Almost Optimal Semi-streaming Maximization for k-Extendible Systems

Ran Haba 16-206874
With supervision of Prof. Moran Feldman

Dept. of Mathematics and Computer Science, Open University of Israel
November 13, 2019

Contents

List of Figures

Abstract

In this thesis we consider the problem of finding a maximum weight set subject to a k-extendible constraint in the data stream model. The only non-trivial algorithm known for this problem to date—to the best of our knowledge—is a semi-streaming $k^2(1+\varepsilon)$-approximation algorithm (Crouch and Stubbs, 2014), but semi-streaming $O(k)$-approximation algorithms are known for many restricted cases of this general problem. In this thesis, we close most of this gap by presenting a semi-streaming $O(k \log k)$-approximation algorithm for the general problem, which is almost the best possible even in the offline setting (Feldman et al., 2017).

1 Introduction

Many problems in combinatorial optimization can be cast as special cases of the following general task. Given a ground set $\mathcal{N}$ of weighted elements, find a maximum weight subset of $\mathcal{N}$ obeying some constraint $\mathcal{C}$. In general, one cannot get any reasonable approximation ratio for this general task since it captures many hard problems such as maximum independent set in graphs. However, the existing literature includes many interesting classes of constraints for which the above task becomes more tractable. In particular, in the 1970's Jenkyns [15] and Korte and Hausmann [16] suggested, independently, a class of constraints named *k-set system* constraints which represents a sweet spot between generality and tractability. On the one hand, finding a maximum weight set subject to a k-set system constraint captures many well known problems such as matching in hypergraphs, matroid intersection and asymmetric travelling salesperson. On the other hand, k-set system constraints have enough structure to allow a simple greedy algorithm to find a maximum weight set subject to such a constraint up to an approximation ratio of k.[1]

The k-approximation obtained by the greedy algorithm for finding a maximum weight set subject to a k-set system constraint was recently shown to be the best possible [2]. Nevertheless, over the years many works improved over it either by achieving a better guarantee for more restricted classes of constraints [10, 18, 19], or by extending the guarantee to more general objectives (such as maximizing a submodular function) [8, 10, 11, 12, 17, 18, 21, 24]. Unfortunately, many of the above mentioned improvements are based on quite slow algorithms. Moreover, as modern applications require the processing of increasingly large amounts of data, even the simple greedy algorithm is often viewed these days as too slow for practical use. This state of affairs has motivated recent works aiming to study the problem of finding a maximum weight set subject to a k-set system constraint in a Big Data oriented setting such as Map-Reduce and the data stream model. For the Map-Reduce setting, Ponte Barbosa et al. [7] essentially solved this problem by presenting a $(k+O(\varepsilon))$-approximation Map-Reduce algorithm for it using $O(1/\varepsilon)$ rounds, which almost matches the optimal approximation ratio in the sequential setting. In contrast, the situation for the data stream model is currently much more involved.

The only non-trivial data stream algorithm known to date (as far as we know) for finding a maximum weight set subject to a general k-set system constraint is a $k^2(1+\varepsilon)$-approximation semi-streaming algorithm by Crouch and Stubbs [6]. As one can observe, there is a large gap between the last approximation ratio and the k-approximation that can be achieved in the offline setting. Several works partially addressed this gap by providing an $O(k)$-approximation semi-streaming algorithms for more restricted classes of constraints, the most general of which is known as k-

[1] k is a parameter of the constraint which intuitively captures its complexity. The exact definition of k is given in Section 2, but we note here that in many cases of interest k is quite small. For example, matroid intersection is a 2-set system.

matchoid constraints [4, 5, 9, 22]. However, these results cannot be considered a satisfactory solution for the gap because k-matchoid constraints are much less general than k-set system constraints.[2]

In this thesis we make a large step towards resolving the above gap. Specifically, we present an $\tilde{O}(k)$-approximation semi-streaming algorithm for finding a maximum weight set subject to a class of constraints, known as *k-extendible* constraints, that was introduced by [20] and captures (to the best of our knowledge) all the special cases of k-set system constraints studied in the literature to date (including, in particular, k-matchoid constraints). Formally, we prove the following theorem.

Theorem 1.1. *There is a polynomial time semi-streaming algorithm achieving $O(k \log k)$-approximation for the problem of finding a maximum weight set subject to a k-extendible constraint. Assuming it takes constant space to store a single element and a single weight, the space complexity of the algorithm is $O(\rho(\log k + \log \rho))$, where ρ is the maximum size of a feasible set according to the constraint.*

As the class of k-extendible constraints captures every other restricted class of k-set system constraints from the literature, we believe Theorem 1.1 represents the final intermediate step before closing the above mentioned gap completely (*i.e.*, either finding an $\tilde{O}(k)$ semi-streaming algorithm for k-set system constraints, or proving that this cannot be done). It should also be mentioned that the approximation ratio guaranteed by Theorem 1.1 is optimal up to an $O(\log k)$ factor since it is known that one cannot achieve better than k-approximation for finding a maximum weight set subject to a k-extendible constraint even in the offline setting [8].

1.1 Additional Related Work

In the k-dimensional matching problem, one is given a weighted hypergraph in which the vertices are partitioned into k subsets, and every edge contains exactly one vertex from each one of these subsets. The objective in this problem is to find a maximum weight matching in the hypergraph. Hazan et al. [13] showed that no algorithm can achieve a better than $\Omega(k/\log k)$-approximation for k-dimension matching unless $\mathtt{P} = \mathtt{NP}$. Interestingly, it turns out that k-dimensional matching is captured by all the standard restricted cases of the the problem of finding a maximum weight set subject to k-set system constraint, and thus, the inapproximability of Hazan et al. [13] extends to them as well. For most of these restricted cases this is the strongest inapproximability known, although a tight inapproximability of k was proved for k-set system and k-extendible constraints by [2] and [8], respectively.

Complementing the hardness result of [13], some works presented algorithmic results for either k-dimensional matching or natural generalizations of it such as k-set packing [3, 14, 23].

2 Preliminaries and Notation

In this section we formally define some of the terms used in Section 1 and the notation that we use in the rest of this thesis. Given a finite ground set $\mathcal{N}$, an *independence system* over this ground set is a pair $(\mathcal{N}, \mathcal{I})$ in which $\mathcal{I}$ is a non-empty collection of subsets of $\mathcal{N}$ (formally, $\varnothing \neq \mathcal{I} \subseteq 2^{\mathcal{N}}$) which is *down-closed* (*i.e.*, if T is a set in $\mathcal{I}$ and S is a subset of T, then S also belongs to $\mathcal{I}$). One easy way to get an example of an independence system is to take an arbitrary vector space W, designate the set of vectors in this space as the ground set $\mathcal{N}$, and make $\mathcal{I}$ the collection of all independent

[2] We do not formally define k-matchoid constraints in this thesis, but it should be noted that they usually fail to capture knapsack like constraints. For example, a single knapsack constraint in which the ratio between the largest and smallest item sizes is at most k is a k-set system constraint, but usually not a k-matchoid constraint.

sets of vectors in W. Since removing a vector from an independent set of vectors cannot make the set dependent, the pair $(\mathcal{N}, \mathcal{I})$ obtained from W in this way is indeed an independence system.

The above example for getting an independence system from a vector space was one of the original motivations for the study of independence systems, and thus, a lot of the terminology used for independence systems is borrowed from the world of vector spaces. In particular, a set is called *independent* in a given independence system $(\mathcal{N}, \mathcal{I})$ if and only if it belongs to $\mathcal{I}$, and it is called a *base* of the independence system if it is an inclusion-wise maximal independent set. Using this terminology, we can now define k-set systems.

Definition 2.1. *An independence system $(\mathcal{N}, \mathcal{I})$ is a k-set system for an integer $k \geq 1$ if for every set $S \subseteq \mathcal{N}$, all the bases of $(S, 2^S \cap \mathcal{I})$ 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 $(S, 2^S \cap \mathcal{I})$ 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 independent set in a given k-set system $(\mathcal{N}, \mathcal{I})$ by outputting an arbitrary base of the k-set system, which can be done using the following simple strategy, which we call the *unweighted greedy algorithm*. 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 (*i.e.*, not independent).

A *k-set system constraint* is a constraint defined by a k-set system, and a set S obeys this constraint if and only if it is independent in that k-set system. Note that using this notion we can refer to the problem studied in the previous paragraph as finding a maximum cardinality set subject to a k-set system constraint. More generally, given a weight function $w \colon \mathcal{N} \to \mathbb{R}_{\geq 0}$ and a k-set system $(\mathcal{N}, \mathcal{I})$ over the same ground set, it is often useful to consider the problem of finding a maximum weight set $S \subseteq \mathcal{N}$ subject to the constraint corresponding to this k-set system (the weight of a set S is defined as $\sum_{u \in S} w(u)$). Jenkyns [15] and Korte and Hausmann [16] showed that one can get a k-approximation for this problem using an algorithm, known simply as the *greedy algorithm*, which is a variant of the unweighted greedy algorithm that considers the elements of $\mathcal{N}$ in a non-decreasing weight order.

The definition of k-set systems is very general, which occasionally does not allow them to capture all the necessary structure of a given application. Thus, various stronger kinds of independent set systems have been considered over the years, the most well known of which is the intersection of k matroids (which is equivalent to a k-set system for $k = 1$, and represents a strictly smaller class of independence systems for larger values of k). In this work we consider another kind of independence systems, which was originally defined by [20]. In this definition we use the expression $S + u$ to denote the union $S \cup \{u\}$. We use the plus sign in a similar way throughout the rest of this thesis.

Definition 2.2. *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 $T \setminus Y + u \in \mathcal{I}$.*

The class of k-extendible systems is general enough to capture the intersection of k matroids and every other restricted class of k-set systems from the literature that we are aware of. In contrast, it is not difficult to verify that any k-extendible system is a k-set system (we refer the reader to Appendix A for more information about the hierarchy of these constraints classes and a few others). Thus, the greedy algorithm provides k-approximation for the problem of finding a maximum weight set subject to a k-extendible constraint—*i.e.*, a constraint defined by a k-extendible system and allowing only sets that are independent in this system.

In the data stream model version of the above problem, the elements of the ground set of a k-extendible system $(\mathcal{N}, \mathcal{I})$ arrive one after the other in an adversarially chosen order. An algorithm for this model views the elements of $\mathcal{N}$ as they arrive, and it gets to know the weight $w(u)$ of every element u upon its arrival. Additionally, as is standard in the field, we assume the algorithm has access to an *independence oracle* that given a set $S \subseteq \mathcal{N}$ answers whether S is independent. The objective of the algorithm is to output a maximum weight independent set of the k-extendible system. If the algorithm is allowed enough memory to store the entire input, then the data stream model version becomes equivalent to the offline version of the problem. Thus, an algorithm for this model is interesting only if it has a low space complexity. Since any algorithm for this model must use at least the space necessary for storing its output, most works on this model look for *semi-streaming* algorithms, which are data stream algorithms whose space complexity is upper bounded by $O(\rho \cdot \operatorname{polylog} n)$—where ρ is the maximum size of an independent set and n is the size of the ground set. In particular, we note that the space complexity guaranteed by Theorem 1.1 falls within this regime because $\rho \leq n$ by definition, and one can assume that $k \leq n$ because any independence system is n-extendible.

One can observe that the unweighted greedy algorithm (unlike the greedy algorithm itself) can be implemented as a semi-streaming algorithm because it considers the elements in an arbitrary order. This observation is crucial for our result since the algorithm we develop is heavily based on using the unweighted greedy algorithm as a subroutine (a similar use of the unweighted greedy algorithm is done by the current state-of-the-art algorithm for the problem due to Crouch and Stubbs [6]).

Thesis Organization: In Section 3 we present a reduction that allows us to assume that the weights of the elements are powers of k, at the cost of losing a factor of $O(\log k)$ in the space complexity of the algorithm. Using this reduction, we present a basic version of our algorithm in Section 4. This basic version presents our main new ideas, but achieves semi-streaming space complexity only under the simplifying assumption that the ratio between the maximum and minimum element weights is polynomially bounded. This simplifying assumption can be dropped using standard techniques, and we defer the details to Appendix B.

3 Reduction to k-Power Weights

In this section we present a reduction that allows us to assume that the weights of all the elements in the ground set $\mathcal{N}$ are powers of k. This reduction simplifies the algorithms we present later in this thesis. However, before presenting the reduction itself, let us note that we assume from this point on that $k = 2^i$ for some integer $i \geq 1$. This assumption is without loss of generality because if k does not obey it, then we can increase its value to the nearest integer that does obey it. Since the new value of k is larger than the old value by at most a factor of 2, the approximation ratio guaranteed for both values of k by Theorem 1.1 is asymptotically equal.

We say that an instance of the problem of finding a maximum weight set subject to a k-extendible constraint is a *k-power* instance if the weights of all the elements in it are powers of k.

Reduction 3.1. *Assume that we are given a polynomial time data stream algorithm ALG for the problem of finding a maximum weight set subject to a k-extendible constraint. If ALG provides α-approximation for k-power instances of the problem using S_{ALG} space, then there exists a polynomial time data stream algorithm for the same problem which achieves $O(\alpha \log k)$-approximation for arbitrary instances using $O(S_{ALG} \cdot \log k)$ space. Moreover, if the weights of all the elements fall*

within some range $[w_{\min}, w_{\max}]$*, then it suffices for ALG to provide* α*-approximation for* k*-power instances in which all the weights fall within the range* $[w_{\min}/k, w_{\max}]$*.*

Before presenting the algorithm we use to prove the above reduction, we need to define some additional notation. Let $\ell \triangleq \log_2 k$, and note that ℓ is a positive integer because we assume that k is at least 2 and a power of 2. For every element $u \in \mathcal{N}$ of weight $w(u)$, we define an auxiliary weight $w_2(u) \triangleq k^{\lfloor \log_k w(u) \rfloor}$. Intuitively, $w_2(u)$ is the highest power of k which is not larger than $w(u)$. The following observation formally states the properties of w_2 that we need. In this observation we use the notation $i(u) \triangleq \lfloor \log_2 w(u) \rfloor$.

Observation 3.2. *For every element* $u \in \mathcal{N}$*,* $w_2(u)$ *is a power of* k *obeying* $w(u)/2 \leq w_2(u) \cdot 2^{i(u) \bmod \ell} \leq w(u)$ *and* $w(u)/k \leq w_2(u) \leq w(u)$*.*

Proof. The first part of the observation, namely that $w_2(u)$ is a power of k, follows immediately from the definition of w_2. Thus, we concentrate here on proving the other parts of the observation.

Note that

$$w_2(u) = k^{\lfloor \log_k w(u) \rfloor} = k^{\lfloor \ell^{-1} \cdot \log_2 w(u) \rfloor} = k^{\ell^{-1} \cdot \{\lfloor \log_2 w(u) \rfloor - \lfloor \log_2 w(u) \rfloor \bmod \ell\}} = k^{\ell^{-1} \cdot \lfloor \log_2 w(u) \rfloor} / 2^{i(u) \bmod \ell} \ .$$

Rearranging the last equality, we get

$$\frac{w(u)}{2} = k^{\log_k w(u) - \log_k 2} = k^{\ell^{-1} \log_2 w(u) - \ell^{-1}} \leq k^{\ell^{-1} \cdot \lfloor \log_2 w(u) \rfloor} = w_2(u) \cdot 2^{i(u) \bmod \ell} \ ,$$

and

$$w_2(u) \cdot 2^{i(u) \bmod \ell} = k^{\ell^{-1} \cdot \lfloor \log_2 w(u) \rfloor} \leq k^{\ell^{-1} \cdot \log_2 w(u)} = k^{\log_k w(u)} = w(u) \ .$$

To complete the proof of the observation, we note that it also holds that

$$w_2(u) = k^{\lfloor \log_k w(u) \rfloor} \leq k^{\log_k w(u)} = w(u) \quad \text{and} \quad w_2(u) = k^{\lfloor \log_k w(u) \rfloor} \geq k^{\log_k w(u) - 1} = \frac{w(u)}{k} \ . \quad \square$$

We are now ready to present the algorithm that we use to prove Reduction 3.1, which appears as Algorithm 1. To intuitively understand this algorithm, it is useful to think of $i(u)$ as the "class" element u belongs to. All the elements within class i have weights between 2^i and 2^{i+1}, and thus, treating them all as having the weight 2^i does not affect the approximation ratio by more than a factor of 2. Let us call 2^i the *characteristic* weight of class i. Note now that the ratio between the characteristic weight of class i_1 and the characteristic weight of class i_2 is $2^{i_1 - i_2}$, which is a power of k whenever $i_1 - i_2$ is an integer multiple of $\ell = \log_2 k$. Thus, one can group the classes into ℓ groups such that the ratio between the characteristic weights of any pair of classes within a group is a power of k (see Figure 1 for a graphial illustration of these groups). Moreover, by multiplying all the characteristic weights in the group by an appropriate scaling factor, one can make them all powers of k. This means that for every group there exists a transformation that converts all the weights of the elements in it to powers of k and preserves the ratio between any two weights in the group up to a factor of 2. In particular, we get that the elements of the group after the transformation form a k-power instance.

Adding up all the above, we have described a way to transform any instance of finding a maximum weight independent set subject to a k-extendible constraint into ℓ new instances of this problem that are guaranteed to be k-power. Algorithm 1 essentially creates these ℓ new instances on the fly, and feeds them to ℓ copies of the algorithm ALG whose existence is assumed in Reduction 3.1. Given this point of view, $i(u) \bmod \ell$ should be understood as the group to which element u belongs, and $w_2(u)$ is the transformed weight of u. Observation 3.2 can now be

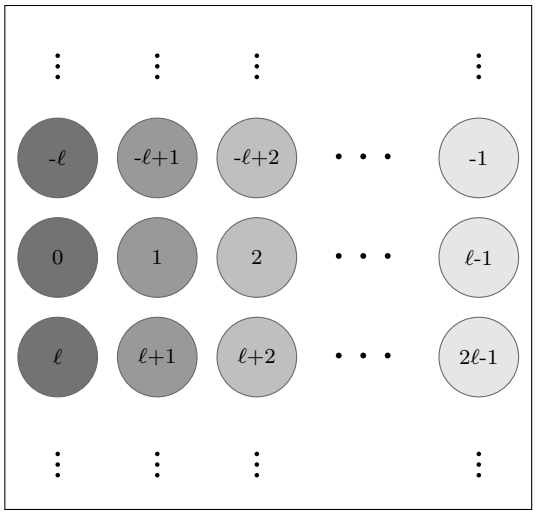

Figure 1: Each circle in this drawing represent a class, and the value $i(u)$ of the elements in this class appears in the center the circle. The classes are grouped according to the columns in the drawing. We note that element u belonging to group j has weight within the range $[2^{k\ell+j}, 2^{k\ell+j+1})$, where k is an integer and $i(u) = k\ell + j$.

Algorithm 1: Modulo ℓ Split

Create ℓ instances of ALG named $ALG_0, ALG_1, \ldots, ALG_{\ell-1}$.
for *each element u that arrives from the stream* **do**
    Calculate $i(u)$ and $w_2(u)$ as defined above.
    Feed u to $ALG_{(i(u) \bmod \ell)}$ with the weight $w_2(u)$.
Let C_i denote the output of ALG_i for every $0 \leq i \leq \ell - 1$.
return *the best solution among $C_0, C_1, \ldots, C_{\ell-1}$.*

interpreted as stating that the ratio between the weights of elements belonging to the same group (and thus, having the same $i(u)$ mod ℓ value) is indeed changed by the transformation by at most a factor of 2.

In the rest of this section, we use B_i to denote the set of elements fed to instance ALG_i by Algorithm 1, and T to denote the output of Algorithm 1. Additionally, we denote by OPT an arbitrary (fixed) optimal solution for the original instance recieved by Algorithm 1. The following lemma proves that Algorithm 1 has the approximation ratio guaranteed by Reduction 3.1.

Lemma 3.3. $w(OPT) \leq O(\alpha \log k) \cdot w(T)$.

Proof. Since Algorithm 1 feeds every arriving element into exactly one of the instances ALG_0, $ALG_1, \ldots, ALG_{\ell-1}$, the sets $B_0, B_1, \ldots, B_{\ell-1}$ form a disjoint partition of $\mathcal{N}$. Thus,

$$w(OPT) = \sum_{i=0}^{\ell-1} w(B_i \cap OPT) \ .$$

Hence, by an averaging argument, there must exist an index i such that $w(OPT) \leq \ell \cdot w(OPT \cap B_i)$.

We now note that it follows from the pseudocode of Algorithm 1 and Observation 3.2 that the copies of ALG get only weights that are powers of k, and moreover, these weights belong to the

range $[w_{\min}/k, w_{\max}]$ whenever the original weights received by Algorithm 1 belong to the range $[w_{\min}, w_{\max}]$. Thus, by the assumption of Reduction 3.1, ALG_i achieves α-approximation for the instance it faces. Since $B_i \cap OPT$ is a feasible solution within this instance and C_i is the output of ALG_i, we get $w_2(OPT \cap B_i) \leq \alpha \cdot w_2(C_i)$. Therefore,

$$w(OPT) \leq \ell \cdot w(OPT \cap B_i) \leq 2\ell \cdot w_2(OPT \cap B_i) \cdot 2^i \leq 2\ell\alpha \cdot w_2(C_i) \cdot 2^i \leq 2\ell\alpha \cdot w(C_i) \leq 2\ell\alpha \cdot w(T) \ ,$$

where the second and penultimate inequalities hold by Observation 3.2, and the last inequality is due to the fact that T is the best solution among $C_0, C_1, \ldots, C_{\ell-1}$. □

The next lemma analyzes the space complexity of Algorithm 1 and completes the proof of Reduction 3.1.

Lemma 3.4. *Algorithm 1's space complexity is $O(S_{ALG} \cdot \log k)$.*

Proof. Algorithm 1 runs $\log k$ parallel copies of ALG, each of them is assumed (by Reduction 3.1) to use S_{ALG} space. Thus, the space required by these $\log k$ copies is $O(S_{ALG} \cdot \log k)$. In addition to this space, Algorithm 1 only requires enough space to do two things.

- The algorithm has to store the outputs of the copies of ALG. However, these outputs are originally stored by the copies themselves, and thus, storing them requires no more space than what is used by the copies.
- Calculate the sum of the weights of the elements in the solutions produced by the copies of ALG. Since we assume that the weight of an element can be stored in constant space, this requires again (up to constant factors) no more space than the space used by the copies of ALG to store their solutions. □

4 Algorithm

In this section we present a data stream algorithm for k-power instances of the problem of finding a maximum weight set subject to a k-extendible constraint. This algorithm assumes access to positive upper bound $w_{\max}$ and lower bound $w_{\min}$ on the weights of all the elements, and has a semi-streaming space complexity when the ratio between $w_{\max}$ and $w_{\min}$ is upper bounded by a polynomial in n. Proposition 4.1 states the properties that we prove for this algorithm more formally.

Proposition 4.1. *There exists a $2k$-approximation data stream algorithm for k-power instances of the problem of finding a maximum weight set subject to a k-extendible constraint. This algorithm assumes access to positive upper bound $w_{\max}$ and lower bound $w_{\min}$ on the weights of all the elements, and its space complexity is $O(\rho(\log(w_{\max}/w_{\min})/\log k + 1))$ under the assumption that constant space suffices to store a single element and a single weight.*

Before getting to the proof of Proposition 4.1, we note that together with Reduction 3.1 this proposition immediately implies the following corollary.

Corollary 4.2. *There exists an $O(k \log k)$-approximation data streaming algorithm for the problem of finding a maximum weight set subject to a k-extendible constraint. This algorithm assumes access to positive upper bound $w_{\max}$ and lower bound $w_{\min}$ on the weights of all the elements. The space complexity of this algorithm is $O(\rho(\log(w_{\max}/w_{\min}) + \log k))$ under the assumption that constant space suffices to store a single element and a single weight.*

Note that when the ratio between $w_{\max}$ and $w_{\min}$ is polynomial in n, the space complexity of the algorithm from Corollary 4.2 becomes $O(\rho \log n)$, and thus, the algorithm is semi-streaming. In Appendix B we explain how the algorithm can be modified so that it keeps the "effective" ratio $w_{\max}/w_{\min}$ on the order of $O(k^2\rho^2)$ even when no values $w_{\max}$ and $w_{\min}$ are supplied to the algorithm and the weights of the elements come from an arbitrary range. This leads to the space complexity of $O(\rho(\log k + \log \rho))$ stated in Theorem 1.1.

The rest of this section is devoted to the proof of Proposition 4.1. As a first step towards this goal, let us recall that the unweighted greedy algorithm is an algorithm that considers the elements of the ground set $\mathcal{N}$ in an arbitrary order, and adds every considered element to the solution it constructs if that does not violate independence. As mentioned above, it follows immediately from the definition of k-set systems that the unweighted greedy algorithm achieves an approximation ratio of k for the problem of finding a maximum cardinality independent set subject to a k-set system constraint. Since k-set systems generalize k-extendible systems, the same is true also for k-extendible constraints. The following lemma improves over this by showing a tighter guarantee for k-extendible constraints.

Lemma 4.3. *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 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. □

We are now ready to present the algorithm we use to prove Proposition 4.1, which is given as Algorithm 2. This algorithm has two main stages. In the first stage, the algorithm runs an independent copy of the unweighted greedy algorithm for every possible weight of elements. The copy corresponding to the weight k^i is denoted by $\texttt{Greedy}_i$ in the pseudocode of the algorithm, and Algorithm 2 feeds to it only the input elements whose weight is at least k^i. The output of $\texttt{Greedy}_i$ is denoted by C_i in the algorithm. We also denote in the analysis by E_i the set of elements fed to $\texttt{Greedy}_i$. By definition, C_i is obtained by running the unweighted greedy algorithm on the elements of E_i, which is a property we use below.

In the second stage of Algorithm 2 (which is done as a post-processing after the stream has ended), the algorithm constructs an output set T based on the outputs of the copies of the unweighted greedy algorithm. Specifically, this is done by running the unweighted greedy algorithm on the elements of $\bigcup_{i=i_{\min}}^{i_{\max}} C_i$, considering the elements of the sets C_i in a decreasing value of i order. While doing so, the given pseudocode also keeps in T_i the temporary solution obtained by

the unweighted greedy algorithm after considering only the elements of C_j for $j \geq i$. This temporary solution is used by the analysis below, but need not be kept by a real implementation of Algorithm 2.

Algorithm 2: Greedy of Greedies

Let $i_{\min} \leftarrow \lceil \log_k w_{\min} \rceil$ and $i_{\max} \leftarrow \lfloor \log_k w_{\max} \rfloor$.
Create $i_{\max} - i_{\min} + 1$ instances of the unweighted greedy algorithm named $\texttt{Greedy}_{i_{\min}}, \texttt{Greedy}_{i_{\min}+1}, \ldots, \texttt{Greedy}_{i_{\max}}$.
for *each element u that arrives from the stream* **do**
    Let $i_u \leftarrow \log_k w(u)$.
    Feed u to $\texttt{Greedy}_{i_{\min}}, \texttt{Greedy}_{i_{\min}+1}, \ldots, \texttt{Greedy}_{i_u}$.
Let C_i denote the output of $\texttt{Greedy}_i$ for every $i_{\min} \leq i \leq i_{\max}$.
Let $T \leftarrow \varnothing$.
for *every $i_{\min} \leq i \leq i_{\max}$ in descending order* **do**
    Greedily add elements from C_i to T as long as this is possible.
    Let T_i denote the current value of T.
return T.

We begin the analysis of Algorithm 2 by analyzing its space complexity.

Lemma 4.4. *Algorithm 2 can be implemented using a space complexity of* $O(\rho(\log(w_{\max}/w_{\min})/\log k + 1))$.

Proof. Note that each copy of the unweighted greedy algorithm only has to store its solution, which contains up to ρ elements since it is independent. Algorithm 2 uses $i_{\max} - i_{\min} + 1$ such copies, and thus, the space it needs for these copies is only

$$\rho(i_{\max} - i_{\min} + 1) \leq \rho\left(\log_k\left(\frac{w_{\max}}{w_{\min}}\right) + 1\right) = \rho \cdot O\left(\frac{\log(w_{\max}/w_{\min})}{\log k} + 1\right) \ .$$

In addition to the space used by the copies of the unweighted greedy algorithm, Algorithm 2 only needs to store the set T. This set contains a subset of the elements from the outputs of the above copies, and thus, can increases the space required only by a constant factor. □

To complete the proof of Proposition 4.1, it remains to analyze the approximation ratio of Algorithm 2. We begin with the following lemma, which is the technical heart of our analysis. Like in Section 3, let us denote by OPT be an arbitrary (fixed) optimal solution to the problem we want to solve. We also assume for consistency that $T_{i_{\max}+1} = \varnothing$ (note that $T_{i_{\max}+1}$ is not defined by Algorithm 2).

Lemma 4.5. *For each integer $i_{\min} \leq i \leq i_{\max}$, $k^2 \cdot |T_{i+1}| + k \cdot |T_i \setminus T_{i+1}| \geq |OPT \cap E_i|$.*

Proof. The set T_i can be viewed as the output of the unweighted greedy algorithm running on $\bigcup_{i \leq j \leq i_{\max}} C_j$. Since we also know that C_i is independent, Lemma 4.3 guarantees

$$k \cdot |T_i \setminus C_i| \geq |C_i \setminus T_i| \ .$$

Adding $k \cdot |C_i \cap T_i|$ to both its sides, we get

$$\begin{aligned} k \cdot |T_i| &\geq k \cdot |C_i \cap T_i| + |C_i \setminus T_i| = k \cdot |C_i \cap T_i| + \{|C_i| - |C_i \cap T_i|\} \\ &= (k-1) \cdot |C_i \cap T_i| + |C_i| \geq (k-1) \cdot |C_i \cap T_i| + k^{-1} \cdot |OPT \cap E_i| \ , \end{aligned}$$

where the last inequality holds since the unweighted greedy algorithm achieves k-approximation and $OPT \cap E_i$ is an independent set within E_i (recall that E_i is the set of elements that were fed to $\texttt{Greedy}_i$). Using the last inequality we can now get

$$\begin{aligned} k \cdot |T_i \setminus T_{i+1}| + k \cdot |T_{i+1}| = k \cdot |T_i| &\geq (k-1) \cdot |C_i \cap T_i| + k^{-1} \cdot |OPT \cap E_i| \\ &\geq (k-1) \cdot |T_i \setminus T_{i+1}| + k^{-1} \cdot |OPT \cap E_i| \ , \end{aligned}$$

where the first equality holds because $T_{i+1} \subseteq T_i$, and the second inequality holds because $T_i \setminus T_{i+1} \subseteq C_i \cap T_i$ (recall that the algorithm constructs T_i by adding elements of C_i to T_{i+1}). The lemma now follows by rearranging the above inequality and multiplying it by k. □

Using the last lemma, we can prove the existence of a useful mapping from the elements of OPT to the elements of T.

Lemma 4.6. *There exists a mapping $f \colon OPT \to T$ such that*

1. *for each $t \in T$, $|f^{-1}(t)| \leq k^2$.*
2. *for each $t \in T$, $|\{u \in f^{-1}(t) \mid w(u) = w(t)\}| \leq k$.*
3. *for each $u \in OPT$, $w(u) \leq w(f(u))$.*

Proof. We construct f by scanning the elements OPT and defining the mapping $f(e)$ for every element e scanned. To describe the order in which we scan the elements of OPT, let us define $P_i = OPT \cap (E_i \setminus E_{i-1})$. Note that $P_{i_{\min}}, P_{i_{\min}+1}, \ldots, P_{i_{\max}}$ is a disjoint partition of OPT, and thus, any scan of the elements of $P_{i_{\min}}, P_{i_{\min}+1}, \ldots, P_{i_{\max}}$ is a scan of the elements of OPT. Specifically, we scan the elements of OPT by first scanning the elements of $P_{i_{\max}}$ in an arbitrary order, then scanning the elements of $P_{i_{\max}-1}$ in an arbitrary order, and so on. Consider now the situation when our scan gets to an arbitrary element u of set P_i. One can note that prior to scanning u, we scanned (and mapped) only elements of $P_i \cup P_{i+1} \cup \cdots \cup P_{i_{\max}} = OPT \cap E_i$, and thus, we mapped at most $|OPT \cap E_i| - 1$ elements (the -1 is due to the fact that $u \in OPT \cap E_i$, and u was not mapped yet). Combining this with Lemma 4.5, we get that at the point in which we scan u there must still be either an element $t \in T_{i+1}$ that still has less than k^2 elements mapped to it or an element $t \in T_i \setminus T_{i+1}$ that still has less than k elements mapped to it. We choose the mapping $f(u)$ of u to be an arbitrary such element t.

Property 1 of the lemma is clearly satisfied by the above construction because we never map an element u to an element t that already has k^2 elements mapped to it. To see why Property 3 of the lemma also holds, note that every element $u \in P_i$ must have a weight of k^i by the definition of P_i. This element is mapped by f to some element $t \in T_{i+1} \cup (T_i \setminus T_{i+1}) = T_i \subseteq E_i$, and the weight of t is at least $k^i = w(u)$ by the definition of E_i. It remains to prove Property 2 of the lemma. Consider an arbitrary element $t \in T$ of weight k^i. The elements of OPT whose weight is k^i are exactly the elements of P_i, and thus, we need to show that $|f^{-1}(t) \cap P_i| \leq k$. Since all the elements of $T_{i+1} \subseteq C_{i+1} \cup C_{i+2} \cup \cdots \cup C_{i_{\max}} \subseteq E_{i+1}$ have weights of at least k^{i+1}, t cannot belong to T_{i+1}. Thus, an element of P_i can be mapped to t when scanned only if t has less than k elements already mapped to it (if $t \in T_i$) or not at all (if $t \notin T_i$), which implies that no more than k elements of P_i can get mapped to t, which is exactly what we wanted to prove. □

We are now ready to prove the approximation ratio of Algorithm 2 (and complete the proof of Proposition 4.1).

Lemma 4.7. *Algorithm 2 is a $2k$-approximation algorithm for k-power instances of the problem of finding a maximum weight set subject to a k-extendible constraint.*

Proof. Let f be the function whose existence is guaranteed by Lemma 4.6. The properties of this function imply that, for each element $t \in T$,

$$\sum_{u \in f^{-1}(t)} w(u) = \sum_{\substack{u \in f^{-1}(t) \\ w(u)=w(t)}} w(u) + \sum_{\substack{e \in f^{-1}(t) \\ w(u)<w(t)}} w(u) \leq k \cdot w(t) + (k^2 - k) \cdot \frac{w(t)}{k} \leq 2k \cdot w(t) \ .$$

Thus,

$$w(OPT) = \sum_{u \in OPT} w(u) = \sum_{t \in T} \sum_{u \in f^{-1}(t)} w(u) \leq \sum_{t \in T} [2k \cdot w(t)] = 2k \cdot w(T) \ ,$$

which completes the proof of the lemma. □

5 Conclusion

In this work we have presented the first semi-streaming $\tilde{O}(k)$-approximation algorithm for the problem of finding a maximum weight set subject to a k-extendible constraint. This result is intrinsically interesting because the generality of k-extendible constraints makes our algorithm applicable to many problems of interest. Additionally, we believe (as discussed in Section 1) that our result is likely to be the final intermediate step towards the goal of designing an algorithm with similar properties for general k-set system constraints or proving that this cannot be done.

Given our work, the immediate open question is to settle the approximation ratio that can be obtained for k-set system constraints in the data stream model. Another interesting research direction is to find out whether one can improve over the approximation ratio of our algorithm. Specifically, we leave open the question of whether there is a semi-streaming algorithm for finding a maximum weight set subject to a k-extendible constraint whose approximation ratio is clean $O(k)$.

References

[1] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 1–16, 2002.

[2] Ashwinkumar Badanidiyuru and Jan Vondrák. Fast algorithms for maximizing submodular functions. In *SODA*, pages 1497–1514, 2014.

[3] Piotr Berman. A $d/2$ approximation for maximum weight independent set in d-claw free graphs. *Nord. J. Comput.*, 7(3):178–184, 2000.

[4] Amit Chakrabarti and Sagar Kale. Submodular maximization meets streaming: matchings, matroids, and more. *Math. Program.*, 154(1-2):225–247, 2015.

[5] Chandra Chekuri, Shalmoli Gupta, and Kent Quanrud. Streaming algorithms for submodular function maximization. In *ICALP*, pages 318–330, 2015.

[6] Michael Crouch and Daniel M. Stubbs. Improved streaming algorithms for weighted matching, via unweighted matching. In *APPROX*, pages 96–104, 2014.

[7] Rafael da Ponte Barbosa, Alina Ene, Huy L. Nguyen, and Justin Ward. A new framework for distributed submodular maximization. In *FOCS*, pages 645–654, 2016.

[8] Moran Feldman, Christopher Harshaw, and Amin Karbasi. Greed is good: Near-optimal submodular maximization via greedy optimization. In *COLT*, pages 758–784, 2017.

[9] Moran Feldman, Amin Karbasi, and Ehsan Kazemi. Do less, get more: Streaming submodular maximization with subsampling. In *NeurIPS 2018*, pages 730–740, 2018.

[10] Moran Feldman, Joseph Naor, Roy Schwartz, and Justin Ward. Improved approximations for k-exchange systems - (extended abstract). In *ESA*, pages 784–798, 2011.

[11] Marshall L. Fisher, George L. Nemhauser, and Laurence A. Wolsey. An analysis of approximations for maximizing submodular set functions–II. *Mathematical Programming*, 8:73–87, 1978.

[12] Anupam Gupta, Aaron Roth, Grant Schoenebeck, and Kunal Talwar. Constrained non-monotone submodular maximization: Offline and secretary algorithms. In *WINE*, pages 246–257, 2010.

[13] Elad Hazan, Shmuel Safra, and Oded Schwartz. On the complexity of approximating k-set packing. *Computational Complexity*, 15(1):20–39, 2006.

[14] C. Hurkens and A. Schrijver. On the size of systems of sets every t of which have an sdr, with an application to the worst-case ratio of heuristics for packing problems. *SIAM Journal on Discrete Mathematics*, 2(1):68–72, 1989.

[15] Tom A. Jenkyns. The efficacy of the "greedy" algorithm. In *South Eastern Conference on Combinatorics, Graph Theory and Computing*, pages 341–350, 1976.

[16] Bernhard Korte and Dirk Hausmann. An analysis of the greedy heuristic for independence systems. *Annals of Discrete Math.*, 2:65–74, 1978.

[17] Jon Lee, Vahab S. Mirrokni, Viswanath Nagarajan, and Maxim Sviridenko. Maximizing non-monotone submodular functions under matroid or knapsack constraints. *SIAM J. Discrete Math.*, 23(4):2053–2078, 2010.

[18] Jon Lee, Maxim Sviridenko, and Jan Vondrák. Submodular maximization over multiple matroids via generalized exchange properties. *Math. Oper. Res.*, 35(4):795–806, 2010.

[19] Jon Lee, Maxim Sviridenko, and Jan Vondrák. Matroid matching: The power of local search. *SIAM J. Comput.*, 42(1):357–379, 2013.

[20] Julián Mestre. Greedy in approximation algorithms. In *ESA*, pages 528–539, 2006.

[21] Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, and Amin Karbasi. Fast constrained submodular maximization: Personalized data summarization. In *ICML*, pages 1358–1367, 2016.

[22] Baharan Mirzasoleiman, Stefanie Jegelka, and Andreas Krause. Streaming non-monotone submodular maximization: Personalized video summarization on the fly. In *AAAI*, pages 1379–1386, 2018.

[23] Maxim Sviridenko and Justin Ward. Large neighborhood local search for the maximum set packing problem. In *ICALP*, pages 792–803, 2013.

[24] Justin Ward. A $(k+3)/2$-approximation algorithm for monotone submodular k-set packing and general k-exchange systems. In *STACS*, pages 42–53, 2012.

A Hierarchies of Constraints classes

In this section, we discuss the hierarchy of constraints classes. First, we present another class of constraints named *k-matroid intersection*. Then, we discuss the relationship between this class and two classes defined in Section 2, namely k-set system and k-extendible. In particular, we show that k-matroid intersection $\subseteq$ k-extendible $\subseteq$ k-set system. Formally, we prove the following theorem.

Theorem A.1. *Any k-matroid intersection set system is a k-extendible set system, and any k-extendible set system is a k-set system.*

Before proving this theorem, let us formally define *matroids* and k-matroid intersection.

Definition A.1. *An independence system $(\mathcal{N}, \mathcal{I})$ is a matroid if for any two sets $S, T \in \mathcal{I}$ such that $|S| < |T|$, there is an element $u \in T \setminus S$ such that $S + u \in \mathcal{I}$.*

Definition A.2. *An independence system is a k-matroid intersection (or k-intersection), for an integer $k \geq 1$, if there exist k matroids $(\mathcal{N}, \mathcal{I}_1), \cdots, (\mathcal{N}, \mathcal{I}_k)$ over the same ground set $\mathcal{N}$ such that $\mathcal{I} = \bigcap_{i=1}^{k} \mathcal{I}_i$.*

The following lemma now proves the first part of Theorem A.1.

Lemma A.2. *Any k-matroid intersection set system is a k-extendible set system.*

Proof. The proof of the lemma is based on the next claim.

Claim 1. *Every matroid $(\mathcal{N}, \mathcal{I})$ is 1-extendible.*

Proof. Fix two sets $C \subset D \in \mathcal{I}$ and an element $x \notin D$ such that $C + x \in \mathcal{I}$. We need to find a $y \in D \setminus C$ such that $D - y + x \in \mathcal{I}$. Let $A = C + x$. We examine two complementary cases. The first case is when $|A| = |D|$. In this case, let y be the single element of $D \setminus C$. Clearly, we have $D - y + x = A \in \mathcal{I}$, and therefore, the property of extendibility holds. The remaining case is when $|D| > |A|$. Since $(\mathcal{N}, \mathcal{I})$ is a matroid, we can use A to form a set A', with $|A'| = |D|$, by adding to it elements of $D \setminus A$. The lemma now holds for the second case by noting that $A' \in \mathcal{I}$, and therefore, by setting y to be the single element of $D \setminus A'$, we get $D - y + x = A' \in \mathcal{I}$. □

Consider some k-intersection system $(\mathcal{N}, \mathcal{I})$, and let $(\mathcal{N}, \mathcal{I}_1), (\mathcal{N}, \mathcal{I}_2), \ldots, (\mathcal{N}, \mathcal{I}_k)$ denote the k matroids whose intersection forms $(\mathcal{N}, \mathcal{I})$. Fix now two sets $C \subset D \in \mathcal{I}$ and an elements $x \notin D$ such that $C + x \in \mathcal{I}$. We need to find a set $Y \subseteq D \setminus C$ such that $D \setminus Y + x \in \mathcal{I}$ and $|Y| \leq k$. According to Lemma 1, each matroid $(\mathcal{N}, \mathcal{I}_i)$ is 1-extendible, and therefore, there exists a $y_i \in D \setminus C$ such that $D - y_i + x \in \mathcal{I}_i$. Let $Y = \{y_i \mid 1 \leq i \leq k\}$, and note that $|Y| \leq k$. For every $1 \leq i \leq k$, the set $D \setminus Y + x$ belongs to $\mathcal{I}_i$ because it is a subset of $D - y_i + x$; and therefore, $D \setminus Y \in \mathcal{I}$. □

It now remains to prove the second part of Theorem A.1, which is done by the next lemma.

Lemma A.3. *Any k-extendible set system is a k-set system.*

Proof. Let $(\mathcal{N}, \mathcal{I})$ be a k-extendible set system for an integer $k \geq 1$, let F be some subset of $\mathcal{N}$, and let $A, B \subseteq F$ be two bases of $(F, 2^F \cap \mathcal{I})$ of maximum and minimum size, respectively. We need to show that A and B have the same size up to a factor of k, i.e., $|A| \leq k \cdot |B|$. Consider an execution of the unweighted greedy algorithm on $(F, 2^F \cap \mathcal{I})$ in which the first of elements of the stream are exactly the elements of B. Since $|B|$ is a base of F, it will be the output of the unweighted greedy algorithm in such an execution. Thus, by Lemma 4.3, $|A \setminus B| \leq k \cdot |B \setminus A|$. The lemma now holds by the following simple set theory arithmetic.

$$|A| = |A \cap B| + |A \setminus B| \leq k \cdot |A \cap B| + k \cdot |B \setminus A| = k \cdot |B| \ . \qquad \square$$

B Algorithm for General Weights

In this section we present a semi-streaming algorithm for k-power instances of the problem of finding a maximum weight set subject to a k-extendible constraint. Unlike Algorithm 2, this algorithm does not assume access to the bounds $w_{\max}$ and $w_{\min}$, and its space complexity remains nearly linear regardless of the ratio between these bounds. A more formal statement of the properties of this algorithm is given in Proposition B.1. Note that, together with Reduction 3.1, this proposition immediately implies Theorem 1.1.

Proposition B.1. *There exists a $4k$-approximation semi-streaming algorithm for k-power instances of the problem of finding a maximum weight set subject to a k-extendible constraint. The space complexity of this algorithm is $O(\rho(\log k + \log \rho)/\log k)$ under the assumption that constant space suffices to store a single element and a single weight.*

Throughout this section we assume for simplicity that the k-extendible system does not include any self-loops (a *self-loop* is an element $u \in \mathcal{N}$ such that $\{u\}$ is a dependent set—*i.e.*, $\{u\} \notin \mathcal{I}$). This assumption is without loss of generality since a self-loop cannot belong to any independent set, and thus, an algorithm can safely ignore self-loops if they happen to exist. One consequence of this assumption is that $\max_{u \in \mathcal{N}} w(u) \leq w(OPT)$, where OPT is an arbitrary fixed optimal solution like in the previous sections. This inequality holds since $\{u\}$ is a feasible solution for every element $u \in \mathcal{N}$, and therefore, its weight cannot exceed the weight of OPT.

As mentioned in Section 4, the algorithm we use to prove Proposition B.1 is a variant of Algorithm 2 that includes additional logic designed to force the ratio $w_{\max}/w_{\min}$ to be effectively polynomial—specifically, $O(k^2\rho^2)$. Given access to ρ and $\max_{u \in \mathcal{N}} w(u)$, this could be done simply by settings $w_{\max} = \max_{u \in \mathcal{N}} w(u)$ and $w_{\min} = \max_{u \in \mathcal{N}} w(u)/(2\rho)$ and discarding any element whose weight is lower then $w_{\min}$.[3] This guarantees that the ratio $w_{\max}/w_{\min}$ is small, and affects the weight of the optimal solution OPT by at most a constant factor since the total weight of the elements of this solution that get discarded is upper bounded by

$$|OPT| \cdot w_{\min} \leq \rho \cdot \frac{\max_{u \in \mathcal{N}} w(u)}{2\rho} = \frac{\max_{u \in \mathcal{N}} w(u)}{2} \leq \frac{w(OPT)}{2} \ .$$

Unfortunately, our algorithm does not have access (from the beginning) to ρ and $\max_{u \in \mathcal{N}} w(u)$. As an alternative, this algorithm, which is given as Algorithm 3, does two things. First, it keeps $w_{\max}$ equal to the maximum weight of the elements seen so far, which guarantees that eventually $w_{\max}$ becomes $\max_{u \in \mathcal{N}} w(u)$. Second, it runs the unweighted greedy algorithm on the input it receives. The size of the solution maintained by the unweighted greedy algorithm, which we denoted by g, provides an estimate for the maximum size of an independent set consisting only of elements that have already arrived. In particular, after all the elements arrive, $\rho/k \leq g \leq \rho$ because the unweighted greedy algorithm is a k-approximation algorithm.

Given the above discussion and the fact that the final value of kg is an upper bound on ρ, it is natural to define $w_{\min}$ as $w_{\max}/(2kg)$ and discard every element whose weight is lower than $w_{\min}$. Unfortunately, this does not work since $w_{\max}$ and g change during the execution of Algorithm 3, and reach their final values only when it terminates. Thus, we need to set $w_{\min}$ to a more conservative (lower) value. In particular, Algorithm 3 uses $w_{\min} = w_{\max}/(2gk)^2$.

Like Algorithm 2, Algorithm 3 maintains an instance of the unweighted greedy algorithm for every possible weight between $w_{\min}$ and $w_{\max}$. However, doing so is somewhat more involved for

[3] Starting from this point, $w_{\max}$ and $w_{\min}$ are no longer necessarily upper and lower bounds on the weights of all the elements. However, they remain upper and lower bounds on the weights of the non-discarded elements.

Algorithm 3 because $w_{\min}$ and $w_{\max}$ change during the algorithm's execution, which requires the algorithm to occasionally create and remove instances of unweighted greedy. The creation of such instances involves one subtle issue that needs to be kept in mind. In Algorithm 2 every instance of unweighted greedy associated with a weight w receives all elements whose weight is at least w. To mimic this behavior, when Algorithm 3 creates new instances of unweighted greedy following a decrease in $w_{\min}$ (which can happen when g increases), the newly created instances are not fresh new instances but copies of the instance of unweighted greedy that was previously associated with the lowest weight.

The rest of the details of Algorithm 3 are identical to the details of Algorithm 2. Specifically, every arriving element u is feed to every instance of unweighted greedy associated with a weight of $w(u)$ or less, and at termination the outputs of all the unweighted greedy instances are combined in the same way in which this is done in Algorithm 2.

Algorithm 3: Greedy of Greedies for Unbounded Weights

Create an instance of the unweighted greedy algorithm named `Greedy`, and let g denote the size of the solution maintained by it.
for *each element u that arrives from the stream* **do**
  Feed u to `Greedy`.
  if *u is the first element to arrive* **then**
    Let $w_{\max} \leftarrow w(u)$ and $w_{\min} \leftarrow w_{\max}/(2gk)^2$.
    Let $i_{\min} \leftarrow \lceil \log_k w_{\min} \rceil$ and $i_{\max} \leftarrow \log_k w_{\max}$.
    Create new instances of the unweighted greedy algorithm named $\texttt{Greedy}_{i_{\min}}$, $\texttt{Greedy}_{i_{\min}+1}, \ldots, \texttt{Greedy}_{i_{\max}}$.
  else
    Update $w_{\max} \leftarrow \max\{w_{\max}, w(u)\}$ and $i_{\max} \leftarrow \log_k w_{\max}$. If the value of $w_{\max}$ increased following this update, create new instances of unweighted greedy named $\texttt{Greedy}_{i'_{\max}+1}, \texttt{Greedy}_{i'_{\max}+2}, \ldots, \texttt{Greedy}_{i_{\max}}$, where $i'_{\max}$ is the old value of $i_{\max}$.[4]
    Update $w_{\min} \leftarrow w_{\max}/(2gk)^2$ and $i_{\min} \leftarrow \lceil \log_k w_{\min} \rceil$. If the value of $w_{\min}$ increased following this update, delete the instances of unweighted greedy named $\texttt{Greedy}_{i'_{\min}}$, $\texttt{Greedy}_{i'_{\min}+1}, \ldots, \texttt{Greedy}_{i_{\min}-1}$, where $i'_{\min}$ is the old value of $i_{\min}$. In contrast, if the value of $w_{\min}$ decreased following the update, copy $\texttt{Greedy}_{i'_{\min}}$ into new instances of unweighted greedy named $\texttt{Greedy}_{i_{\min}}, \texttt{Greedy}_{i_{\min}+1}, \ldots, \texttt{Greedy}_{i'_{\min}-1}$.
  if *$w(u) \geq w_{\min}$* **then**
    Let $i_u \leftarrow \log_k w(u)$.
    Feed u to $\texttt{Greedy}_{i_{\min}}, \texttt{Greedy}_{i_{\min}+1}, \ldots, \texttt{Greedy}_{i_u}$.
Let C_i denote the output of $\texttt{Greedy}_i$ for every $i_{\min} \leq i \leq i_{\max}$.
Let $T \leftarrow \varnothing$.
for *every $i_{\min} \leq i \leq i_{\max}$ in descending order* **do**
  Greedily add elements from C_i to T as long as this is possible.
  Let T_i denote the current value of T.
return T.

We now get to the analysis of Algorithm 3, and let us begin by bounding its space complexity. Let $g(h)$, $i_{\min}(h)$, $i_{\max}(h)$, $w_{\min}(h)$ and $w_{\max}(h)$ denote the values of g, $i_{\min}$, $i_{\max}$, $w_{\min}$ and $w_{\max}$, respectively, at the end of iteration number h of Algorithm 3.

Lemma B.2. *Algorithm 3 can be implemented using a space complexity of* $O(\rho(\log k+\log\rho)/\log k)$.

Proof. Using the same argument used in the proof of Lemma 4.4, it can be shown that the space complexity of Algorithm 3 is upper bounded by $O(\rho)$ times the maximum number of unweighted greedy instances maintained by the algorithm at the same time. By making the deletions of unweighted greedy instances precede the creation of new instances within every given iteration of the main loop of Algorithm 3 (and avoiding the creation of instances that need to be immediately deleted), it can be guaranteed that the maximum number of instances of unweighted greedy maintained by Algorithm 3 at any given time is exactly $\max_{1\leq h\leq n}\{i_{\max}(h)-i_{\min}(h)+2\}$. Thus, the algorithm's space complexity is at most

$$\begin{aligned}
&O(\rho)\cdot\max_{1\leq h\leq n}\{i_{\max}(h)-i_{\min}(h)+2\}=O(\rho)\cdot\max_{1\leq h\leq n}\{\log_k w_{\max}(h)-\log_k\lceil w_{\min}(h)\rceil+2\}\\
&\leq O(\rho)\cdot\max_{1\leq h\leq n}\left\{\log_k\left(\frac{w_{\max}}{w_{\min}}\right)+2\right\}=O(\rho)\cdot\max_{1\leq h\leq n}\left\{\log_k(2k\cdot g(h))^2+2\right\}\\
&\leq O(\rho)\cdot[\log_k(2\rho k)^2+2]\leq O(\rho)\cdot\frac{2\ln\rho+4\ln k+2}{\ln k}\ ,
\end{aligned}$$

where the second inequality is due to the fact that g is always the size of an independent set, and thus, cannot exceed ρ. □

Our next objective is to analyze the approximation ratio of Algorithm 3. Like in the toy analysis presented above for the case in which the algorithm has access to ρ and $\max_{u\in\mathcal{N}} w(u)$, the analysis we present starts by upper bounding the total weight of the discarded elements. However, to do that we need the following technical observation, which can be proved by induction.

Observation B.3. *Algorithm 3 maintains the invariant that, at the end of every one of its loops, if an element* $u\in\mathcal{N}$ *was fed to some instance of unweighted greedy currently kept by the algorithm, then it was fed exactly to those instances associated with a weight of at most* $\log_k w(u)$.

We say that an element $u\in\mathcal{N}$ is *discarded* by Algorithm 3 if u was never fed to the final instance $\texttt{Greedy}_{i_{\min}(n)}$ (during the execution of Algorithm 3 there might be multiple instances of unweighted greedy named $\texttt{Greedy}_i$ for $i=i_{\min}(n)$—by *final instance* we mean the last of these instances). Let F be the set of discarded elements.

Lemma B.4. $w(OPT\cap F)\leq\frac{1}{2}\cdot w(OPT)$.

Proof. For every $1\leq i\leq|OPT\cap F|$, let u_i be the i-th element of $OPT\cap F$ to arrive, and let h_i be its location in the input stream. Given Observation B.3, the fact that $u_i\in F$ implies that u_i was not feed to the final instance $\texttt{Greedy}_{\log_k w(u)}$, which can only happen if an instance named $\texttt{Greedy}_{\log_k w(u)}$ either did not exist when u_i arrived or was deleted at some point after u_i's arrival. Thus, $i_{\min}(h_i')>\log_k w(u_i)$ for some $h_i\leq h_i'\leq n$.

The crucial observation now is that $g(h_i')\geq g(h_i)\geq i/k$ because by the time u_i arrives there are already i elements of OPT that arrived, and these elements form together an independent set

[4] As written, Line 9 might create a large number of instances of unweighted greedy when there is a large increase in $w_{\max}$. However, when this happens most of the newly created instances are immediately deleted by Line 10. A smart implementation of Algorithm 3 can avoid the creation of unweighted greedy instances that are destined for such immediate deletion, and this is crucial for the analysis of the space complexity of Algorithm 3 in the proof of Lemma B.2.

of size i (recall that g is a k-approximation for the maximum size of an independent set consisting only of elements that already arrived). Thus, we get

$$w(u_i) = 2^{\log_k w(u_i)} \leq 2^{i_{\min}(h'_i)-1} \leq w_{\min}(h'_i) = \frac{w_{\max}(h'_i)}{(2k \cdot g(h'_i))^2} \leq \frac{\max_{u \in \mathcal{N}} w(u)}{(2k \cdot (i/k))^2} \leq \frac{w(OPT)}{4i^2} \ ,$$

where the first inequality holds since $i_{\min}(h'_i) > \log_k w(u_i)$ and both $i_{\min}(h'_i)$ and $\log_k w(u_i)$ are integers. Adding up the last inequality over $1 \leq i \leq |OPT \cap F|$ yields

$$w(OPT \cap F) = \sum_{i=1}^{|OPT \cap F|} w(u_i) \leq \sum_{i=1}^{|OPT \cap F|} \frac{w(OPT)}{4i^2} \leq \frac{w(OPT)}{4} \cdot \left[1 + \int_1^{\infty} i^{-2}\right] = \frac{w(OPT)}{2} \ . \quad \square$$

The next lemma shows that Algorithm 3 has a good approximation ratio with respect to the non-discarded elements of OPT.

Lemma B.5. *$w(OPT \setminus F) \leq 2k \cdot w(T)$.*

Proof. Observe that $(\mathcal{N} \setminus F, \mathcal{I} \cap 2^{\mathcal{N} \setminus F})$ is a k-extendible system, derived from $(\mathcal{N}, \mathcal{I})$ by removing all elements of F. In addition, all the weights of the elements of this set system are powers of k, and thus, by Proposition 4.1, Algorithm 2 achieves $2k$-approximation for the problem of finding a maximum weight independent set of $(\mathcal{N} \setminus F, \mathcal{I} \cap 2^{\mathcal{N} \setminus F})$. In other words, when Algorithm 2 is fed only the elements of $\mathcal{N} \setminus F$, its output set T' obeys $w(OPT') \leq 2k \cdot w(T')$, where OPT' is an arbitrary maximum weight set independent set of $(\mathcal{N} \setminus F, \mathcal{I} \cap 2^{\mathcal{N} \setminus F})$.

We now note that one consequence of Observation B.3 is that, by the time Algorithm 3 terminates, the instances $\texttt{Greedy}_{i_{\min}(n)}, \texttt{Greedy}_{i_{\min}(n)+1}, \ldots, \texttt{Greedy}_{i_{\max}(n)}$ it maintains receive exactly the input received by the corresponding instances in Algorithm 2 when the last algorithm gets only the elements of $\mathcal{N} \setminus F$ as input. Since Algorithms 2 and 3 compute their outputs based on the outputs of $\texttt{Greedy}_{i_{\min}(n)}, \texttt{Greedy}_{i_{\min}(n)+1}, \ldots, \texttt{Greedy}_{i_{\max}(n)}$ in the same way, this implies that the output set T of Algorithm 3 is identical to the output set T' produced by Algorithm 2 when this algorithm is given only the elements of $\mathcal{N} \setminus F$ as input.

Combining the above observations, we get

$$w(T) = w(T') \geq \frac{w(OPT')}{2k} \geq \frac{w(OPT \setminus F)}{2k} \ ,$$

where the last inequality holds since OPT' is a maximum weight independent set in $(\mathcal{N} \setminus F, \mathcal{I} \cap 2^{\mathcal{N} \setminus F})$ and $OPT \setminus F$ is independent in this set system. The lemma now follows by rearranging the last inequality. $\square$

Corollary B.6. *$w(OPT) \leq 4k \cdot w(T)$, and thus, the approximation ratio of Algorithm 3 is at most $4k$.*

Proof. Combining the last two lemmata, one gets

$$\frac{w(OPT)}{2} \leq w(OPT) - w(OPT \cap F) = w(OPT \setminus F) \leq 2k \cdot w(T) \ .$$

The corollary now follows by rearranging the above inequality. $\square$

We conclude the section by noticing that Proposition B.1 is an immediate consequence of Lemma B.2 and Corollary B.6.

הקירוב המובטח על ידי המשפט הוא אופטימלי עד כדי גורם כפלי של $O(\log k)$ מכיוון שידוע כבר שלא ניתן להשיג טוב יותר מקירוב k עבור הבעיה של מציאת תת-קבוצה בעלת משקל מקסימלי תחת אילוץ k-בת הרחבה אפילו בגרסה הלא מקוונת של הבעיה [8].

בכדי להוכיח את המשפט, בפרק 3 אנו מציגים רדוקציה של הבעיה שבה אנו עוסקים למקרה פרטי שלה שבו המשקלים הם כולם חזקות של k, אך מגדילה את יחס הקירוב פי $O(\log k)$. לאחר מכן, בפרק 4, אנחנו מציגים את אלגוריתם 2, שהוא אלגוריתם הזרמה למחצה למקרה הפרטי הזה. לאלגוריתם 2 שני שלבים. בשלב הראשון האלגוריתם מצמצם את הבעיה למספר פולי-לוגריתמי של מופעים של הבעיה של מציאת תת-קבוצה עם **גודל** מקסימלי (`maximum cardinality`) תחת אילוץ k-בת הרחבה, ומייצר באופן חמדני פתרון מקורב לכל אחד מהמופעים האלו. השלב השני של האלגוריתם מתרחש לאחר שהזרם מסתיים, ובו האלגוריתם ממזג באופן חמדני את הפתרונות המקורבים שיוצרו בשלב הראשון לפתרון אחד בעל יחס קירוב $O(k)$. באמצעות שילוב הרדוקציה ואלגוריתם זה למקרה הפרטי, מתקבל משפט 1.

אלגוריתם 2 והניתוח שלו דומים מאוד לאלו של האלגוריתם של קראוץ וסטאבס [6], שיחס הקירוב שלו הוא $O(k^2)$ (למקרה הכללי). העובדה שאנחנו במקרה הפרטי שבו המשקלים הם כולם חזקות של k והאילוץ הוא k-בת הרחבה היא זו שמאפשרת לנו לקבל יחס קירוב משופר של $O(k)$.

ידיעתנו, האלגוריתם היחיד שידוע כיום לבעיה הכללית במודל זה הוא אלגוריתם הזרמה למחצה (semi-streaming) המשיג יחס קירוב של $k^2(1+\varepsilon)$ [6]. נציין כי אלגוריתם במודל זרם המידע נחשב לאלגוריתם הזרמה למחצה אם סיבוכיות המקום שלו לינארית בגודל הפלט המקסימאלי האפשרי עבורו עד כדי גורמים פולי-לוגריתמיים (זו ההגדרה המקובלת של הזרמה למחצה, אבל לא היחידה בספרות).

כפי שניתן לראות בבירור, במודל זרם המידע ישנו פער גדול בין הבעיה המקוונת והלא-מקוונת מכיוון שיחס הקירוב שמשיג האלגוריתם האחרון רחוק מאוד מיחס הקירוב k שמשיג האלגוריתם החמדן עבור הבעיה הלא מקוונת. כאמור, פותחו בעבודות קודמות אלגוריתמים במודל זרם המידע עבור מקרים פרטיים של הבעיה. בין היתר, פותחו אלגוריתמי הזרמה למחצה שיחס הקירוב שלהם הוא $O(k)$ למחלקות מצומצמות יותר של אילוצים הכלולות בתוך מחלקת אילוצי ה-k-מערכת קבוצות, כאשר הרחבה ביותר מבין מחלקות אלה היא מחלקת אילוצי ה-k-זיווגואיד (k-matchoid constraint) [4,5,9,22]. אלגוריתמים אלה מגשרים באופן חלקי על הפער שצוין לעיל בין יחס הקירוב שניתן להשיג עבור הגרסה הלא מקוונת של הבעיה הכללית והגרסה שלה במודל הזרם, אך גישור זה היינו חלקי מאוד מכיוון שמחלקת אילוצי k-מערכת קבוצות היא הרבה יותר רחבה ממחלקת אילוצי k-זיווגואיד.

מסטרה [20] הציג מחלקת אילוצים רחבה פחות ממחלקת אילוצי k-מערכת קבוצות, אך רחבה הרבה יותר ממחלקת אילוצי k-זיווגואיד, בשם k-בת הרחבה (k-extendible). למיטב ידיעתנו, מחלקה זו מכלילה את כל המקרים הפרטיים האחרים של מחלקת אילוצי k-מערכת קבוצות שנחקרו בספרות המקצועית עד היום (כולל, בפרט, את מחלקת אילוצי k-זיווגואיד). מצד שני, אלגוריתם ההזרמה למחצה הטוב ביותר הידוע כיום עבור מחלקה זו של אילוצים הוא בדיוק אותו אלגוריתם הידוע עבור אילוצי k-מערכת קבוצות כלליים, ומשיג רק יחס קירוב של $k^2(1+\varepsilon)$. בעבודה זו אנו מציגים אלגוריתם הזרמה למחצה בעל יחס קירוב של $\tilde{O}(k)$ לבעיה של מציאת תת-קבוצה עם משקל מקסימלי תחת אילוץ k-בת הרחבה במטרה לנסות לגשר על הפער הנזכר לעיל. תוצאות העבודה שלנו מרוכזות במשפט הבא.

<u>משפט 1.</u> קיים אלגוריתם קירוב הזרמה למחצה שעובד בזמן פולינומיאלי ובעל יחס קירוב $O(k\cdot\log(k))$ לבעיה של מציאת תת-קבוצה עם משקל מקסימלי תחת אילוץ k-בת הרחבה במודל זרם המידע. תחת ההנחה שדרוש מקום בגודל קבוע על מנת לאכסן איבר יחיד ומשקל יחיד, וש-ρ הוא הגודל המקסימלי של פיתרון חוקי, סיבוכיות המקום של האלגוריתם היא $O(\rho\cdot(\log(k)+\log(\rho)))$.

מכיוון שמחלקת אילוצי k-בת הרחבה מכלילה את שאר תתי מחלקות האילוצים של k-מערכת קבוצות שנחקרו, אנו מאמינים כי תוצאה זו מהווה את שלב הביניים האחרון בדרך לסגירת הפער הנ"ל לחלוטין (כלומר, מציאת אלגוריתם הזרמה למחצה עם יחס קירוב $\tilde{O}(k)$ תחת אילוץ k-מערכת קבוצות, או הוכחה שלא קיים אלגוריתם כזה). בנוסף, שימו לב שיחס

תקציר

בעבודה זו אנחנו חוקרים את הבעיה של מציאת תת-קבוצה עם משקל מקסימלי תחת אילוץ k-בת הרחבה במודל זרם המידע. האלגוריתם הלא-טריוויאלי היחיד שידוע היום לבעיה זו – למיטב ידיעתנו – הוא אלגוריתם הזרמה למחצה בעל יחס קירוב של $k^2(1+\varepsilon)$ (קראוץ׳ וסטאבס, 2014), אבל ידועים אלגוריתמי הזרמה למחצה בעלי יחס קירוב $O(k)$ עבור מקרים פרטיים של הבעיה הכללית הזו. בעבודה זו אנחנו סוגרים את רובו של הפער הנזכר לעיל על ידי הצגת אלגוריתם הזרמה למחצה בעל יחס קירוב $O(k\cdot\log(k))$ לבעיה הכללית. ידוע שיחס קירוב זה הוא כמעט אופטימלי אפילו עבור הגרסה הלא מקוונת של הבעיה (פלדמן ואחרים, 2017).

מבוא

קיימות לא מעט בעיות אופטימיזציה קומבינטוריות שניתנות להכללה על-ידי הבעיה הבאה. בהינתן קבוצת בסיס (ground set) N של אלמנטים ממושקלים, מצא תת קבוצה של N עם משקל מקסימלי, תחת אילוץ מסוים C. הבעיה הזו רחבה מאוד, וכוללת בתוכה בעיות קשות כגון מציאת קבוצה בלתי תלויה בגודל מקסימלי בגרף, ולכן קשה למצוא קירוב סביר לפתרונה. לעומת זאת, בספרות המקצועית כיום ישנן מחלקות של אילוצים שעבורן ניתן למצוא פתרונות מקורבים טובים, ולעיתים גם פתרונות אופטימליים. בשנות ה-70 גם קורטה והאוסמן [16] וגם ג׳נקינס [15] הציעו מחלקת אילוצים בשם k-מערכת קבוצות (k-set system). הבעיה של מציאת תת-קבוצה בעלת משקל מקסימלי תחת אילוץ k-מערכת קבוצות מכלילה הרבה בעיות מוכרות כגון זיווג בגרף-על, חיתוך מטרואידים, והגרסה האסימטרית של בעיית הסוכן הנוסע. בנוסף, אלגוריתם חמדני פשוט מוצא פתרון מקורב בעל יחס קירוב k לבעיה הכללית הזו [15,14]. לא מזמן הוכח שיחס קירוב זה הוא הטוב ביותר שניתן להשיג [2]. עבודות נוספות בנושא הצליחו להשיג יחס קירוב משופר עבור מחלקות אילוצים קטנות יותר [9,17,18] וגם להרחיב את התוצאה עצמה לבעיה כללית יותר, כגון מציאת מקסימום של פונקציה תת-מודולרית (maximizing sub-modular function) תחת אילוץ k-מערכת קבוצות [7,9,10,11,16,17,20,23].

בגלל מספר הבעיות החשובות שהבעיה מכלילה, ובגלל כמויות המידע העצומות עמן נאלצים להתמודד יישומים מודרניים (למשל בתחומי האינטרנט של הדברים (IoT) וחישוב מבוסס ענן), חוקרים החלו לחקור את הבעיה של מציאת תת-קבוצה עם משקל מקסימלי תחת אילוץ k-מערכת קבוצות בסביבות נתונים מרובים (Big Data). בין היתר, בעיה זו נחקרה במודל זרם המידע (Data Stream Model), שבו הקלט לא זמין לגישה אקראית אלא מגיע ביחידות כחלק מזרם [1], ומודל המיפוי צמצום (Map-Reduce). עבור מודל המיפוי צמצום נמצא אלגוריתם בעל יחס קירוב $k+O(\varepsilon)$ המשתמש ב $O(1/\varepsilon)$ סבבים [6]. במודל זרם המידע הוצעו מספר אלגוריתמים עבור מקרים פרטיים של הבעיה [10,18,19], אך, למיטב

תוכן עניינים

רשימת איורים

הזרמה למחצה כמעט אופטימלי למערכת k–בת הרחבה

רן האבא 16-206874

בהנחיית פרופ' מורן פלדמן

המחלקה למדעי המחשב, האוניברסיטה הפתוחה של ישראל
13/11/2019