the term "streaming algorithm" to refer to algorithms that are technically "semi-streaming algorithms", i.e., their space complexity is allowed to be nearly-linear in the size of the output set.

Definition 5. *Consider a data stream algorithm[3] for maximizing a non-negative submodular function $f\colon 2^{\mathcal{N}} \to \mathbb{R}_{\geq 0}$ subject to a constraint $(\mathcal{N}, \mathcal{I})$. We say that such an algorithm is an (α, γ)-approximation algorithm, for some $\alpha \geq 1$ and $\gamma \geq 0$, if it returns two sets $S \subseteq A \subseteq \mathcal{N}$ such that $S \in \mathcal{I}$, and for all $T \in \mathcal{I}$ we have*

$$\mathbb{E}[f(T \cup A)] \leq \alpha \cdot \mathbb{E}[f(S)] + \gamma \enspace .$$

Intuitively, the set S in Definition 5 is the output of the streaming algorithms, and the set A is the set of "bad" elements in the sense that the approximation guarantee of the algorithm is with respect to $f(OPT \cup A)$ rather than $f(OPT)$ (for non-monotone functions $f(OPT \cup A)$ might be smaller than $f(OPT)$). Many previous algorithms satisfy Definition 5 even with $\gamma = 0$, when the set A is the set of all elements that are "seriously" considered by the algorithm at some point. Unfortunately, however, this set A can be quite large, and therefore, cannot be stored by the algorithm if we want it to remain a streaming algorithm. Chekuri et al. (2015) described a technique to bypass this issue by creating a tradeoff between the size of A and the value of γ. They do that by guessing the value of the optimal solution, and discarding elements whose marginal is very small compared to the guess. This shrinks the size of the elements that are "seriously" considered, and thus, the size of the set A, but requires a positive value for γ representing the value of elements of OPT that are discarded. By setting the parameters right, it is possible to keep both γ and $|A|$ reasonably small. The same technique can be used to get a similar result for the algorithms of (Buchbinder et al., 2019; Chakrabarti & Kale, 2015) as well. More generally, as far as we know, similar modifications can be used to make every currently existing streaming algorithm for submodular maximization fit Definition 5, and thus, our framework is applicable to all these algorithms.

We are now ready to describe the algorithm at the heart of our framework. This algorithm is given as Algorithm 1, and it assumes access to two procedures: (1) a data stream algorithm StreamingAlg for the problem of maximizing a non-negative submodular function $f\colon 2^{\mathcal{N}} \to \mathbb{R}_{\geq 0}$ subject to a constraint $(\mathcal{N}, \mathcal{I})$, and (2) an offline algorithm ConstrainedAlg for the same problem. For the data stream algorithm StreamingAlg we use the following, not very standard, semantics. Algorithm 1 has two ways to call StreamingAlg. In the first way, every time that Algorithm 1 would like to pass additional elements to StreamingAlg, it calls it with the set of these new elements, and StreamingAlg updates its internal data structures accordingly and returns a set including all the

[3]We remind the reader that a data stream algorithm is any algorithm that receives its input in the form of a stream. A (semi-)streaming algorithm is a data stream algorithm whose space complexity is nearly linear.

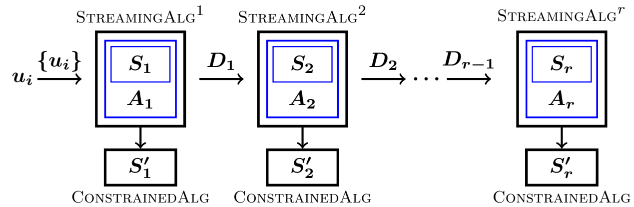


Figure 1. Schematic representation of Algorithm 1.

Algorithm 1: Non-monotone Data Stream Algorithm

Input: a positive integer r
Output: a set $S \in \mathcal{I}$
Initialize r independent copies of StreamingAlg: StreamingAlg$^{(1)}, \ldots,$ StreamingAlg$^{(r)}$.
while *there are more elements in the stream* **do**
  Let D_0 be a singleton set containing the next element of the stream.
  for $i = 1$ **to** r **do**
    $D_i \leftarrow$ StreamingAlg$^{(i)}(D_{i-1})$.
Let $D_0 \leftarrow \varnothing$.
for $i = 1$ **to** r **do**
  $[S_i, A_i, D_i] \leftarrow$ StreamingAlg$^{(i)}_{\mathsf{end}}(D_{i-1})$.
Let $S' \leftarrow$ ConstrainedAlg(A_i).
return *the set maximizing f among S' and $\{S_i\}_{i=1}^r$.*

elements that it decided to remove from its memory. In the second way, once the stream ends, Algorithm 1 calls StreamingAlg (denoted by the subscript end) and passes to it any final elements it would like StreamingAlg to get. StreamingAlg then process these elements and returns three sets: the sets S and A produced by StreamingAlg (as described in Definition 5) and a set D consisting of all the elements that are still in the memory of StreamingAlg and did not end up in A. Figure 1 is a graphic representation of the flow of elements within Algorithm 1.

It is clear that Algorithm 1 outputs a feasible solution. The following theorem gives additional properties of this algorithm. Its proof can be found in Appendix A.2.

Theorem 6. *Given an (α, γ)-approximation data stream algorithm* StreamingAlg *for maximizing a non-negative submodular function subject to some constraint and an offline β-approximation algorithm* ConstrainedAlg *for the same problem. There exists a data stream algorithm returning a feasible set S that obeys*

$$\mathbb{E}[f(S)] \geq \frac{(1 - 1/r) \cdot \texttt{OPT} - \gamma}{\alpha + \beta} \enspace .$$

Furthermore,

- *this algorithm is deterministic if* StreamingAlg *and*

CONSTRAINEDALG *are both deterministic.*

- *the space complexity of this algorithm is upper bounded by $O(r \cdot M_{\text{STREAMINGALG}} + M_{\text{CONSTRAINEDALG}})$, where $M_{\text{STREAMINGALG}}$ and $M_{\text{CONSTRAINEDALG}}$ represent the space complexities of their matching algorithms under the assumption that the input for* STREAMINGALG *is a subset of the full input and the input for* CONSTRAINEDALG *is the set A produced by* STREAMINGALG *on some such subset.*

We note that the algorithm guaranteed by Theorem 6 is a streaming algorithm when STREAMINGALG is a streaming algorithm, the algorithm CONSTRAINEDALG is a nearly-linear space algorithm and r is upper bounded by a poly-log function (note that the sets S_i and A_i are produced by STREAMINGALG, and thus, their space complexity is already accounted for by the space complexity of STREAMINGALG).

In Appendix C we show that by plugging one of the versions of the algorithm of Chekuri et al. (2015) into our framework it is straightforward to get a deterministic streaming algorithm for the problem of maximizing a non-negative (not necessarily monotone) submodular function subject to a k-matchoid constraint whose approximation ratio is $(5 + 15\varepsilon)k + O(\sqrt{k})$ for every constant $\varepsilon > 0$, which is an improvement over the guarantee of the previous state-of-the-art deterministic streaming algorithm for this problem (also due to (Chekuri et al., 2015)) which has an approximation guarantee of $8k + \gamma$, where γ is the best offline approximation ratio for the same problem.

5. Streaming Submodular Maximization under a k-System Constraint

In this section, we formally prove our results for the problem of maximizing a non-negative submodular function $f\colon 2^{\mathcal{N}} \to \mathbb{R}_{\geq 0}$ subject to a k-set system and k-extendible system constraint $(\mathcal{N}, \mathcal{I})$. A simple version of the algorithm we use to prove these results is given as Algorithm 2. This version assumes pre-access to a value ρ equal to the size of the largest independent set in $\mathcal{I}$ and a threshold τ estimating the value $M = \max_{u \in \mathcal{N}, \{u\} \in \mathcal{I}} f(\{u\})$. In Appendix D, we present a more involved version of our algorithm that does not need this pre-access, but has a space complexity larger than that of Algorithm 2 by a factor of $O(\log \rho + \log k)$—the approximation guarantee remains unchanged.

Intuitively, Algorithm 2 maintains ℓ independent sets E_i, where each one of these sets corresponds to a different range of marginal contributions: the larger i, the smaller the marginal contributions E_i is associated with. When an element u arrives, the algorithm calculates the marginal contribution $m(u)$ of u with respect to the union of the E_i sets,[4] and then adds u to the E_i corresponding to this marginal contribution, unless this violates the independence of this E_i. Once the entire input has been processed, the algorithm combines the E_i sets into h possible output sets $T_0, T_1, \ldots, T_{h-1}$. Each output sets T_j is constructed by greedily taking elements from the E_i sets obeying $i \equiv j \pmod h$ (the algorithm scans the sets obeying this condition in an increasing i order, which is a decreasing order with respect to the marginal contributions associated with these sets). The final output of the algorithm is simply the best set among the sets $T_0, T_1, \ldots, T_{h-1}$.

Algorithm 2: Streaming Algorithm for k-set Systems

Input: a threshold $\tau \in [M, 2M]$, the size ρ of the largest independent set, and the parameter k of the constraint. `// ` $M = \max_{u \in \mathcal{N}, \{u\} \in \mathcal{I}} f(\{u\})$.
Output: a solution $T \in \mathcal{I}$
Let $\ell \leftarrow \lfloor \log_2(4\rho) \rfloor$ and $h \leftarrow \lceil \log_2(2k+1) \rceil$.
for $i = 0$ ***to*** ℓ **do** Initialize $E_i \leftarrow \varnothing$.
for *every element u arriving* **do**
  `/*` Adds u to a set $E_{i(u)}$ based on its marginal gain. `*/`
  Let $m(u) \leftarrow f\left(u \mid \cup_{i=0}^{\ell} E_i\right)$.
  if $m(u) > 0$ **then** Let $i(u) \leftarrow \lfloor \log_2(\tau / m(u)) \rfloor$ **else** Let $i(u) \leftarrow \infty$.
  if $0 \leq i(u) \leq \ell$ *and* $E_{i(u)} + u \in \mathcal{I}$ **then** Update $E_{i(u)} \leftarrow E_{i(u)} + u$.
for $j = 0$ ***to*** $h - 1$ **do**
  Let $i \leftarrow j$ and $T_j \leftarrow \varnothing$.
  while $i \leq \ell$ **do**
    while *there is an element $u \in E_i$ such that* $T_j + u \in \mathcal{I}$ **do** Update $T_j \leftarrow T_j + u$.
    $i \leftarrow i + h$. `// Greedily generates solution` T_j `from sets` E_i `for` $i \equiv j \pmod h$.
return *the set T maximizing function f among sets* $T_0, T_1, \cdots, T_{h-1}$.

Let's denote $E = \cup_{i=0}^{\ell} E_i$. It is not difficult to argue that the value of $f(E)$ is large. In Lemma 7, we show that the value of the output set of the algorithm is proportional to $f(E)$ subject to a k-set system constraint.[5]

Lemma 7. *If $(\mathcal{N}, \mathcal{I})$ is a k-set system, then Algorithm 2 returns a set T such that*

$$f(T) \geq \frac{f(E \cup U) - \tau/4}{4kh(2k+1)} = \frac{f(E \cup U) - \tau/4}{O(k^2 \log k)}$$

[4]The notation $m(u)$ might suggest that the value $m(u)$ depends only on the identity of the element u. However, this is not the case. In fact, $m(u)$ might depend also on the set of elements that arrived before u and the order of their arrival.

[5]In the supplementary material, we provide a stronger version of this result for k-extendible constraints.

for every set $U \in \mathcal{I}$.

Following (see Lemma 8) is the key result that we use to prove Lemma 7. This lemma relates the value of $f(T_j)$ to the sum of the $m(u)$ values of the elements u that belong to the sets E_i that are combined to create T_j (recall that these are exactly the sets E_i for which $i \equiv j \pmod h$). Intuitively, the lemma holds because when Algorithm 2 adds elements of a set E_i to a set T_j, this increases the size of the set T_j to at least E_i/k (since the constraint is k-set system). Thus, either about $1/k$ of the elements of E_i are added to T_j, or the size of T_j before the addition of the elements of E_i is already significant compared to the size of E_i. Moreover, in the latter case, the elements of T_j can pay for the elements of E_i that they have blocked because they all have a relatively high value (as they originate in a set $E_{i'}$ for some $i' \leq i - h$).

Lemma 8. *If* $(\mathcal{N}, \mathcal{I})$ *is a* k*-set system, then for every integer* $0 \leq j < h$ *we have*

$$f(T_j \mid \varnothing) \geq \frac{1}{4k} \cdot \sum_{\substack{0 \leq i \leq \ell \\ i \equiv j \pmod h}} \sum_{u \in E_i} m(u) \enspace .$$

From the result of Lemma 7, we can directly guarantee the performance of our algorithm for monotone submodular functions. Furthermore, Lemma 7 implies that Algorithm 2 is a $(4kh(2k + 1), \tau/4)$-approximation data stream algorithm subject to a k-set system constraint.[6] Therefore, we can use the framework of Algorithm 1 to make this algorithm suitable for non-monotone submodular functions. The following two theorems describe the guarantees we provide for Algorithm 2, and their complete proofs can be found in Appendix A.3.

Theorem 9. *There is a streaming* $O(k^2 \log k) = \tilde{O}(k^2)$*-approximation algorithm for maximizing a non-negative submodular function subject to a* k*-set system constraint.*

Theorem 10. *There is a streaming* $O(k \log k) = \tilde{O}(k)$*-approximation algorithm for maximizing a non-negative submodular function subject to a* k*-extendible system constraint.*

6. Experiments

In this section, we compare our proposed algorithms (both monotone and non-monotone versions) with two other groups of algorithms: other streaming algorithms and state-of-the-art *offline* algorithms.

We consider three baseline streaming algorithms: i) The **Streaming-Greedy** algorithm: this algorithm keeps a solution S which is initially set to the empty set. For every incoming element u, it is added to the set S if this does not violate feasibility (i.e., $S \cup \{u\} \in \mathcal{I}$). ii) The **Preemption** algorithm: Inspired by the streaming algorithms of (Chekuri et al., 2015; Buchbinder et al., 2019), we consider a heuristic preemptive algorithm. For every incoming element u, given that S is the current solution, this algorithm generates a set $U \subseteq S$ such that $(S \cup \{u\}) \setminus U$ is feasible under the non-knapsack constraints. The element u is then added to the solution in exchange for the elements of U if this does not violate the knapsack constraints and the exchange is beneficial in the sense that $f(u \mid S) \geq \sum_{u' \in U} f(u' : S)$, where $f(u' : S) = f(u' \mid S')$ for $S' = \{s \in S : \text{element } s \text{ arrived before } u'\}$. For more detail refer to (Chekuri et al., 2015; Feldman et al., 2018). iii) The **Sieve-Streaming** algorithm: this heuristic algorithm is implemented based on the ideas of (Badanidiyuru et al., 2014). In the first step, it finds an accurate estimation of $\texttt{OPT}$. Then, each incoming element u is added to the solution S if $S \cup \{u\} \in \mathcal{I}$ and $f(u \mid S) \geq \texttt{OPT}/(2\rho)$, where ρ is the maximum cardinality of a feasible solution.

For the offline algorithms we consider 1) the vanilla greedy algorithm (referred to as **Greedy**), 2) **Fast** (Badanidiyuru & Vondrák, 2014) and **FANTOM** (Mirzasoleiman et al., 2016a). Both Fast and FANTOM are designed to maximize submodular functions under a k-set system constraint combined with ℓ knapsack constraints.

Sections 6.1 and 6.2 compare the above algorithms on tasks of maximizing linear and cut objective functions over instances produced using synthetic and real-world data, respectively. Then, in Section 6.3 and Appendices E.4 and E.5, we evaluate the performance of the same algorithms on three different real-world applications. In a movie recommendation system application, we are given movie ratings from users, and our goal is to recommend diverse movies from different genres. In a Yelp location summarization application, we are given thousands of business locations with several related attributes. Our objective is to find a good summary of the locations from six different cities. In a third application, our goal is to generate real-time summaries for Twitter feeds of several news agencies.

6.1. Independent set

In the experiments of this section we define submodular functions over the nodes of a given graph $G = (V, E)$, and consider the maximization of such functions subject to an independent set constraint, i.e., we are not allowed to select a set of vertices if there is any edge of the graph connecting any two of these vertices. It is easy to show that this constraint is a $d_{\max}$-extendible system, where $d_{\max}$ is the maximum degree in graph G. In our experiments in this section, we use two types of synthetically generated random graphs: Erdős Rény graphs (Erdős & Rény,

[6] In the supplementary material, we prove that Algorithm 2 is a $(4h(2k + 1), \tau/4)$-approximation data stream algorithm subject to a k-extendible constraint.

1960) and Watts–Strogatz graphs (Watts & Strogatz, 1998). For the Erdős Rény graphs we vary in our experiments the probability p that each possible edge is included in the graph (independently from every other edge), and for the Watts-Strogatz graphs we vary the rewiring probability β. The number of nodes is set to $n = 2000$ in all the graphs, and in the Watts–Strogatz model each node is connected to $k = 100$ nearest neighbors in the ring structure.

In our first experiment of the section, we study the maximization of the following monotone linear function $f(S) = \sum_{u \in S} w_u$, where $S \subseteq V$ and w_u is the weight of node $u \in S$. Figs. 2a and 2b compare different algorithms for optimizing this function over a random graph chosen from the above discussed random graph models. We observe that our proposed algorithm consistently outperforms the other baseline streaming algorithms. We also observe that the performance our algorithm is comparable with (or even better at times) the greedy algorithm, which is provably optimal for the maximization of linear functions subject to k-extendible constraints (Feldman et al., 2017).

In the second experiment, we study the non-monotone submodular graph-cut function: $f(S) = \sum_{u \in S} \sum_{v \in V \setminus S} w_{u,v}$, where $w_{u,v}$ is the weight of the edge $e = (u, v)$. Again, in Figs. 2c and 2d we observe that the solutions provided by our algorithms are clearly better than those produced by the other streaming algorithms. Furthermore, the non-monotone version of our algorithm always outperforms the monotone algorithm. We also note that the for this non-monotone submodular function, the vanilla greedy algorithm performs very poorly (which is consistent with the lack of a theoretical guarantee for this algorithm for such functions).

6.2. Graph Planarity with Knapsack

In the experiment of this section, our objective is to maximize a linear function over the edges of a graph $G = (V, E)$. For the constraint, we require that an independent set of edges corresponds to a planar sub-graph of G, and in addition, it satisfies a given knapsack constraint. We remind the reader that a graph is planar if it can be embedded in the plane. Furthermore, a knapsack constraint is defined by a cost function $c\colon \mathcal{N} \to \mathbb{R}_{\geq 0}$, and we say that a set $S \subseteq \mathcal{N}$ satisfies the knapsack constraint if $c(S) = \sum_{e \in S} c(e) \leq b$ for a given knapsack budget b. In Appendix E.1 we explain why the above mentioned constraint is k-set system for a (relatively) modest value of k.[7]

For the objective function f, we use the monotone linear function: $f(S) = \sum_{e \in S} w_e \quad \forall S \subseteq E$, where w_e is the weight of edge $e \in S$, and for simplicity, we set all these weights to 1. The knapsack cost of each edge $e = (u, v) \in E$ is chosen to be proportional to $\max(1, d_u - q)$, where d_u is the degree of node u in graph G and $q = 6$, and the costs are normalized so that $\sum_{e \in E} c_e = |V|$, where c_e represents the knapsack cost of edge e.

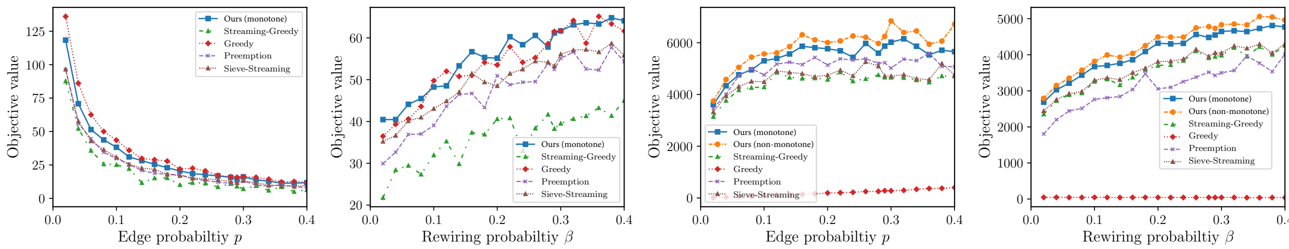
In the experiment, we use two real-world networks from (Leskovec & Krevl, 2014) as the graph, and vary the knapsack budget between 0 and 1 (note that the normalization gives this range of budgets an intuitive meaning). In Figs. 3a and 3b we compare the performance of our streaming algorithm with the performance of Streaming Greedy and Sieve Streaming. One can observe that our algorithm outperforms the two other baselines. Due to the prohibitive computational complexity of the offline algorithms, we do not report their results for this experiment. Furthermore, as it is not clear how to execute a preemptive streaming algorithm under a planarity constraint, we did not include a version of the Preemption algorithm in this experiment. Further experiments in this setting can be found in Appendix E.2.

6.3. Movie Recommendation

In the movie recommendation application, our goal is to select a diverse set of movies subject to constraints that can be adjusted by the user. The dataset for this experiment contains 1793 movies from the genres: Adventure, Animation, and Fantasy (note that a single movie may be identified with multiple genres). The user may specify an upper limit m on the number of movies in the set we recommend for them, as well as an upper limit m_i on the number of movies from each genre. For simplicity, we use a single value for all m_i and refer to this value as the genre limit. It is easy to show that this set of constraints forms a 3-extendible system. In addition, we enforce two knapsack constraints. For the first knapsack constraint c_1, the cost of each movie is proportional to the absolute difference between the release year of the movie and the year 1985 (the implicit goal of this constraint is to pick movies with a release year which is as close as possible to the year 1985). For the second knapsack constraint c_2, the cost of each movie is proportional to the difference between the maximum possible rating (which is 10) and the rating of the particular movie—here the goal is to pick movies with higher ratings. More formally, for a movie $v \in \mathcal{N}$, we have: $c_1(v) \propto |1985 - \text{year}_v|$ and $c_2(v) \propto (10 - \text{rating}_v)$. Here, year_v and rating_v, respectively, denote the release year and IMDb rating of movie v. We normalize the costs in both knapsacks constraints so that the average cost of each movie is $1/10$, i.e., $\frac{\sum_{v \in \mathcal{N}} c_i(V)}{|\mathcal{N}|} = 1/10$, and we set the knapsack budgets to 1; which intuitively means that a feasible set can contain no more than about 10% of the movies.

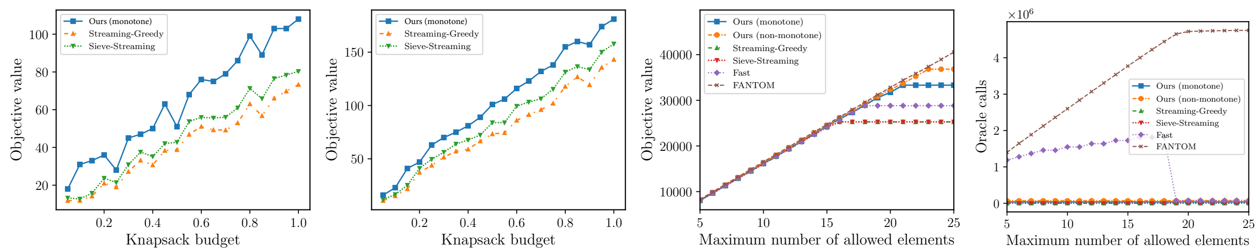
In our experiments, we try to maximize two kinds of objective functions (each trying to capture diversity in a different way) subject to these constraints, and we vary the upper limit m on the number of movies in the recommended

[7] We would like to thank Chandra Chekuri for pointing out some parts of this explanation to us.



(a) Erdős Rény (linear) (b) Watts–Strogatz (linear) (c) Erdős Rény (cut) (d) Watts–Strogatz (cut)

Figure 2. For Erdős Rény graphs p is the probability of having an edge between any two nodes. For Watts–Strogatz graphs β is the probability of rewiring of each edge.



(a) EU Email (b) Facebook ego network (c) Non-monotone function (d) Non-monotone function

Figure 3. (a) and (b): Planarity with knapsack (linear objective function). (c) and (d): Movie recommendation with two knapsacks.

set of movies. Both objective functions are based a set of attributes calculated for each movie using the method described in (Lindgren et al., 2015), and both objective functions are non-negative and submodular. However, one of them is monotone, and the other is not (guaranteed to be) monotone. The experimental result for the monotone function is given in Appendix E.3.

An intuitive utility function for choosing a diverse set of movies S is the following not necessarily monotone submodular function

$$f(S) = \sum_{i \in S} \sum_{j \in \mathcal{N}} M_{i,j} - \sum_{i \in S} \sum_{j \in S} M_{i,j} \quad , \tag{3}$$

where $\mathcal{N}$ is the set of all movies and $M_{i,j}$ is the non-negative similarity score between movies $i, j \in \mathcal{N}$ as defined in the previous section. It is beneficial to note that the first term is a sum-coverage function that captures the representativeness of the selected set, and the second term is a dispersion function penalizing similarity within S (Feldman et al., 2017).

In our experiment with this function as the object, we set the genre limit to 20. In Figs. 3c and 3d, we observe that i) our streaming algorithm returns solutions with higher utilities compared to the baseline streaming algorithms, ii) the non-monotone version of our algorithm clearly outperforms the monotone one for this non-monotone function, and iii) the quality of the solutions returned by our algorithms is comparable with the quality obtained by offline algorithms.

7. Conclusion

In this paper, we have proposed a novel framework for converting streaming algorithms for monotone submodular maximization into streaming algorithms for non-monotone submodular maximization, which immediately led us to the currently tightest deterministic approximation ratio for submodular maximization subject to a k-matchoid constraint. We also proposed the first streaming algorithm for monotone submodular maximization subject to k-extendible and k-set system constraints, which (together with our proposed framework), yields approximation ratios of $O(k \log k)$ and $O(k^2 \log k)$ for maximization of general non-negative submodular functions subject to the above constraints, respectively. Finally, we extensively evaluated the empirical performance of our algorithm against the existing work in a series of experiments including finding the maximum independent set in randomly generated graphs, maximizing linear functions over social networks, movie recommendation, Yelp location summarization, and Twitter data summarization.

Acknowledgements

The research of Moran Feldman and Ran Haba was partially supported by ISF grant 1357/16. Amin Karbasi is partially supported by NSF (IIS- 1845032), ONR (N00014-19-1-2406), and AFOSR (FA9550-18-1-0160).

References

Alaluf, N. and Feldman, M. Making a sieve random: Improved semi-streaming algorithm for submodular maximization under a cardinality constraint. *CoRR*, abs/1906.11237, 2019.

Badanidiyuru, A. and Vondrák, J. Fast algorithms for maximizing submodular functions. In *ACM-SIAM symposium on Discrete algorithms (SODA)*, pp. 1497–1514, 2014.

Badanidiyuru, A., Mirzasoleiman, B., Karbasi, A., and Krause, A. Streaming Submodular Maximization:Massive Data Summarization on the Fly. In *International Conference on Knowledge Discovery and Data Mining, KDD*, pp. 671–680, 2014.

Badanidiyuru, A., Karbasi, A., Kazemi, E., and Vondrák, J. Submodular maximization through barrier functions. *arXiv preprint arXiv:2002.03523*, 2020.

Buchbinder, N., Feldman, M., Naor, J., and Schwartz, R. Submodular Maximization with Cardinality Constraints. In *SODA*, pp. 1433–1452, 2014.

Buchbinder, N., Feldman, M., and Schwartz, R. Online Submodular Maximization with Preemption. *ACM Trans. Algorithms*, 15(3):30:1–30:31, 2019.

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

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

Chekuri, C., Gupta, S., and Quanrud, K. Streaming algorithms for submodular function maximization. In *ICALP*, pp. 318–330, 2015.

Crouch, M. and Stubbs, D. M. Improved streaming algorithms for weighted matching, via unweighted matching. In *APPROX*, pp. 96–104, 2014.

Das, A. and Kempe, D. Submodular meets spectral: greedy algorithms for subset selection, sparse approximation and dictionary selection. In *International Conference on International Conference on Machine Learning*, pp. 1057–1064, 2011.

Dasgupta, A., Kumar, R., and Ravi, S. Summarization through submodularity and dispersion. In *Annual Meeting of the Association for Computational Linguistics*, pp. 1014–1022, 2013.

De, A., Koley, P., Ganguly, N., and Gomez-Rodriguez, M. Regression Under Human Assistance. *CoRR*, abs/1909.02963, 2019.

Elenberg, E. R., Dimakis, A. G., Feldman, M., and Karbasi, A. Streaming Weak Submodularity: Interpreting Neural Networks on the Fly. In *Advances in Neural Information Processing Systems*, pp. 4047–4057, 2017.

Elhamifar, E. and Clara De Paolis Kaluza, M. Online summarization via submodular and convex optimization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1783–1791, 2017.

Ene, A., Nguyen, H. L., and Suh, A. An optimal streaming algorithm for non-monotone submodular maximization. *CoRR*, abs/1911.12959, 2019.

Erdős, P. and Rény, A. On the Evolution of Random Graphs. *Publ. Math. Inst. Hungary. Acad. Sci.*, 5:17–61, 1960.

Feldman, M. and Haba, R. Almost Optimal Semi-streaming Maximization for k-Extendible Systems. *arXiv preprint arXiv:1906.04449*, 2019.

Feldman, M., Harshaw, C., and Karbasi, A. Greed is good: Near-optimal submodular maximization via greedy optimization. In *COLT*, pp. 758–784, 2017.

Feldman, M., Karbasi, A., and Kazemi, E. Do less, get more: Streaming submodular maximization with subsampling. In *NeurIPS*, pp. 730–740, 2018.

Feldman, M., Norouzi-Fard, A., Svensson, O., and Zenklusen, R. The one-way communication complexity of submodular maximization with applications to streaming and robustness. In *Symposium on Theory of Computing (STOC)*, pp. 1363–1374, 2020.

Fisher, M., Nemhauser, G., and Wolsey, L. An analysis of approximations for maximizing submodular set functions–II. *Mathematical Programming*, 8:73–87, 1978.

Frieze, A. M. A cost function property for plant location problems. *Mathematical Programming*, 7(1):245–248, 1974.

Gupta, A., Roth, A., Schoenebeck, G., and Talwar, K. Constrained Non-monotone Submodular Maximization: Offline and Secretary Algorithms. In *WINE*, pp. 246–257, 2010.

Herbrich, R., Lawrence, N. D., and Seeger, M. Fast sparse Gaussian process methods: The informative vector machine. In *Advances in Neural Information Processing Systems*, pp. 625–632, 2003.

Kazemi, E., Zadimoghaddam, M., and Karbasi, A. Scalable Deletion-Robust Submodular Maximization: Data Summarization with Privacy and Fairness Constraints. In *International Conference on Machine Learning (ICML)*, pp. 2549–2558, 2018.

Kazemi, E., Mitrovic, M., Zadimoghaddam, M., Lattanzi, S., and Karbasi, A. Submodular Streaming in All Its Glory: Tight Approximation, Minimum Memory and Low Adaptive Complexity. In *International Conference on Machine Learning (ICML)*, pp. 3311–3320, 2019.

Kazemi, E., Minaee, S., Feldman, M., and Karbasi, A. Regularized submodular maximization at scale. *arXiv preprint arXiv:2002.03503*, 2020.

Kirchhoff, K. and Bilmes, J. Submodularity for data selection in statistical machine translation. In *Proceedings of EMNLP*, 2014.

Krause, A. and Golovin, D. Submodular Function Maximization. In *Tractability: Practical Approaches to Hard Problems*. Cambridge University Press, 2012.

Lei, Q., Wu, L., Chen, P.-Y., Dimakis, A., Dhillon, I., and Witbrock, M. Discrete Adversarial Attacks and Submodular Optimization with Applications to Text Classification. *Systems and Machine Learning (SysML)*, 2019.

Leskovec, J. and Krevl, A. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data, June 2014.

Libbrecht, M. W., Bilmes, J. A., and Noble, W. S. Choosing non-redundant representative subsets of protein sequence data sets using submodular optimization. *Proteins: Structure, Function, and Bioinformatics*, 2018. ISSN 1097-0134.

Lindgren, E. M., Wu, S., and Dimakis, A. G. Sparse and greedy: Sparsifying submodular facility location problems. In *NIPS Workshop on Optimization for Machine Learning*, 2015.

Mirzasoleiman, B., Karbasi, A., Sarkar, R., and Krause, A. Distributed Submodular Maximization: Identifying Representative Elements in Massive Data. In *Advances in Neural Information Processing Systems*, pp. 2049–2057, 2013.

Mirzasoleiman, B., Badanidiyuru, A., and Karbasi, A. Fast Constrained Submodular Maximization: Personalized Data Summarization. In *ICML*, pp. 1358–1367, 2016a.

Mirzasoleiman, B., Karbasi, A., Sarkar, R., and Krause, A. Distributed Submodular Maximization. *Journal of Machine Learning Research*, 17:238:1–238:44, 2016b.

Mirzasoleiman, B., Jegelka, S., and Krause, A. Streaming Non-Monotone Submodular Maximization: Personalized Video Summarization on the Fly. In *AAAI Conference on Artificial Intelligence,*, pp. 1379–1386, 2018.

Mitrovic, M., Kazemi, E., Zadimoghaddam, M., and Karbasi, A. Data Summarization at Scale: A Two-Stage Submodular Approach. In *International Conference on Machine Learning (ICML)*, pp. 3593–3602, 2018.

Mitrovic, M., Kazemi, E., Feldman, M., Krause, A., and Karbasi, A. Adaptive sequence submodularity. In *Advances in Neural Information Processing Systems*, pp. 5352–5363, 2019.

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

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

Norouzi-Fard, A., Tarnawski, J., Mitrovic, S., Zandieh, A., Mousavifar, A., and Svensson, O. Beyond 1/2-approximation for submodular maximization on massive data streams. In *International Conference on Machine Learning (ICML)*, pp. 3826–3835, 2018.

Salehi, M., Karbasi, A., Scheinost, D., and Constable, R. T. A Submodular Approach to Create Individualized Parcellations of the Human Brain. In *MICCAI*, pp. 478–485, 2017.

Singla, A., Bogunovic, I., Bartók, G., Karbasi, A., and Krause, A. Near-Optimally Teaching the Crowd to Classify. In *International Conference on Machine Learning (ICML)*, 2014.

Tschiatschek, S., Iyer, R. K., Wei, H., and Bilmes, J. A. Learning mixtures of submodular functions for image collection summarization. In *Advances in neural information processing systems*, pp. 1413–1421, 2014.

Tschiatschek, S., Singla, A., and Krause, A. Selecting sequences of items via submodular maximization. In *AAAI Conference on Artificial Intelligence*, 2017.

Watts, D. J. and Strogatz, S. H. Collective dynamics of 'small-world'networks. *nature*, 393(6684):440, 1998.

Yelp. Yelp Academic Dataset. https://www.kaggle.com/yelp-dataset/yelp-dataset, 2019a.

Yelp. Yelp Dataset. https://www.yelp.com/dataset, 2019b.

A. Proofs

A.1. Proof of Lemma 4

In this section, we restate Lemma 4 and then prove it.

Lemma 4. *Given a k-extendible set system $(\mathcal{N}, \mathcal{I})$, the unweighted greedy algorithm is guaranteed to produce an independent set B such that $k \cdot |B \setminus A| \geq |A \setminus B|$ for any independent set $A \in \mathcal{I}$.*

Proof. Let us denote the elements of $B \setminus A$ by $x_1, x_2, \ldots, x_m$ in an arbitrary order. Using these elements, we recursively define a series of independent sets $A_0, A_1, \ldots, A_m$. The set A_0 is simply the set A. For $1 \leq i \leq m$, we define A_i using A_{i-1} as follows. Since $(\mathcal{N}, \mathcal{I})$ is a k-extendible system and the subsets A_{i-1} and $A_{i-1} \cap B + x_i \subseteq B$ are both independent, there must exist a subset $Y_i \subseteq A_{i-1} \setminus (A_{i-1} \cap B) = A_{i-1} \setminus B$ such that $|Y_i| \leq k$ and $A_{i-1} \setminus Y_i + x_i \in \mathcal{I}$. Using the subset Y_i, we now define $A_i = A_{i-1} \setminus Y_i + x_i$. Note that by the definition of Y_i, $A_i \in \mathcal{I}$ as promised. Furthermore, since $Y_i \cap B = \varnothing$ for each $0 \leq i \leq m$, we know that $(A \cup \{x_1, x_2, \ldots, x_m\}) \cap B \subseteq A_m$, which implies $B \subseteq A_m$ because $\{x_1, x_2, \ldots, x_m\} = B \setminus A$. However, B, as the output of the unweighted greedy algorithm, must be an inclusion-wise maximal independent set (i.e., a base), and thus, it must be in fact equal to the independent set A_m containing it.

Let us now denote $Y = \bigcup_{i=1}^{m} Y_i$, and consider two different ways to bound the number of elements in Y. On the one hand, since every set Y_i includes up to k elements, we get $|Y| \leq km = k \cdot |B \setminus A|$. On the other hand, the fact that $B = A_m$ implies that every element of $A \setminus B$ belongs to Y_i for some value of i, and therefore, $|Y| \geq |A \setminus B|$. The lemma now follows by combining these two bounds. □

A.2. Proof of Theorem 6

In this section we prove Theorem 6. For convenience, we begin by restating the theorem itself.

Theorem 6. *Given an (α, γ)-approximation data stream algorithm* STREAMINGALG *for maximizing a non-negative submodular function subject to some constraint and an offline β-approximation algorithm* CONSTRAINEDALG *for the same problem. There exists a data stream algorithm returning a feasible set S that obeys*

$$\mathbb{E}[f(S)] \geq \frac{(1 - 1/r) \cdot \texttt{OPT} - \gamma}{\alpha + \beta} \enspace .$$

Furthermore,

- *this algorithm is deterministic if* STREAMINGALG *and* CONSTRAINEDALG *are both deterministic.*
- *the space complexity of this algorithm is upper bounded by $O(r \cdot M_{\text{STREAMINGALG}} + M_{\text{CONSTRAINEDALG}})$, where $M_{\text{STREAMINGALG}}$ and $M_{\text{CONSTRAINEDALG}}$ represent the space complexities of their matching algorithms under the assumption that the input for* STREAMINGALG *is a subset of the full input and the input for* CONSTRAINEDALG *is the set A produced by* STREAMINGALG *on some such subset.*

Recall that our goal is to show that Algorithm 1 has the properties guaranteed by the theorem. The following observation shows that this is the case for the space complexity.

Observation 11. *The space complexity of Algorithm 1 is upper bounded by $O(r \cdot M_{\text{STREAMINGALG}} + M_{\text{CONSTRAINEDALG}})$, where $M_{\text{STREAMINGALG}}$ and $M_{\text{CONSTRAINEDALG}}$ represent the space complexities of their matching algorithms under the assumption that the input for* STREAMINGALG *is a subset of the full input and the input for* CONSTRAINEDALG *is the A set produced by* STREAMINGALG *on some such subset.*

Proof. Note that every set assigned to a variable D_i by Algorithm 1 is either an input set for some copy of STREAMINGALG or an output set of such a copy. Furthermore, any such set is either ignored or fed immediately after construction to some copy of STREAMINGALG. Thus, the space complexity required for these sets is upper bounded by the space complexity required for the r copies of STREAMINGALG used by Algorithm 1, which is $O(r \cdot M_{\text{STREAMINGALG}})$. Additionally, since the sets S_i and A_i are the final outputs of these copies of STREAMINGALG, we get they can also be stored using $O(r \cdot M_{\text{STREAMINGALG}})$ space. Finally, the set S' is a subset of $\bigcup_{i=1}^{r} A_i$, and therefore, does not require more space than the sets A_i. Combining all the above, we get that the space complexity of Algorithm 1—excluding the space required for running CONSTRAINEDALG—is at most $O(r \cdot M_{\text{STREAMINGALG}})$. □

To complete the proof of Theorem 6, it remains to analyze the approximation guarantee of Algorithm 1. Towards this goal, we need the following known lemma.

Lemma 12 (Lemma 2.2 of (Buchbinder et al., 2014)). *Let $g\colon 2^{\mathcal{N}} \to \mathbb{R}_{\geq 0}$ be a non-negative submodular function, and let B be a random subset of $\mathcal{N}$ containing every element of $\mathcal{N}$ with probability at most q (not necessarily independently). Then, $\mathbb{E}[g(B)] \geq (1-q) \cdot g(\varnothing)$.*

Lemma 13. *Assume* StreamingAlg *is an (α, γ)-approximation algorithm and* ConstrainedAlg *is an offline β-approximation algorithm. Then, Algorithm 1 returns a solutions S such that*

$$\mathbb{E}[f(S)] \geq \frac{(1-1/r) \cdot \texttt{OPT} - \gamma}{\alpha + \beta} \enspace .$$

Proof. Let S^* be an arbitrary optimal solution, i.e., a set obeying $S^* \in \mathcal{I}$ and $f(S^*) = \texttt{OPT}$. For every integer $1 \leq i \leq r$, we denote by $\mathcal{N}_i$ the set of elements that StreamingAlg$^{(i)}$ has received. Note that we have $\mathcal{N}_1 = \mathcal{N}$ and $\mathcal{N}_i = \mathcal{N} \setminus (\cup_{1 \leq j \leq i-1} A_j)$ for $2 \leq i \leq r$ since StreamingAlg$^{(i)}$ outputs every element that it gets and does not end up in A_i as an element of D_i at some point. Since A_i is a subset of $\mathcal{N}_i$, this implies that the sets $A_1, A_2, \ldots, A_r$ are disjoint.

Let us define now $\bar{A}$ to be a uniformly random set from $\{A_1, A_2, \ldots, A_r\}$, and $g(S) = f(S \cup S^*)$. Then,

$$\frac{1}{r}\sum_{i=1}^{r} f(A_i \cup S^*) = \mathbb{E}_{\bar{A}}[f(\bar{A} \cup S^*)] = \mathbb{E}_{\bar{A}}[g(\bar{A})] \geq \left(1 - \frac{1}{r}\right) \cdot g(\varnothing) = \left(1 - \frac{1}{r}\right) \cdot f(S^*) \enspace ,$$

where the notation $\mathbb{E}_{\bar{A}}$ stands for expectation over the random choice of $\bar{A}$ out of $\{A_1, A_2, \ldots, A_r\}$ (but not over any randomness that might be introduced by StreamingAlg), and the inequality results from Lemma 12 because (i) every element of $\mathcal{N}$ belongs to $\bar{A}$ with probability at most $\frac{1}{r}$ since the sets A_i are disjoint, and (ii) g is a non-negative submodular function on its own right.

Let us now define $A' = \bigcup_{i=1}^{r} A_i$. Using this notation, we get from the last inequality

$$(r-1) \cdot \texttt{OPT} = (r-1) \cdot f(S^*) \leq \sum_{i=1}^{r} f(A_i \cup S^*) \leq \sum_{i=1}^{r} [f(A_i \cup (S^* \setminus A')) + f(S^* \cap A')] \enspace ,$$

where the second inequality follow from the submodularity and non-negativity of f. Observe now that $S^* \setminus A'$ is a feasible solution that StreamingAlg$^{(i)}$ can output since it is a subset of $\mathcal{N} \setminus A' \subseteq \mathcal{N}_i$ and $S^* \cap A'$ is a feasible solution that ConstrainedAlg can output. Taking now expectation over any randomness introduced by StreamingAlg and ConstrainedAlg, we get from the last inequality using the guarantees of these two algorithms that

$$\begin{aligned}(r-1) \cdot \texttt{OPT} &\leq \sum_{i=1}^{r} \{\mathbb{E}[f(A_i \cup (S^* \setminus A'))] + \mathbb{E}[f(S^* \cap A')]\} \\ &\leq \sum_{i=1}^{r} \{\alpha \cdot \mathbb{E}[f(S_i)] + \gamma + \beta \cdot \mathbb{E}[f(S')]\} \leq [r\alpha + r\beta] \cdot \mathbb{E}[f(S)] + r\gamma \enspace ,\end{aligned}$$

where the last inequality holds since S is selected as the set maximizing f among all the sets S' and $\{S_i\}_{i=1}^{r}$. □

A.3. Proofs of Theorems 9 and 10

We begin the analysis of Algorithm 2 with the following lemma showing that it is a semi-streaming algorithm.

Lemma 14. *Algorithm 2 stores $O(\rho(\log \rho + \log k)) = \tilde{O}(\rho)$ elements at every given time point.*

Proof. Algorithm 2 stores elements only in the sets $E_0, E_1, \ldots, E_\ell$ and the sets $T_0, T_1, \ldots, T_{h-1}$. Since these sets are kept independent by the algorithm, each one them contains at most ρ elements. Thus, the number of elements stored by Algorithm 2 is upper bounded by

$$(\ell + h)\rho = [O(\log \rho + \log k)]\rho = O(\rho(\log \rho + \log k)) \enspace .$$ □

Our next objective to analyze the approximation ratio of Algorithm 2. Recall that we defined for that purpose $E = \cup_{i=0}^{\ell} E_i$. The following lemma shows that $f(E)$ is large.

Lemma 15. *For every set $S \in \mathcal{I}$, $f(E \mid \varnothing) = \sum_{i=0}^{\ell} \sum_{u \in E_i} m(u) \geq \frac{f(S \cup E \mid \varnothing) - \tau/4}{2k+1}$.*

Proof. First, note that we have $f(E \mid \varnothing) = \sum_{i=0}^{\ell} \sum_{u \in E_i} m(u)$ because $m(u)$ is the marginal contribution of u with respect to the elements that were added to $\cup_{i=0}^{\ell} E_i$ before u. Let us also define, for every integer $0 \leq i \leq \ell$, $S_i = \{u \in S \mid i(u) = i\}$. Then,

$$f(E \mid \varnothing) = \sum_{i=0}^{\ell} \sum_{u \in E_i} m(u) \geq \sum_{i=0}^{\ell} |E_i| \cdot \frac{\tau}{2^{i+1}} \geq \frac{1}{k} \cdot \sum_{i=0}^{\ell} |S_i| \cdot \frac{\tau}{2^{i+1}}$$

$$\geq \frac{1}{2k} \cdot \sum_{i=0}^{\ell} \sum_{u \in S_i} m(u) = \frac{1}{2k} \cdot \left[\sum_{u \in S} m(u) - \sum_{\substack{u \in S \\ i(u) < 0 \text{ or } i(u) > \ell}} m(u) \right] ,$$

where the first and third inequalities hold since an element u is added to a set E_i only when $i = i(u)$, and the second inequality holds since one can view E_i as the output of running the unweighted greedy algorithm on a ground set which includes the independent set S_i as a subset.

By the submodularity of f, we can immediately get

$$\sum_{u \in S} m(u) \geq \sum_{u \in S} f(u \mid E) \geq f(S \mid E) = f(S \cup E \mid \varnothing) - f(E \mid \varnothing) .$$

We also note that $\{u\} \in \mathcal{I}$ for every element $u \in S$ because S itself is independent, and thus, $\tau \geq M \geq f(\{u\}) \geq m(u)$ (recall that M was defined as $\max_{u \in \mathcal{N}, \{u\} \in \mathcal{I}} f(\{u\})$). Hence, $i(u) = \lfloor \log_2(\tau/m(u)) \rfloor \geq 0$, which implies

$$\sum_{\substack{u \in S \\ i(u) < 0 \text{ or } i(u) > \ell}} m(u) = \sum_{\substack{u \in S \\ i(u) > \ell}} m(u) \leq \sum_{\substack{u \in S \\ i(u) > \ell}} \frac{\tau}{2^{\ell+1}} \leq \rho \cdot \frac{\tau}{2^{\log_2(4\rho)}} = \frac{\tau}{4} .$$

Combining all the above inequalities gives us

$$f(E \mid \varnothing) \geq \frac{1}{2k} \cdot \left[f(S \cup E \mid \varnothing) - f(E \mid \varnothing) - \frac{\tau}{4} \right] ,$$

and the lemma follows by rearranging this inequality. □

As discussed in Section 5, our next objective is to relate the value of the output set of Algorithm 2 to $f(E)$. As an intermediate step, we relate $f(T_j)$ to the sum of the $m(u)$ values of the elements u that belong to the sets E_i that are combined to create T_j (recall that these are exactly the sets E_i for which $i \equiv j \pmod{h}$). In the next lemma we assume that the constraint $(\mathcal{N}, \mathcal{I})$ is a k-set system. Naturally, the lemma holds also for constraints that are k-extendible systems, but for such constraints it is possible to get a better bound on $f(T_j)$ using a more careful analysis, and this bound appears below as Lemma 16.

Intuitively, the next lemma holds because when Algorithm 2 adds elements of a set E_i to a set T_j, this increases the size of the set T_j to at least E_i/k (since the constraint is k-set system). Thus, either about $1/k$ of the elements of E_i are added to T_j, or the size of T_j before the addition of the elements of E_i is already significant compared to the size of E_i. Moreover, in the later case, the elements of T_j can pay for the elements of E_i that they have blocked because they all have a relatively high value (as they originate in a set $E_{i'}$ for some $i' \leq i - h$). Next, we restate Lemma 8 and then prove it.

Lemma 8. *If $(\mathcal{N}, \mathcal{I})$ is a k-set system, then for every integer $0 \leq j < h$ we have*

$$f(T_j \mid \varnothing) \geq \frac{1}{4k} \cdot \sum_{\substack{0 \leq i \leq \ell \\ i \equiv j \pmod{h}}} \sum_{u \in E_i} m(u) .$$

Proof. Since T_j is a subset of E, if we denote by $v_1, v_2, \cdots, v_m$ the element of T_j in the order their arrival, then the submodularity of f guarantees that

$$f(T_j \mid \varnothing) = \sum_{r=1}^{m} f(v_r \mid v_1, v_2, \cdots, v_{r-1}) \geq \sum_{r=1}^{m} m(v_r) = \sum_{u \in T_j} m(u) \quad .$$

Thus, to prove the lemma it suffice to prove

$$\sum_{u \in T_j} m(u) \geq \frac{1}{4k} \cdot \sum_{\substack{0 \leq i \leq \ell \\ i \equiv j \pmod h}} \sum_{u \in E_i} m(u) \quad . \tag{4}$$

We prove Inequality (4) by proving a stronger claim via induction. However, before we can present this stronger claim, we need to define some additional notation. Recall that T_j is constructed by starting with the empty set, greedily adding to it elements of E_j, then greedily adding to it elements of E_{j+h}, then greedily adding to it elements of E_{j+2h} and so on. Thus, let us define, for every integer $0 \leq i \leq \ell$ obeying $i \equiv j \pmod h$, the set T_j^i to be the set T_j immediately after Algorithm 2 is done greedily adding elements of E_i to T_j. Additionally, it is useful to define T_j^{j-h} to be the empty set. Using these definitions, we can now define the stronger claim that we prove below by induction.

For every integer $-h \leq i \leq \ell$ and $j = i \bmod h$,

$$\sum_{u \in T_j^i} m(u) \geq \frac{1}{4k} \cdot \sum_{\substack{j \leq r \leq i \\ r \equiv j \pmod h}} \sum_{u \in E_r} m(u) + \frac{|T_j^i|\tau}{2^{i+2}k} \quad . \tag{5}$$

Before we prove this claim, let us observe that it indeed implies Inequality (4) by setting i to be the largest integer that obeys $i \equiv j \pmod h$ and is not larger than ℓ.

It now remains to prove Inequality (5) by induction on i. For $i < 0$, this inequality holds since $j \geq 0 > i$ and $T_j^i = \varnothing$, which implies that the value of both sides of the inequality is 0. Next, we need to prove Inequality (5) for an integer $0 \leq i \leq \ell$ under the assumption that it holds for every $-h \leq i' < i$. Recall that the set T_j^i is obtained by greedily adding elements of E_i to T_j^{i-h}. Since the constraint is a k-set system, the size of the set obtained in this way must be at least $|E_i|/k$ (otherwise, T_j^i is a base of $E_i \cup T_j^{i-h}$ whose size is smaller than the size the independent set E_i by more than a factor of k). Thus, we know that the number of elements of E_i that are added to T_j^{i-h} to form T_j^i is at least $|E_i|/k - |T_j^{i-h}|$, which implies

$$\begin{aligned}\sum_{u \in T_j^i \setminus T_j^{i-h}} m(u) &\geq \left[\frac{|E_i|}{k} - \left|T_j^{i-h}\right|\right] \cdot \frac{\tau}{2^{i+1}} \\ &= \frac{\sum_{u \in E_i} \tau/2^{i+2}}{k} + \frac{|E_i|\,\tau}{2^{i+2}k} - \frac{\left|T_j^{i-h}\right|\tau}{2^{i+1}} \geq \frac{\sum_{u \in E_i} m(u)}{4k} + \frac{|E_i|\,\tau}{2^{i+2}k} - \frac{\left|T_j^{i-h}\right|\tau}{2^{i+1}} \quad ,\end{aligned}$$

where the two inequality hold since $\tau/2^i \geq m(u) \geq \tau/2^{i+1}$ for every element $u \in E_i$.

Adding the induction hypothesis for $i - h$ to the above inequality, we get

$$\begin{aligned}\sum_{u \in T_j^i} m(u) \geq{}& \frac{\sum_{u \in E_i} m(u)}{4k} + \frac{|E_i|\,\tau}{2^{i+2}k} - \frac{\left|T_j^{i-h}\right|\tau}{2^{i+1}} + \frac{1}{4k} \cdot \sum_{\substack{j \leq r \leq i-h \\ r \equiv j \pmod h}} \sum_{u \in E_r} m(u) + \frac{|T_j^{i-h}|\tau}{2^{i-h+2}k} \\ ={}& \frac{1}{4k} \cdot \sum_{\substack{j \leq r \leq i \\ r \equiv j \pmod h}} \sum_{u \in E_r} m(u) + \frac{|E_i|\,\tau}{2^{i+2}k} + \frac{\left|T_j^{i-h}\right|\tau}{2^{i+2}k}\left(2^h - 2k\right) \\ \geq{}& \frac{1}{4k} \cdot \sum_{\substack{j \leq r \leq i \\ r \equiv j \pmod h}} \sum_{u \in E_r} m(u) + \frac{|E_i|\,\tau}{2^{i+2}k} + \frac{\left|T_j^{i-h}\right|\tau}{2^{i+2}k} \geq \frac{1}{4k} \cdot \sum_{\substack{j \leq r \leq i \\ r \equiv j \pmod h}} \sum_{u \in E_r} m(u) + \frac{\left|T_j^i\right|\tau}{2^{i+2k}} \quad ,\end{aligned}$$

where the second inequality holds by the definition of h, and the last inequality holds since every element of T_j^i must belong either to E_i or to T_j^{i-h}. □

Next, we restate Lemma 7 and then provide its proof.

Lemma 7. *If $(\mathcal{N}, \mathcal{I})$ is a k-set system, then Algorithm 2 returns a set T such that*

$$f(T) \geq \frac{f(E \cup U) - \tau/4}{4kh(2k+1)} = \frac{f(E \cup U) - \tau/4}{O(k^2 \log k)}$$

for every set $U \in \mathcal{I}$.

Proof. Since the output set of Algorithm 2 is the best set among $T_0, T_1, \ldots, T_{h-1}$, we get

$$f(T) = \max_{0 \leq j < h} f(T_j) \geq \frac{\sum_{j=0}^{h-1} f(T_j)}{h} \geq \frac{f(\varnothing) + \sum_{j=0}^{h-1} \sum_{i \in \{0 \leq i \leq \ell | i \equiv j (\bmod\ h)\}} \sum_{u \in E_i} m(u)}{4kh}$$
$$= \frac{f(\varnothing) + \sum_{i=0}^{\ell} \sum_{u \in E_i} m(u)}{4kh} = \frac{f(\varnothing) + f(E \mid \varnothing)}{4kh} \geq \frac{f(E \cup U) - \tau/4}{4kh(2k+1)} \enspace ,$$

where the second inequality follows from Lemma 8, and the last inequality follows from Lemma 15 and the non-negativity of f. □

Using the last lemma and the framework described in Section 4, we can now prove our result for k-set system constraints.

Theorem 9. *There is a streaming $O(k^2 \log k) = \tilde{O}(k^2)$-approximation algorithm for the problem of maximizing a non-negative submodular function subject to a k-set system constraint.*

Proof. If the objective function is monotone, then the theorem follows immediately from Lemma 7 by setting U to be the optimal solution, since this choice implies that the output set of Algorithm 2 has a value of at least

$$\frac{f(E \cup U) - \tau/4}{O(k^2 \log k)} \geq \frac{\texttt{OPT} - \texttt{OPT}/2}{O(k^2 \log k)} = \frac{\texttt{OPT}}{O(k^2 \log k)} \enspace ,$$

where the inequality holds since $\tau \leq 2M = 2 \max_{u \in \mathcal{N}, \{u\} \in \mathcal{I}} f(\{u\}) \leq 2\texttt{OPT}$ because $\{u\}$ is a candidate set to be $\texttt{OPT}$ whenever it is feasible.

Otherwise, if the objective function is non-monotone, then we observe that Lemma 7 implies that Algorithm 2 is an $(O(k^2 \log k), \tau/4)$-approximation algorithm when we take $A = E$. Thus, by setting $r = 4$, using Algorithm 2 as StreamingAlg and using the $(k + O(\sqrt{k}))$-approximation algorithm RepeatedGreedy due to (Feldman et al., 2017) (mentioned in Appendix C) as ConstrainedAlg, we get via our framework a streaming algorithm whose output set is guaranteed to have a value of at least

$$\frac{(1 - 1/r) \cdot \texttt{OPT} - \gamma}{\alpha + \beta} = \frac{(3/4) \cdot \texttt{OPT} - \tau/4}{O(k^2 \log k) + k + O(\sqrt{k})} = \frac{3 \cdot \texttt{OPT} - \tau}{O(k^2 \log k)} \geq \frac{\texttt{OPT}}{O(k^2 \log k)} \enspace .$$ □

We now prove a stronger version of Lemma 8 for k-extendible constraints. This version takes advantage of the stronger guarantee of the unweighted greedy algorithm for such constraints, which is given by Lemma 4.

Lemma 16. *If $(\mathcal{N}, \mathcal{I})$ is a k-extendible system, then for every integer $0 \leq j < h$ we have*

$$f(T_j \mid \varnothing) \geq \frac{1}{k} \cdot \sum_{\substack{0 \leq i \leq \ell \\ i \equiv j (\bmod\ h)}} \sum_{u \in E_i} m(u) \enspace .$$

Proof. We use in this lemma the notation defined in the proof of Lemma 8. Furthermore, the same arguments used in the proof of Lemma 8 to show that Lemma 8 follows from Inequality (5) can also be used to show that the current lemma follows from the following claim. For every integer $-h \leq i \leq \ell$ and $j = i \bmod h$,

$$\sum_{u \in T_j^i} m(u) \geq \frac{1}{4} \cdot \sum_{\substack{j \leq r \leq i \\ r \equiv j (\bmod\ h)}} \sum_{u \in E_r} m(u) + \frac{|T_j^i| \tau}{2^{i+2}} \enspace . \tag{6}$$

Thus, the rest of this proof is devoted to proving this claim by induction on i.

For $i < 0$, Inequality (6) holds since $j \geq 0 > i$ and $T_j^i = \varnothing$, which implies that the value of both sides of the inequality is 0. Next, we need to prove Inequality (6) for an integer $0 \leq i \leq \ell$ under the assumption that it holds for every $-h \leq i' < i$. Recall that the set T_j^i is obtained by starting with T_j^{i-h}, and then greedily adding elements of E_i to it. Thus, T_j^i can be viewed as the output of the greedy algorithm when this algorithm is given the elements of T_j^{i-h} first, and then the elements of E_i. Given this point of view, since E_i is independent, Lemma 4 guarantees

$$|E_i \setminus T_j^i| \leq k \cdot |T_j^i \setminus E_i| = k \cdot |T_j^{i-h}| \ .$$

Hence,

$$\sum_{u \in T_j^i \setminus T_j^{i-h}} m(u) = \sum_{u \in E_i \cap T_j^i} m(u) \geq \left|E_i \cap T_j^i\right| \cdot \frac{\tau}{2^{i+1}} = \left[|E_i| - \left|E_i \setminus T_j^i\right|\right] \cdot \frac{\tau}{2^{i+1}}$$

$$\geq \sum_{u \in E_i} \frac{\tau}{2^{i+2}} + \frac{|E_i| \cdot \tau}{2^{i+2}} - \frac{k\tau \left|T_j^{i-h}\right|}{2^{i+1}} \geq \frac{\sum_{u \in E_i} m(u)}{4} + \frac{|E_i| \cdot \tau}{2^{i+2}} - \frac{k\tau \cdot \left|T_j^{i-h}\right|}{2^{i+1}} \ ,$$

where the first and third inequalities hold since $\tau/2^i \geq m(u) \geq \tau/2^{i+1}$ for every element $u \in E_i$.

Adding the induction hypothesis for $i - h$ to the above inequality, we get

$$\sum_{u \in T_j^i} m(u) \geq \frac{\sum_{u \in E_i} m(u)}{4} + \frac{|E_i| \cdot \tau}{2^{i+2}} - \frac{k\tau \cdot \left|T_j^{i-h}\right|}{2^{i+1}} + \frac{1}{4} \cdot \sum_{\substack{j \leq r \leq i-h \\ r \equiv j \pmod h}} \sum_{u \in E_r} m(u) + \frac{\left|T_j^{i-h}\right| \tau}{2^{i-h+2}}$$

$$= \frac{1}{4} \cdot \sum_{\substack{j \leq r \leq i \\ r \equiv j \pmod h}} \sum_{u \in E_r} m(u) + \frac{|E_i| \cdot \tau}{2^{i+2}} + \frac{\left|T_j^{i-h}\right| \cdot \tau}{2^{i+2}} \left(2^h - 2k\right)$$

$$\geq \frac{1}{4} \cdot \sum_{\substack{j \leq r \leq i \\ r \equiv j \pmod h}} \sum_{u \in E_r} m(u) + \frac{|E_i| \cdot \tau}{2^{i+2}} + \frac{|T_{i-h}| \cdot \tau}{2^{i+2}} \geq \frac{1}{4} \cdot \sum_{\substack{j \leq r \leq i \\ r \equiv j \pmod h}} \sum_{u \in E_r} m(u) + \frac{|T_i| \cdot \tau}{2^{i+2}} \ ,$$

where the second inequality holds by the definition of h, and the last inequality holds since every element of T_j^i belongs either to E_i or to T_j^{i-h}. □

Using Lemma 16 we can prove the Theorem 10. The proof is identical to the proof of Theorem 9, except for the use Lemma 16 instead of Lemma 8.

B. Counter Examples for Inequality (2)

In Section 4, we discussed the framework proposed by Mirzasoleiman et al. (2018) for maximizing a non-monotone submodular function using an algorithm for monotone functions. This framework requires the input streaming algorithm to satisfy the inequality

$$f(S) \geq \alpha \cdot f(S \cup T) \ ,$$

where S as the output of the algorithm, T is an arbitrary feasible solution and α is a positive value. In the rest of this section, we provide two instances of the streaming maximization problem under a simple cardinality constraint k. These instances show that the algorithms of (Chekuri et al., 2015), (Chakrabarti & Kale, 2015) and (Buchbinder et al., 2019) fail to satisfy Eq. (2) for any constant α.

Both our instances are based on a graph-cut function $f\colon 2^V \to \mathbb{R}_{\geq 0}$ over vertices of a directed and weighted graph $G(V, E)$. This function is defined as follows:

$$f(S) = \sum_{u \in S} \sum_{v \in V \setminus S} w_{u,v} \ , \tag{7}$$

where $w_{u,v}$ is the weight of the edge $e = (u, v)$. It is easy to see that f is a (usually non-monotone) submodular function. Furthermore, in our examples we assume the graph contains $3\rho + 1$ vertices named $u_0, u_1, u_2, \ldots, u_{3\rho}$. The vertex u_0 does not appear in the input stream at all (it is there only for the purpose of allowing the description of the objective function as a cut function), and the other vertices appear in the stream in the order of their subscripts.

B.1. Example for the Algorithms of Chekuri et al. and Chakrabarti and Kale

The streaming algorithm of Chekuri et al. (2015), in the context of a cardinality constraint, is given as Algorithm 3. The algorithm of Chakrabarti & Kale (2015) is very similar, and exhibits exactly the same behavior given the example we describe in this section, and therefore, we do not restate it here.

Algorithm 3: Streaming Algorithm of Chekuri et al. (2015)

$S \leftarrow \varnothing$.
while *there are more elements in the stream* **do**
    $u \leftarrow$ next element in the stream.
    if $|S| < \rho$ **then**
        if $f(u \mid S) \geq 0$ **then**
            $S \leftarrow S \cup \{u\}$.
    else
        $u' \leftarrow \arg\min_{x \in S} f(x : S)$, where $f(x : S)$ is the marginal contribution of x to the part of S that arrived before x itself.
        if $f(u \mid S) \geq 2 \cdot f(u' : S)$ **then**
            $S \leftarrow (S \setminus \{u'\}) \cup \{u\}$.
return S.

The counter example we suggest for Algorithm 3 is given by the weighted graph $G_1(V, E)$ shown in Fig. 5. The weight of the black edges is 1, and the weight of the blue edges is $2 + \epsilon$ for some small and positive value ϵ.

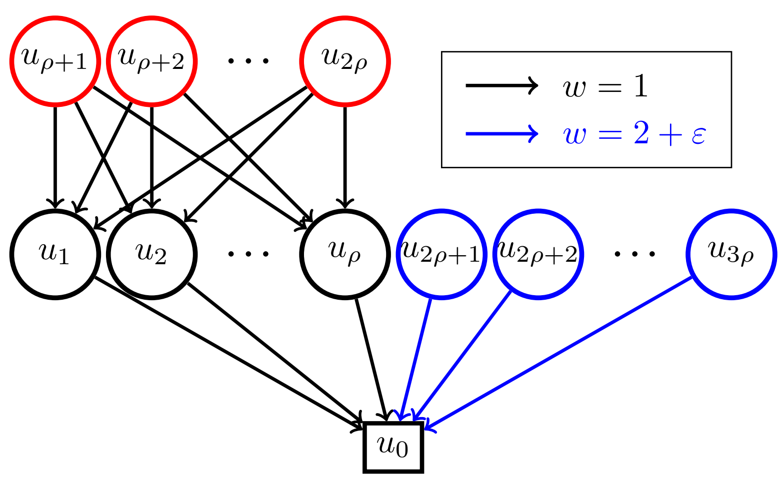


Figure 4. Weighted graph $G_1(V, E)$ used to define the counter example for the algorithm of Chekuri et al. (2015).

Lemma 17. *Assume S is the output of Algorithm 3 for maximizing the graph-cut function f (of the graph $G_1(V, E)$ and as defined in Eq. (7)) under a cardinality constraint ρ. Then,*

$$f(S) \leq \frac{2 + \epsilon}{\rho} \cdot f(S \cup S^*) \ ,$$

where S^* is the optimal solution.

Proof. First, it is clear that the optimal solution is the set $S^* = \{u_{\rho+1}, u_{\rho+2}, \dots, u_{2\rho}\}$, for which $f(S^*) = \rho^2$. When the first ρ elements $V_1 = \{u_1, \dots, u_\rho\}$ arrive, all of them are added to the solution S as the marginal gain of each one of them is 1. Furthermore, when the elements $u \in S^*$ arrive, it is obvious that $f(u \mid S) = 0$, and therefore,

$$f(u \mid S) < f(e' : S) = 1 \quad \forall u' \in S \ .$$

Hence, none of the elements of S^* would be added to the solution. Finally, it is straightforward to see that all elements in $V_2 = \{u_{2\rho+1}, \dots, u_{3\rho}\}$ would replace an element in V_1 and be in the final solution S. This is true because for $u \in V_2$ we have $f(u \mid S) = 2 + \epsilon$, which is larger than $2 \cdot f(u' : S)$ for $u' \in V_1$. The lemma now follows by observing that $f(S) = f(V_2) = (2+\epsilon)\rho$ and $\rho^2 = f(S^*) \leq f(S \cup S^*)$. □

B.2. Example for the Algorithm of Buchbinder et al.

The streaming algorithm of Buchbinder et al. (2019) is given as Algorithm 4.

Algorithm 4: Streaming Algorithm of Buchbinder et al. (2019)

$S \leftarrow \varnothing$.
while *there are more elements in the stream* **do**
  $u \leftarrow$ next element in the stream.
  if $|S| < \rho$ **then**
    if $f(u \mid S) \geq 0$ **then**
      $S \leftarrow S \cup \{u\}$.
  else
    $u' \leftarrow \arg\max_{x \in S} f(S \setminus \{x\} \cup \{u\})$.
    if $f((S \setminus \{u'\}) \cup \{u\}) - f(S) \geq f(S)/\rho$ **then**
      $S \leftarrow (S \setminus \{u'\}) \cup \{u\}$.
return S.

In this section, the objective function of the counter example is given by the graph-cut function f of the weighted graph $G_2(V, E)$ shown in Fig. 5. This graph has the same structure as the graph G_1 from Appendix B.1, but its weight selection is more involved. Specifically, in the graph G_2, the weight of the black edges is 1 and there exist ρ blue edges with weights $w_1, w_2, \dots, w_\rho$ given by $w_1 = 2$ and

$$w_i = \frac{2\rho + 1 - i + \sum_{j=1}^{i-1} w_j}{\rho} \quad \forall\, i \geq 2 \ .$$

Lemma 18. *Assume S is the output of Algorithm 4 for maximizing the graph-cut unction f (of the graph $G_2(V, E)$ and as defined in Eq. (7)) under a cardinality constraint ρ. Then, for $\rho \geq 1 + e$,*

$$f(S) \leq \frac{e}{\rho} \cdot f(S \cup S^*) \ ,$$

where S^* is optimal solution.

Proof. We begin the proof by showing, through an induction argument, that $w_i = 2 + \sum_{j=1}^{i-1} \binom{i-1}{j} \rho^{-j}$. The base of induction is trivial as $w_1 = 2$. Assuming the induction argument is correct for $h \leq i - 1$, we prove that it is also correct for i.

$$\begin{aligned} w_i = \frac{2k + 1 - i + \sum_{j=1}^{i-1} w_j}{\rho} &= 2 + \frac{1 - i + \sum_{j=1}^{i-1}\left(2 + \sum_{\ell=1}^{j-1}\binom{j-1}{\ell}\rho^{-\ell}\right)}{\rho} \\ &= 2 + \frac{i - 1 + \sum_{\ell=1}^{i-2}\sum_{j=\ell+1}^{i-1}\binom{j-1}{\ell}\rho^{-\ell}}{\rho} \overset{(a)}{=} 2 + \frac{i - 1 + \sum_{\ell=1}^{i-2}\binom{i-1}{\ell+1}\rho^{-\ell}}{\rho} \\ &= 2 + \binom{i-1}{1}\rho^{-1} + \sum_{\ell=1}^{i-2}\binom{i-1}{\ell+1}\rho^{-(\ell+1)} = 2 + \sum_{j=1}^{i-1}\binom{i-1}{j}\rho^{-j} \ , \end{aligned}$$

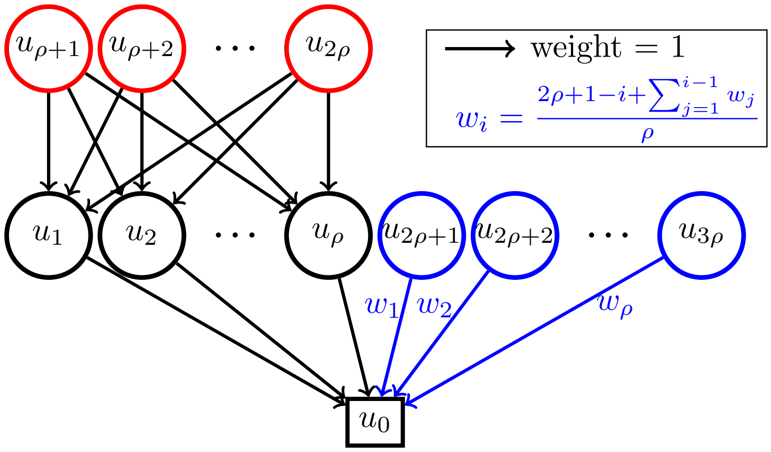


Figure 5. Weighted graph $G_2(V, E)$ used to define the counter example for the algorithm of Buchbinder et al. (2019).

where in (a) we use the following well-known equality $\sum_{\ell=j}^{i} \binom{\ell}{j} = \binom{i+1}{j+1}$, which implies $\sum_{\ell=j+1}^{i+1} \binom{\ell-1}{j} = \binom{i+1}{j+1}$. As a corollary of this proof, we get $w_i \leq 2 + [(1+\rho^{-1})^{i-1} - 1] \leq 1 + (1+\rho^{-1})^{\rho} \leq 1 + e$, which implies that the optimal solution is $S^* = \{e_{\rho+1}, e_{\rho+2}, \dots, e_{2\rho}\}$ whose value is $f(S^*) = \rho^2$.

When the first ρ elements $V_1 = \{u_1, \dots, u_\rho\}$ arrive, all of them are added to the solution S as the marginal gain of each one of them is 1. Thus, when an element $u \in S^*$ arrive, we have $f(S \setminus \{u'\} \cup \{u\}) - f(S) = 0$ for every $u' \in S$. Therefore, none of the elements of S^* would be added to the solution. Next, we prove that all elements in $V_2 = \{u_{2\rho+1}, \dots, u_{3\rho}\}$ would replace an element in V_1 and be in the final solution S of Algorithm 4. Again, we prove this claim by induction. When $u_{2\rho+1}$ arrives, for all $u' \in S$ we have:

$$f(S \setminus \{u'\} \cup \{u_{2\rho+1}\}) - f(S) = 1 \geq 1 = \frac{f(S)}{\rho} \ ,$$

and $u_{2\rho+1}$ replaces one of the elements from V_1. Assume now that elements $\{u_{2\rho+1}, \dots, u_{2\rho+i-1}\}$ for some integer $i < \rho$ have each replaced one of the element of V_1, and let us show that this implies that $u_{2\rho+i}$ would also replace one element u' from V_1. This is true because for every such element $u' \in S$ we have

$$f(S \setminus \{u'\} \cup \{u_{2\rho+i}\}) - f(S) = w_i - 1 = \frac{\rho + 1 - i + \sum_{j=1}^{i-1} w_j}{\rho} = \frac{\rho - (i-1) + \sum_{j=1}^{i-1} w_j}{\rho} = \frac{f(S)}{\rho} \ .$$

As a corollary, we get that for the final solution $S = V_2$, we have

$$\begin{aligned} f(S) &= \sum_{i=1}^{\rho} w_i = 2\rho + \sum_{i=1}^{\rho} \sum_{j=1}^{i-1} \binom{i-1}{j} \rho^{-j} = 2\rho + \sum_{i=1}^{\rho-1} \sum_{j=i}^{\rho-1} \binom{j}{i} \rho^{-i} \\ &= 2\rho + \sum_{i=1}^{\rho-1} \binom{\rho}{i+1} \rho^{-i} = 2\rho + \rho\left((1+\rho^{-1})^{\rho} - 2\right) \leq e\rho \ . \end{aligned}$$

This proves the lemma since $f(S^* \cup V_2) \geq f(S^*) \geq \rho^2$. □

C. A Deterministic Streaming Algorithm for Submodular Maximization Subject to a k-Matchoid Constraint

As discussed in Section 4, Chekuri et al. (2015) already described a method to convert their algorithm for the problem of maximizing a non-negative monotone submodular function subject to a k-matchoid constraint into a deterministic algorithm that works also for non-monotone functions. The algorithm they obtained in this way has an approximation guarantee of $8k + \gamma$, where γ is the approximation ratio of the offline algorithm used in the conversion. In this section we show that via our framework it is possible to get a better guarantee for the same problem.[8]

The algorithm that we use as StreamingAlg is the deterministic algorithm for monotone functions designed by (Chekuri et al., 2015). Following we state some properties of this algorithm. We begin with a bound on its approximation guarantee. For this bound, let us denote by S the final solution of the algorithm and by A the set of elements that ever appeared in the solution maintained by the algorithm.

Lemma 19 (Lemma 11 of (Chekuri et al., 2015))**.** *Let $T \in \mathcal{I}$ be an independent set. Then,*

$$f(T \cup A) \leq \rho\alpha' + \frac{(1+\beta')^2}{\beta'} \cdot k \cdot f(S) \ ,$$

where ρ is an upper bound on the cardinality of the optimal set and the two non-negative parameters α' and β' are inputs to the algorithm.

In our notation, the last lemma implies that the deterministic algorithm of (Chekuri et al., 2015) is an $(k(1+\beta')^2/\beta', \rho\alpha')$-approximation algorithm. Chekuri et al. (2015) also proved that this algorithm has the space complexity of a semi-streaming algorithm as long as α' is at least a constant fraction of $\texttt{OPT}/\rho$. In particular, they showed the following lemma, which shows that in this regime the size of A is linear in ρ.

Lemma 20 (Lemma 5 of (Chekuri et al., 2015))**.** $|A| \leq \texttt{OPT}/\alpha'$.

For ConstrainedAlg we use the RepeatedGreedy algorithm of (Feldman et al., 2017), which works for general k-set systems constraints (k-matchoid constraints are a special case of k-set systems constraints). The approximation ratio of this algorithm is $k + O(\sqrt{k})$, and it can be implemented to run in linear space. Plugging these two algorithms into our framework, we get the following corollary.

Corollary 21. *For every $\varepsilon \in (0, 1/6]$, by setting $\beta' = 1$, $\alpha' = \varepsilon \cdot \texttt{OPT}/\rho$ and $r = \lceil 1/\varepsilon \rceil$, our framework produces a deterministic streaming algorithm for the problem of maximizing a non-negative (not necessary monotone) submodular function subject to a k-matchoid constraint. The approximation ratio of this algorithm is at most $(5 + 15\varepsilon)k + O(\sqrt{k})$.*

Proof. By Theorem 6, the algorithm obtained in this way produces a set whose value is at least

$$\frac{(1 - 1/r) \cdot \texttt{OPT} - \gamma}{\alpha + \beta} \geq \frac{(1-\varepsilon) \cdot \texttt{OPT} - \varepsilon \cdot \texttt{OPT}}{4k + k + O(\sqrt{k})} = \frac{(1-2\varepsilon) \cdot \texttt{OPT}}{5k + O(\sqrt{k})} \ ,$$

and this implies that the approximation ratio of the algorithm is at most

$$\frac{5k + O(\sqrt{k})}{1 - 2\varepsilon} \leq (5 + 15\varepsilon) \cdot k + O(\sqrt{k}) \ . \qquad \square$$

Before concluding this section, we note that the algorithm suggested by Corollary 21 assumes pre-knowledge of $\texttt{OPT}$ and ρ since these values are necessary for calculating α'. It is possible to guess the value of $\texttt{OPT}$ up to a small error using a technique originally due to (Badanidiyuru et al., 2014), and this has no effect on the approximation guarantee of the algorithm (but slightly increases its space complexity). As the details of this are discussed by (Chekuri et al., 2015), we avoid repeating them here. Regarding ρ, Chekuri et al. (2015) assumed pre-knowledge of ρ, and we take the same approach in this section. However, it is possible to modify the algorithm to avoid the need to have this pre-knowledge, and we demonstrate the technique leading to this possibility when discussing our algorithm for general k-set systems.

D. Extended Version of Our Algorithm

In this section we present and analyze an extended version of our algorithm from Section 5 which need not assume pre-knowledge of ρ and τ. We do that in two steps. In Section D.1 we present a version of our algorithm that still assumes pre-access to τ, but not to ρ; and in Section D.2 we show how to remove the need to known τ as well.

[8]Technically, the algorithm of (Chekuri et al., 2015) is very similar to the algorithm obtained via our framework for $r = 2$, and the approximation guarantee they obtained can be reproduced using our framework by setting r to this value. However, as our framework can handle other values of r as well, we manage to get a better guarantee by assigning a larger value to r.

D.1. Algorithm without Access to ρ

As an alternative to ρ, the algorithm we present in this section (which is given as Algorithm 5) uses the size of a set G produced by running the unweighted greedy algorithm on the entire input. Since the value of this alternative can increase over time, the algorithm has to create additional sets E_i on the fly. We also note that the formula for ℓ used by Algorithm 5 is slightly different than the corresponding formula in Algorithm 2.

Algorithm 5: Streaming Algorithm for k-set Systems (with no pre-access to ρ)

Input: a value $\tau \in [M, 2M]$ and the parameter k of the constraint.
Output: a solution $T \in \mathcal{I}$
Let $G \leftarrow \varnothing$, $\ell \leftarrow -1$ and $h \leftarrow \lceil \log_2(2k+1) \rceil$.
for *every element* u *arriving* **do**
    if $G + u \in \mathcal{I}$ **then** Add u to G.
    Let $\ell' \leftarrow \lfloor 2\log_2(k|G|) + 3 \rfloor$.
    for $i = \ell + 1$ **to** ℓ' **do** Initialize $E_i \leftarrow \varnothing$.
    Update $\ell \leftarrow \ell'$.
    Let $m(u) \leftarrow f\left(u \mid \cup_{i=0}^{\ell} E_i\right)$.
    if $m(u) > 0$ **then** Let $i(u) \leftarrow \lfloor \log_2(\tau / m(u)) \rfloor$ **else** Let $i(u) \leftarrow \infty$.
    if $0 \leq i(u) \leq \ell$ *and* $E_{i(u)} + u \in \mathcal{I}$ **then** Update $E_{i(u)} \leftarrow E_{i(u)} + u$.
for $j = 0$ ***to*** $h - 1$ **do**
    Let $i \leftarrow j$ and $T_j \leftarrow \varnothing$.
    while $i \leq \ell$ **do**
        while *there is an element* $u \in E_i$ *such that* $T_j + u \in \mathcal{I}$ **do** Update $T_j \leftarrow T_j + u$.
        $i \leftarrow i + h$.
return *the set* T *maximizing* f *among* $T_0, T_1, \cdots, T_{h-1}$.

We begin the analysis of Algorithm 5 by showing that it has the space complexity of a semi-streaming algorithm.

Lemma 22. *Algorithm 5 stores $O(\rho(\log \rho + \log k)) = \tilde{O}(\rho)$ elements at every given time point.*

Proof. Observe that the set G is kept as an independent set by the algorithm, and thus, its size is at most ρ, and we get that at all times $\ell = O(\log(k\rho)) = O(\log k + \log \rho)$. We now also note that Algorithm 5 stores elements only in the sets $E_0, E_1, \ldots, E_\ell$ and the sets $T_0, T_1, \ldots, T_{h-1}$. Since these sets are kept independent by the algorithm, each one them contains at most ρ elements. Thus, the number of elements stored by Algorithm 5 is upper bounded by

$$(\ell + h)\rho = [O(\log \rho + \log k)]\rho = O(\rho(\log \rho + \log k)) \quad . \qquad \square$$

We now get to analyzing the approximation ratio of Algorithm 5. One can verify that all the proofs in the analysis of the approximation ratio of Algorithm 2 from Section 5 apply (as is) also to Algorithm 5, except for the proof of Lemma 15. Thus, in the rest of this section our objective is to show that Lemma 15 applies to Algorithm 5, despite the fact that its original proof from Section 5 does not apply to it.

Let us define R to be a set including every element $u \in \mathcal{N}$ for which either $i(u) < 0$ or $i(u) > \ell$ at the moment of u's arrival. The following lemma allows us to bound the value of the elements in R.

Lemma 23. *For every independent set S, $\sum_{u \in S \cap R} m(u) \leq \tau/4$.*

Proof. Let us denote the elements of $S \cap R$ by $u_1, u_2, \ldots, u_r$ in the order of their arrival. For every $1 \leq j \leq |S \cap R|$, since $u_j \in R$, at the moment in which either u_j arrived $i(u)$ was either negative or larger than ℓ. However, since S is independent, $\tau \geq M \geq f(\{u_j\}) \geq m(u_j)$, and thus, the first option cannot happen, which leaves us only with the case

$$\left\lfloor \log_2\left(\frac{\tau}{m(u_j)}\right) \right\rfloor \geq \ell + 1 \Rightarrow \frac{\tau}{m(u_j)} \geq 2^{\ell+1} \Rightarrow m(u_j) \leq \frac{\tau}{2^{\ell+1}} \leq \frac{\tau}{2^{2\log_2(k|G|)+3}} = \frac{\tau}{8k^2|G|^2} \quad .$$

We now observe that at the moment referred to by the previous paragraph the algorithm already received at least j elements of S, and thus, the size of G was at least j/k (recall that the unweighted greedy algorithm is a k-approximation algorithm). Hence,

$$m(u_j) \leq \frac{\tau}{8j^2} \ .$$

Summing up this inequality over all $1 \leq j \leq |S \cap R|$, we get

$$\sum_{u \in S \cap R} m(u) \leq \sum_{j=1}^{|S \cap R|} \frac{\tau}{8j^2} \leq \frac{\tau}{8} \cdot \left[1 + \int_1^\infty \frac{dx}{x^2}\right] = \frac{\tau}{8} \cdot \left[1 - \left[\frac{1}{x}\right]_1^\infty\right] = \frac{\tau}{4} \ . \qquad \square$$

Using the last lemma, we can now prove that Lemma 15 applies also to Algorithm 5. Recall that $E = \cup_{i=0}^{\ell} E_i$.

Lemma 15. *For every set $S \in \mathcal{I}$, $f(E \mid \varnothing) = \sum_{i=0}^{\ell} \sum_{u \in E_i} m(u) \geq \frac{f(S \cup E \mid \varnothing) - \tau/4}{2k+1}$.*

Proof. First, note that we have $f(E \mid \varnothing) = \sum_{i=0}^{\ell} \sum_{u \in E_i} m(u)$ because $m(u)$ is the marginal contribution of u with respect to the elements that were added to $\cup_{i=0}^{\ell} E_i$ before u. Let us also define, for every integer $0 \leq i \leq \ell$, $S_i = \{u \in S \setminus R \mid i(u) = i\}$. Then,

$$\begin{aligned} f(E \mid \varnothing) = \sum_{i=0}^{\ell} \sum_{u \in E_i} m(u) \geq \sum_{i=0}^{\ell} |E_i| \cdot \frac{\tau}{2^{i+1}} \geq \frac{1}{k} \cdot \sum_{i=0}^{\ell} |S_i| \cdot \frac{\tau}{2^{i+1}} \\ \geq \frac{1}{2k} \cdot \sum_{i=0}^{\ell} \sum_{u \in S_i} m(u) = \frac{1}{2k} \cdot \left[\sum_{u \in S} m(u) - \sum_{u \in S \cap R} m(u)\right] \ , \end{aligned}$$

where the first and third inequalities hold since an element u is added to a set E_i only when $i = i(u)$, the second inequality holds since one can view E_i as the output of running the unweighted greedy algorithm on a ground set which includes the independent set S_i as a subset, and the last equality holds since $0 \leq i(u) \leq \ell$ for every element $u \notin R$ because ℓ can only increase during the execution of Algorithm 5.

By the submodularity of f, we can immediately get

$$\sum_{u \in S} m(u) \geq \sum_{u \in S} f(u \mid E) \geq f(S \mid E) = f(S \cup E \mid \varnothing) - f(E \mid \varnothing) \ .$$

Combining the two above inequalities and the guarantee of Lemma 23 gives us

$$f(E \mid \varnothing) \geq \frac{1}{2k} \cdot \left[f(S \cup E \mid \varnothing) - f(E \mid \varnothing) - \frac{\tau}{4}\right] \ ,$$

and the lemma follows by rearranging this inequality. $\square$

D.2. Algorithm without Access to τ

In this section we explain how to modify our algorithm so that it does not need to access to τ. This modification is based on a technique due to (Badanidiyuru et al., 2014), but it is made slightly more involved since we assume here no pre-knowledge of ρ, which is not the case in (Badanidiyuru et al., 2014). Following is the crucial observation that we use in this section.

Observation 24. *Except for the sake of maintaining G, Algorithm 5 ignores an element u if $\{u\} \notin \mathcal{I}$ or $f(\{u\}) \leq \tau / 2^{2\log_2(k|G_u|)+4}$, where $|G_u|$ is the size of G immediately after the processing of u.*

Proof. The case of $\{u\} \notin \mathcal{I}$ is simple, so let us consider only the case $f(\{u\}) \leq \tau/2^{2\log_2(k|G_u|)+4}$. Let ℓ_u be the value of ℓ immediately after the processing of u by Algorithm 5. The value $i(u)$ calculated by Algorithm 5 when processing u obeys

$$\begin{aligned} i(u) &= \lfloor \log_2(\tau/m(u)) \rfloor \geq \lfloor \log_2(\tau/f(\{u\})) \rfloor \\ &\geq \lfloor \log_2(2^{2\log_2(k|G_u|)+4} \rfloor = 2\log_2(k|G_u|) + 4 \geq \ell_u + 1 \ , \end{aligned}$$

where the first inequality follows from the submodularity of f, and the second inequality follows from the condition of the lemma. $\square$

Using Observation 24 in mind, we now give the algorithm of this section as Algorithm 6. This algorithm runs multiple copies of Algorithm 5, each having a different τ values. The intuitive objective of the algorithm is to have a copy with $\tau = x$ for every x which is a power of 2, might belong to the range $[M, 2M]$ given the input so far, and might have accepted some element so far even given Observation 24. Since the set G is maintained by Algorithm 5 in a way which is independent of τ, Algorithm 6 maintains G itself, and we assume that the copies of Algorithm 5 that it creates use this set G rather than maintaining their own set G.

Algorithm 6: Streaming Algorithm for k-set Systems (with no pre-access to ρ and τ)

Input: the parameter k of the constraint.
Output: a solution $T \in \mathcal{I}$
Let $G \leftarrow \varnothing$, $M' \leftarrow -\infty$.
for *every element u arriving* **do**
  if $G + u \in \mathcal{I}$ **then** Add u to G.
  if $\{u\} \in \mathcal{I}$ **then** Update $M' \leftarrow \max\{M', f(\{u\})\}$.
  Let $L = \{2^i \mid i \text{ is integer and } M' \leq 2^i \leq M' \cdot 2^{2\log_2(k|G|)+5}\}$.
  Delete any existing copy of Algorithm 5 whose τ value does not belong to L.
  for *every $x \in L$* **do**
    Create a copy of Algorithm 5 with $\tau = x$, unless such a copy already exists.
  Pass u to all the copies of Algorithm 5 that currently exist.
return *the set maximizing f among the output sets of all the currently existing copies of Algorithm 5.*

We begin the analysis of Algorithm 6 by analyzing its space complexity.

Observation 25. *The number of elements stored by Algorithm 6 is larger than the number of elements stored by Algorithm 5 by a factor of $O(\log k + \log \rho)$.*

Proof. It suffices to show that Algorithm 6 maintains at most $O(\log k + \log \rho)$ copies of Algorithm 5 at any given time, and to do that it suffices to show that the size of the set L created by Algorithm 6 in every iteration is upper bounded by $O(\log k + \log \rho)$. Note that the size of this set is at most

$$\left\lceil \log_2\left(\frac{M' \cdot 2^{2\log_2(k|G|)+5}}{M'}\right)\right\rceil = \left\lceil \log_2\left(2^{2\log_2(k|G|)+5}\right)\right\rceil$$
$$= \lceil 2\log_2(k|G|) + 5\rceil \leq 2\log_2(k\rho) + 6 = O(\log_2 k + \log_2 \rho) \enspace . \qquad \square$$

Next, let us show that the approximation guarantee of Algorithm 6 is at least as good as the guarantee of Algorithm 5.

Lemma 26. *Let S be the output set of Algorithm 6. Then, $f(S)$ is at least as large as the value of the output set of Algorithm 5 when executed with some value $\tau \in [M, 2M]$.*

Proof. Let u be the first element to arrive which obeys both $\{u\} \in \mathcal{I}$ and $f(\{u\}) \geq M/2^{2\log_2(k|G_u|)+4}$. The set L generated while processing this element necessarily includes a value $\bar{\tau}$ within the range $[M, 2M]$ because while processing u we have

$$M' \cdot 2^{2\log_2(k|G_u|)+5} = f(\{u\}) \cdot 2^{2\log_2(k|G_u|)+5} \geq 2M \enspace ,$$

and

$$M' = f(\{u\}) \leq M \enspace .$$

Moreover, we can observe that $\bar{\tau}$ belongs to any list L generated after this point by Algorithm 6 because the value of M' can only increase and the inequality $M' \leq M$ remains valid until the end of the algorithm. Thus, a copy of Algorithm 5 with $\tau = \bar{\tau}$ exists from the arrival of u until Algorithm 6 terminates. Let us denote this copy by C.

Observation 24 and the definition of u guarantee that the copy C ignores the elements that arrived before u if they are passed to it (except for the purpose of maintaining G, but we assume in this section that this work is done by Algorithm 6 itself rather than by the copies of Algorithm 5). Thus, the output of C is identical to the output it would have produced if all the elements had been passed to it, including elements that arrived before the creation of C. In other words, the output set of C

is the output set of Algorithm 5 when executed with some value $\tau = \bar{\tau} \in [M, 2M]$ on the entire input. Since C survives until the end of the execution of Algorithm 6, this implies that the output set of Algorithm 6 is at least as good as that. $\square$

E. Supplementary Experiments

In this section, we provide additional experiments as well as a theoretical guarantee referred to in Section 6.2.

E.1. Theoretical Guarantee for Section 6.2

As promised in Section 6.2, we explain in this section why the constraint studied in Section 6.2 is k-set system for a (relatively) modest value of k. It is straightforward to show that a single knapsack constraint is a $\lceil c_{\max}/c_{\min} \rceil$-extendible system, and consequently a $\lceil c_{\max}/c_{\min} \rceil$-set system, where $c_{\max} = \max_{e\in\mathcal{N}} c(e)$ and $c_{\min} = \min_{e\in\mathcal{N}} c(e)$. We complement this with Lemmata 27 and 28, which prove that the planarity constraint is a 3-set system and that the intersection of a k_1-set system and a k_2-set system is a $(k_1 + k_2)$-set system, respectively. We would like to thank Chandra Chekuri for pointing Lemma 27 to us. We also would like to state that Lemma 28 is similar to well-known properties of more restricted classes of set systems, but we are not aware of a previously published explicit proof of it for general k-set systems.

Lemma 27. *For every graph $G = (V, E)$, the system $\mathcal{M} = (E, \mathcal{I})$, where $\mathcal{I} = \{S \subseteq E \mid (V, S)$ is a planar graph$\}$, is a 3-set system.*

Proof. Note that $\mathcal{M}$ is downward closed, because a subgraph of a planar graph is also planar. Furthermore, the empty set is always a member of $\mathcal{I}$ because a graph with no edges is planar. Thus, we concentrate on proving that $\mathcal{M}$ obeys the remaining property of k-set systems. Formally, for an arbitrary set $E' \subseteq E$, and two arbitrary bases B_1 and B_2 of E' (i.e., subsets of E' which are independent, and no other edge of E' can be added to them without violating independence), we need to show $|B_1|/|B_2| \leq 3$.

Assume first, for simplicity, that (V, E') is connected. This implies that (V, B_2) is also connected (otherwise, we can add to it any edge connecting two different connected components without violating planarity, which contradicts the fact that it is a base of E'), and thus, the size of B_2 must be at least $|V| - 1$. Additionally, it is well-known that, using Euler's formula, it is possible to show that the number of edges in a planar graph is at most $3|V| - 6$ as long as $|V| \geq 3$. Thus, $|B_1| \leq 3|V| - 6$ as long as $|V| \geq 3$. For $|V| < 3$, we still get $|B_1| \leq 3|V| - 3$ because a graph with a single vertex can include no edges and a graph with two vertices can include at most a single edge. Combining all these observations, we now get

$$\frac{|B_1|}{|B_2|} \leq \frac{3|V| - 3}{|V| - 1} = 3 \enspace .$$

Consider now the case in which (V, E') has more than one connected component. In this case we can use the above argument for each component of (V, E'). Thus, if we denote by m the number of connected components of this graph, where V_i is the set of vertices of the i-th component, then we get

$$\frac{|B_1|}{|B_2|} \leq \frac{\sum_{i=1}^{m} 3|V_i| - 3}{\sum_{i=1}^{m} |V_i| - 1} = 3 \enspace . \qquad \square$$

Lemma 28. *Let $\mathcal{M}_1 = (\mathcal{N}, \mathcal{I}_1)$ and $\mathcal{M}_2 = (\mathcal{N}, \mathcal{I}_2)$ be a k_1-set system and a k_2-set system, respectively, over the same ground set $\mathcal{N}$. Then, the set system $\mathcal{M} = (\mathcal{N}, \mathcal{I}_1 \cap \mathcal{I}_2)$ is a $(k_1 + k_2)$-set system.*

Proof. For every subset $F \subseteq \mathcal{N}$, let $\mathcal{B}_{\mathcal{M}}(F)$ be the set of bases of F with respect to $\mathcal{M}$. It is clear that $\mathcal{I}_1 \cap \mathcal{I}_2$ is down-monotone and contains the empty set. Thus, to prove the lemma we only need to prove that for every subset $F \subseteq \mathcal{N}$

$$\frac{\max_{B \in \mathcal{B}_{\mathcal{M}}(F)} |B|}{\min_{B \in \mathcal{B}_{\mathcal{M}}(F)} |B|} \leq k_1 + k_2 \enspace . \tag{8}$$

Towards this goal, let us define $B_\ell = \arg\max_{B \in \mathcal{B}_{\mathcal{M}}(F)} |B|$ and $B_s = \arg\min_{B \in \mathcal{B}_{\mathcal{M}}(F)} |B|$. For every $i \in \{1, 2\}$, let D_i be the set of elements of B_ℓ that do not belong to B_s and cannot be added to B_s without violating independence with respect to $\mathcal{M}_i$. Formally, $D_i = \{u \in B_\ell \setminus B_s \mid B_s + u \notin \mathcal{I}_i\}$. Since B_s is a base of $\mathcal{M}$, every element of $B_\ell \setminus B_s$ must belong either to D_1 or to D_2. Hence, we get

$$|D_1| + |D_2| \geq |B_\ell \setminus B_s| \enspace .$$

Observe now that for every $i \in \{1, 2\}$ the set $U_i = D_i \cup (B_\ell \cap B_s)$ is a subset of B_ℓ, and thus, independent with respect to $\mathcal{M}_i$. Moreover, the definition of D_i implies that B_s is a base of $U_i \cup B_s$ with respect to $\mathcal{M}_i$. Since $\mathcal{M}_i$ is a p_i-set system, this implies that the size of the independent set U_i is upper bounded by $p_i \cdot |B_s|$. Thus, we get

$$|D_i| + |B_\ell \cap B_s| = |U_i| \leq p_i \cdot |B_s| \quad \forall\, i \in \{1, 2\} \ .$$

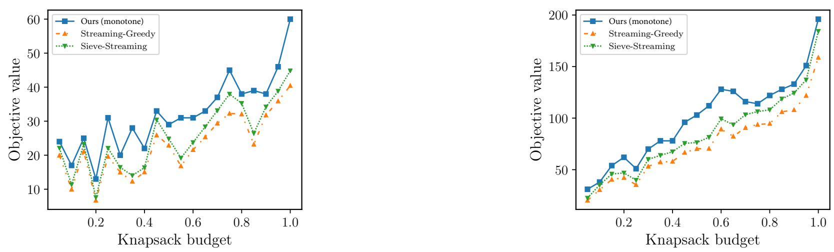
Combining the above inequalities, we get

$$|B_\ell| \leq |D_1| + |D_2| + |B_\ell \cap B_s| \leq p_1 \cdot |B_s| + p_2 \cdot |B_s| - |B_\ell \cap B_s| \leq (p_1 + p_2) \cdot |B_s| \ ,$$

which proves Inequality (8) due to the definitions of B_s and B_ℓ. □

E.2. Planarity with Knapsack

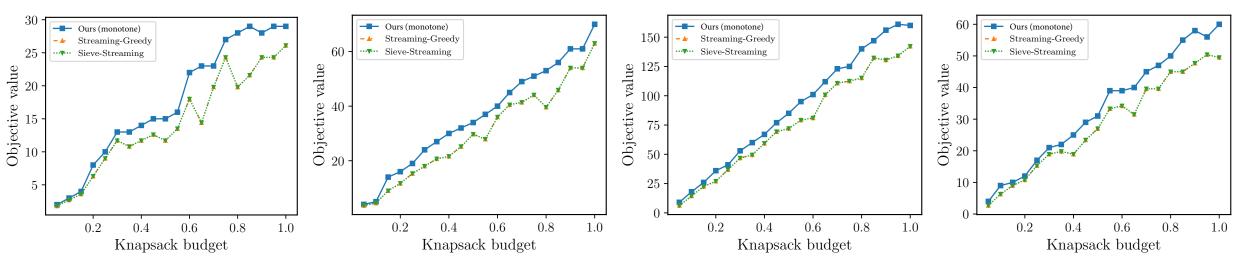
We begin this section with an experiment studying two other real-world networks from (Leskovec & Krevl, 2014) with exactly the same setting of Section 6.2. One can observe in Fig. 6 that our algorithm outperforms the two other baselines.



(a) Social graph (b) Wiki vote network

Figure 6. Planarity with knapsack (linear objective function). The weight of each edge is set to one. Knapsack cost of each edge $e = (u, v)$ is proportional to $\max(1, d_u - 6)$, where d_u is the degree of node u in graph G. The costs are normalized so that $\sum_{e \in E} c_e = |V|$, where c_e represents the knapsack cost of edge e.

In Fig. 7, we compare the performance of our streaming algorithm with the performance of Streaming Greedy and Sieve Streaming under a different knapsack constraint. Here, the cost of each edge $e = (u, v)$ is proportional to an integer picked uniformly at random from set the $\{1, 2, 3, 4, 5\}$. The costs are normalized so that $\sum_{e \in E} c_e = |V|$, where c_e represents the knapsack cost of edge e. Like in the case of the previous knapsack constraint, we observe that our streaming algorithm returns solutions with higher objective values for various knapsack budgets.



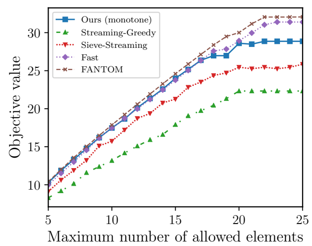
(a) Social graph (b) EU Email (c) Facebook ego network (d) Wiki vote network

Figure 7. Planarity with knapsack (linear objective function). The weight of each edge is set to one. Knapsack cost of each edge $e = (u, v)$ is proportional to an integer picked uniformly at random from set the $\{1, 2, 3, 4, 5\}$. The costs are normalized such that $\sum_{e \in E} c_e = |V|$, where c_e represents the knapsack cost of edge e. We note that it is difficult to view the orange line in the above figures since it is mostly hidden behind the green line.

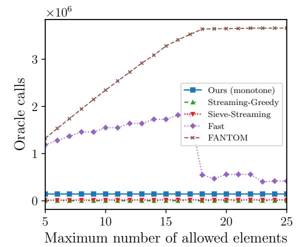
E.3. Movie Recommendation with a Monotone Submodular Function

In this section we describe the part of the experiment using a non-negative, monotone and submodular objective function. Let us begin the section by describing this function. Assume v_i represents the feature vector of the i-th movie, then we define a matrix M such that $M_{ij} = e^{-\lambda \cdot \text{dist}(v_i, v_j)}$, where $\text{dist}(v_i, v_j)$ is the euclidean distance between vectors v_i, v_j—informally M_{ij} encodes the similarity between the frames represented by v_i and v_j. The diversity of a set S of movies is measured by the non-negative monotone submodular objective $f(S) = \log\det(\mathbf{I} + \alpha M_S)$, where $\mathbf{I}$ is the identity matrix, α is a positive scalar and M_S is the principal sub-matrix of M indexed by S (Herbrich et al., 2003).

In the experiment we did with the above objective function, we set the genre limit to 10, λ to 0.1, and α to 20. The results of the experiment appear in Figs. 8a and 8b. In Fig. 8a, we can observe that our streaming algorithm outperforms streaming greedy and sieve streaming. Furthermore, while our algorithm requires only a single pass over the data and enjoys a very low computational complexity (see Fig. 8b), the solutions it returns are competitive with respect to the solutions produced by the offline algorithms we compare with.



(a) Monotone function



(b) Monotone function

Figure 8. Movie recommendation with a monotone submodular function subject to two knapsacks constraints.

E.4. Yelp Location Data Summarization

In this application, given thousands of business locations with several related attributes, our objective is to find a good summary of the locations from the following six cities: Charlotte, Edinburgh, Las Vegas, Madison, Phoenix and Pittsburgh. We use the Yelp Academic dataset (Yelp, a), which is a subset of Yelp's businesses, reviews, and user data (Yelp, b). The dataset contains information about local businesses across 11 metropolitan areas, and we consider only locations in six out of these metropolitan areas. We used the description of each business location and reviews for feature extraction. These features contain information regarding many attributes such as having vegan menus, delivery options, the possibility of outdoor seating, being good for groups, etc.[9]

Suppose we want to select, out of a ground set $\mathcal{N} = \{1, \ldots, n\}$, a subset of locations which provides a good representation of all the existing business locations. Towards this goal, we calculate a matrix M representing the similarity between every two locations $i, j \in \mathcal{N}$ using the same method described in Appendix E.3. Then, intuitively, given a set S, each location $i \in \mathcal{N}$ is represented by the location from the set S with the highest similarity to i. Thus, it is natural to define the total utility provided by a set S using the following non-negative, monotone and submodular set function (Krause & Golovin, 2012; Frieze, 1974):

$$f(S) = \frac{1}{n} \sum_{i=1}^{n} \max_{j \in S} M_{i,j} \quad . \tag{9}$$

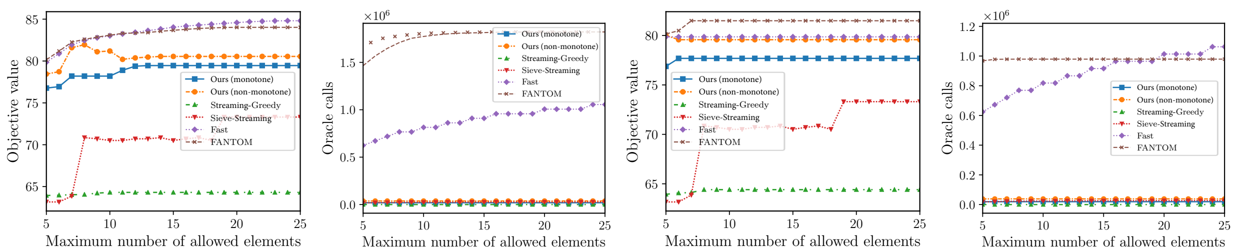
Note that the utility function (9) depends on the entire dataset $\mathcal{N}$. In the streaming setting we do not have access to the full data stream, but fortunately, our objective function is additively decomposable (Mirzasoleiman et al., 2013) over the ground set $\mathcal{N}$. Thus, as long as we can sample uniformly at random from a data stream, it is possible to estimate (9) arbitrarily close to its exact value (Badanidiyuru et al., 2014, Proposition 6.1). To sample randomly from the data stream and estimate the function, we use the reservoir sampling technique explained in (Badanidiyuru et al., 2014, Algortithm 4).

For the constraint, we use a combination of matroid and knapsack constraints (which yields a k-extendible constraint). The matroid constraint is as follows: i) there is a limit m on the total number of selected locations and ii) the maximum number of allowed locations from each of the six cities is 10. For the knapsack constraints we consider two different scenarios: i) in the first scenario, there is a single knapsack c_1 in which the cost assigned to each location is proportional to the distance of that location from a pre-specified location in the down-town of its metropolitan area. ii) in the second scenario, we add another knapsack c_2 which is based on the distance between each location and the international airport serving its

[9]For the feature extraction, we used the script provided at `https://github.com/vc1492a/Yelp-Challenge-Dataset`.

metropolitan area. In this set of experiments, we set the knapsack budgets to 1, where one unit of budget is equivalent to 100km. This means that we allow the sum of the distances of every feasible set of locations to the points of interest (i.e., down-towns or airports) to be at most 100km.

In our experiments, we compare the utility and computational cost of algorithms for different values of m (the upper limit on the number of locations in the produced summary). From the experiments (see Fig. 9), we observe that i) our proposed algorithm, consistently, demonstrates a better performance compared to other streaming algorithms in terms of the utility of the final solution, and ii) the utilities of the solutions produced by our algorithm are comparable to the utility of solutions produced by state-of-the-art offline algorithms, despite the ability of our algorithm to make only a single pass over the data and its several orders of magnitude better computational complexity. We also observe that, as expected, adding more constraints (compare Figs. 9a and 9c) reduces the utility of the selected summary.



(a) One knapsack (b) One Knapsack (c) Two Knapsacks (d) Two Knapsacks

Figure 9. Yelp Location Data Summarization

E.5. Twitter Summarization

There are several news reporting Twitter accounts with millions of followers. One interesting data summarization task is to provide a periodic synopsis of major events from the news feeds of these accounts. While finding an objective function to quantify the utility of a summary is a delicate task, the need to provide the summary in real-time for streams of data which are arriving at a fast pace makes the data summarization task even harder. In this application, our goal is to generate real-time summaries for Twitter feeds of several news agencies with the following Twitter accounts (also known as "handles"): @CNNBrk, @BBCSport, @WSJ, @BuzzfeedNews, @nytimes, @espn.

For this application, we use the twitter dataset provided in (Kazemi et al., 2019). In order to cover the important events of the day without redundancy, we use a monotone and submodular function f that encourages diversity in the selected set of tweets (Kazemi et al., 2019). Let us explain this function. The function f is defined over a ground set $\mathcal{N}$ of tweets. Assume that each tweet $u \in \mathcal{N}$ consists of a non-negative value val_u representing the number of retweets it has received and a set of ℓ_u keywords $W_u = \{w_{u,1}, \cdots, w_{u,\ell_u}\}$ from the set of all possible keywords $\mathcal{W}$. The score of a word $w \in \mathcal{W}$ for a given tweet u is defined by

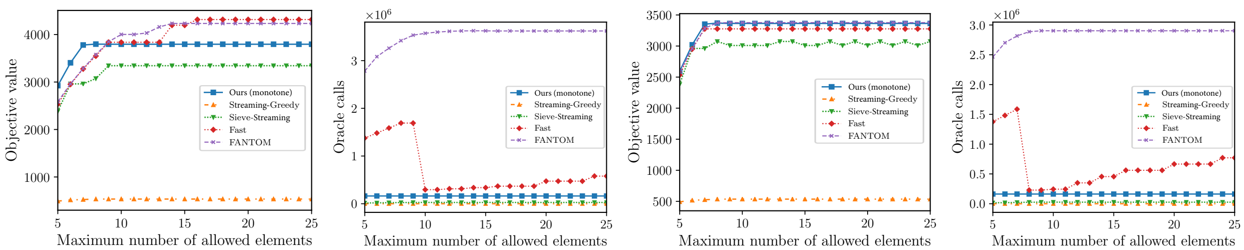
$$\text{score}(w, u) = \begin{cases} \text{val}_u & \text{if } w \in W_u \ , \\ 0 & \text{otherwise} \ , \end{cases}$$

and the function f is defined by

$$f(S) = \sum_{w \in \mathcal{W}} \sqrt{\sum_{u \in S} \text{score}(w, u)} \ .$$

Like in Appendix E.4, each one of our experiments involves a matroid constraint plus one or two knapsack constraints, which yields a k-extendible system constraint. The matroid constraint allows at most five tweets from each one of the six twitter accounts and at most m tweets from all the accounts together. In the first knapsack constraint c_1, which is a constraint that is used in all the experiments of this section, the cost of each tweet is proportional to the absolute time difference (in months) between the tweet and the first of January 2019. In other words, we are more interested in tweets that are closer to the first day of the year 2019. We also have a second knapsack constraint c_2, which is used only in our second experiment. In this constraint, the cost of each element is proportional to the length (number of keywords) of the corresponding tweet, which enables us to provide shorter summaries. We normalize the knapsack costs such that each unit of knapsack budget is equivalent to roughly 10 months for c_1 and 26 keywords for c_2, respectively. Then, we set the budgets of both knapsacks to 1.

In Figs. 10a and 10b, we observe the outcomes of different algorithms for the scenario with a single knapsack. It is evident that the utility of solutions returned by our proposed streaming algorithm exceeds the other baseline streaming algorithm. It is also interesting to point out that, for the case with two knapsack constraint, our streaming algorithms outperform even the Fast algorithm, which is one of the offline algorithms (see Fig. 10c).



(a) One knapsack (b) One Knapsack (c) Two Knapsacks (d) Two Knapsacks

Figure 10. Twitter Data Summarization: The maximum number of allowed tweets from each news agency in the summary is five.