FULL LENGTH PAPER

Series B

Semi-streaming algorithms for submodular matroid intersection

Paritosh Garg[1] · **Linus Jordan[1]** · **Ola Svensson[1]**

Abstract

While the basic greedy algorithm gives a semi-streaming algorithm with an approximation guarantee of 2 for the *unweighted* matching problem, it was only recently that Paz and Schwartzman obtained an analogous result for weighted instances. Their approach is based on the versatile local ratio technique and also applies to generalizations such as weighted hypergraph matchings. However, the framework for the analysis fails for the related problem of weighted matroid intersection and as a result the approximation guarantee for weighted instances did not match the factor 2 achieved by the greedy algorithm for unweighted instances.Our main result closes this gap by developing a semi-streaming algorithm with an approximation guarantee of $2 + \varepsilon$ for *weighted* matroid intersection, improving upon the previous best guarantee of $4 + \varepsilon$. Our techniques also allow us to generalize recent results by Levin and Wajc on submodular maximization subject to matching constraints to that of matroid-intersection constraints. While our algorithm is an adaptation of the local ratio technique used in previous works, the analysis deviates significantly and relies on structural properties of matroid intersection, called kernels. Finally, we also conjecture that our algorithm gives a $(k + \varepsilon)$ approximation for the intersection of k matroids but prove that new tools are needed in the analysis as the structural properties we use fail for $k \geq 3$.

✉ Paritosh Garg
   paritosh.garg@epfl.ch

   Linus Jordan
   linus.jordan@bluewin.ch

   Ola Svensson
   ola.svensson@epfl.ch

1   EPFL, Rte Cantonale, 1015 Lausanne, Switzerland

Published online: 27 July 2022

🖄 Springer

1 Introduction

For large problems, it is often not realistic to assume that the entire input can be stored in random access memory so more memory efficient algorithms are preferable. A popular model for such algorithms is the (semi-)streaming model (see e.g. [13]): the elements of the input are fed to the algorithm in a stream and the algorithm is required to have a small memory footprint.

Consider the classic maximum matching problem in an undirected graph $G = (V, E)$. An algorithm in the semi-streaming model[1] is fed the edges one-by-one in a stream $e_1, e_2, \ldots, e_{|E|}$ and at any point of time the algorithm is only allowed $O(|V| \operatorname{polylog}(|V|))$ bits of storage. The goal is to output a large matching $M \subseteq E$ at the end of the stream. Note that the allowed memory usage is sufficient for the algorithm to store a solution M but in general it is much smaller than the size of the input since the number of edges may be as many as $|V|^2/2$. Indeed, the intuitive difficulty in designing a semi-streaming algorithm is that the algorithm needs to discard many of the seen edges (due to the memory restriction) without knowing the future edges and still return a good solution at the end of the stream.

For the unweighted matching problem, the best known semi-streaming algorithm is the basic greedy approach:

> Initially, let $M = \emptyset$. Then for each edge e in the stream, add it to M if $M \cup \{e\}$ is a feasible solution, i.e., a matching; otherwise the edge e is discarded.

The algorithm uses space $O(|V| \log |V|)$ and a simple proof shows that it returns a 2-approximate solution in the *unweighted* case, i.e, a matching of size at least half the size of an maximum matching. However, this basic approach fails to achieve any approximation guarantee for *weighted graphs*.

Indeed, for weighted matchings, it is non-trivial to even get a small constant-factor approximation. One way to do so is to replace edges if we have a much heavier edge. This is formalized in [7] who get a 6-approximation. Later, [12] improved this algorithm to find a 5.828-approximation; and, with a more involved technique, [5] provided a $(4 + \varepsilon)$-approximation.

It was only in recent breakthrough work [14] that the gap in the approximation guarantee between unweighted and weighted matchings was closed. Specifically, [14] gave a semi-streaming algorithm for weighted matchings with an approximation guarantee of $2 + \varepsilon$ for every $\varepsilon > 0$. Shortly after, [9] came up with a simplified analysis of their algorithm, reducing the memory requirement from $O_\varepsilon(|V| \log^2 |V|)$ to $O_\varepsilon(|V| \log |V|)$. These results for weighted matchings are tight (up to the ε) in the

---

[1] This model can also be considered in the multi-pass setting when the algorithm is allowed to take several passes over the stream. However, in this work we focus on the most basic and widely studied setting in which the algorithm takes a single pass over the stream.

sense that any improvement would also improve the state-of-the-art in the unweighted case, which is a long-standing open problem.

The algorithm of [14] is an elegant use of the local ratio technique [2, 3] in the semi-streaming setting. While this technique is very versatile and it readily generalizes to weighted hypergraph matchings, it is much harder to use it for the related problem of weighted matroid intersection. This is perhaps surprising as many of the prior results for the matching problem also applies to the matroid intersection problem in the semi-streaming model (see Sect. 2 for definitions). Indeed, the greedy algorithm still returns a 2-approximate solution in the unweighted case and the algorithm in [5] returns a $(4 + \varepsilon)$-approximate solution for weighted instances. So, prior to our work, the status of the matroid intersection problem was that of the matching problem *before* [14].

We now describe at a high-level the reason that the techniques from [14] are not easily applicable to matroid intersection and our approach for dealing with this difficulty. The approach in [14] works in two parts, first certain elements of the stream are selected and added to a set S, and then at the end of the stream a matching M is computed by the greedy algorithm that inspects the edges of S in the reverse order in which they were added. This way of constructing the solution M greedily by going backwards in time is a standard framework for analyzing algorithms based on the local ratio technique. Now in order to adapt their algorithm to matroid intersection, recall that the bipartite matching problem can be formulated as the intersection of two partition matroids. We can thus reinterpret their algorithm and analysis in this setting. Furthermore, after this reinterpretation, it is not too hard to define an algorithm that works for the intersection of any two matroids. However, bipartite matching is a *special* case of matroid intersection which captures a rich set of seemingly more complex problems. This added expressiveness causes the analysis and the standard framework for analyzing local ratio algorithms to fail. Specifically, we prove that a solution formed by running the greedy algorithm on S in the reverse order (as done for the matching problem) fails to give any constant-factor approximation guarantee for the matroid intersection problem. To overcome this and to obtain our main result, we make a connection to a concept called matroid kernels (see [8] for more details about kernels), which allows us to, in a more complex way, identify a subset of S with an approximation guarantee of $2 + \varepsilon$.

Finally, for the intersection of more than two matroids, the same approach in the analysis does not work, because the notion of matroid kernel does not generalize to more than two matroids. However, we conjecture that the subset S generated for the intersection of k matroids still contains a $(k + \varepsilon)$-approximation. Currently, the best approximation results are a $(k^2 + \varepsilon)$-approximation from [5] and a $(2(k + \sqrt{k(k-1)}) - 1)$-approximation from [4]. For $k = 3$, the former is better, giving a $(9 + \varepsilon)$-approximation. For $k > 3$, the latter is better, giving an $O(k)$-approximation.

Generalization to submodular functions. Very recently, Levin and Wajc [11] obtained improved approximation ratios for matching and b-matching problems in the semi-streaming model with respect to submodular functions.

Specifically, they get a $(3 + 2\sqrt{2})$-approximation for monotone submodular b-matching, $(4 + 3\sqrt{2})$-approximation for non-monotone submodular matching, and a $(3 + \varepsilon)$-approximation for maximum weight (linear) b-matching. In our paper, we are

able to extend our algorithm for weighted matroid intersection to work with submodular functions by combining our and their ideas. In fact, we are able to generalize all their results to the case of matroid intersection with better or equal[2] approximation ratios: we get $(3 + 2\sqrt{2} + \delta)$-approximation for monotone submodular matroid intersection, $(4 + 3\sqrt{2} + \delta)$-approximation for non-monotone submodular matroid intersection and $(2 + \varepsilon)$-approximation for maximum weight (linear) matroid intersection.

Outline. In Sect. 2 we introduce basic matroid concepts and we formally define the weighted matroid intersection problem in the semi-streaming model. Sections 3 and 4 are devoted to our main result, i.e., the semi-streaming algorithm for weighted matroid intersection with an approximation guarantee of $(2 + \varepsilon)$. Specifically, in Sect. 3 we adapt the algorithm of [14] without worrying about the memory requirements, show why the standard analysis fails, and then give our new analysis. We then make the obtained algorithm memory efficient in Sect. 4. Further in Sect. 5, we adapt our algorithm to work with submodular functions by using ideas from [11]. Finally, in Sect. 6, we discuss the case of more than two matroids.

2 Preliminaries

Matroids. We define and give a brief overview of the basic concepts related to matroids that we use in this paper. For a more comprehensive treatment, we refer the reader to [15]. A *matroid* is a tuple $M = (E, I)$ consisting of a finite ground set E and a family $I \subseteq 2^E$ of subsets of E satisfying:

– if $X \subseteq Y, Y \in I$, then $X \in I$; and
– if $X \in I, Y \in I$ and $|Y| > |X|$, then $\exists\, e \in Y \backslash X$ such that $X \cup \{e\} \in I$.

The elements in I (that are subsets of E) are referred to as the *independent sets* of the matroid and the set E is referred to as the *ground set*. With a matroid $M = (E, I)$, we associate the *rank function* $\mathrm{rank}_M : 2^E \to \mathbb{N}$ and the *span function* $\mathrm{span}_M : 2^E \to 2^E$ defined as follows for every $E' \subseteq E$,

$$\mathrm{rank}_M(E') = \max\{|X| \mid X \subseteq E' \text{ and } X \in I\},$$
$$\mathrm{span}_M(E') = \{e \in E \mid \mathrm{rank}_M(E' \cup \{e\}) = \mathrm{rank}_M(E')\}.$$

We simply write $\mathrm{rank}(\cdot)$ and $\mathrm{span}(\cdot)$ when the matroid M is clear from the context. In words, the rank function equals the size of the largest independent set when restricted to E' and the span function equals the elements in E' and all elements that cannot be added to a maximum cardinality independent set of E' while maintaining independence. The *rank of the matroid* equals $\mathrm{rank}(E)$, i.e., the size of the largest independent set.

The weighted matroid intersection problem in the semi-streaming model. In the *weighted matroid intersection problem*, we are given two matroids $M_1 = (E, I_1), M_2 = (E, I_2)$ on a common ground set E and a non-negative weight function $w : E \to \mathbb{R}_{\geq 0}$

---

on the elements of the ground set. The goal is to find a subset $X \subseteq E$ that is independent in both matroids, i.e., $X \in I_1$ and $X \in I_2$, and whose weight $w(X) = \sum_{e \in X} w(e)$ is maximized.

In seminal work [6], Edmonds gave a polynomial-time algorithm for solving the weighted matroid intersection problem to optimality in the classic model of computation when the whole input is available to the algorithm throughout the computation. In contrast, the problem becomes significantly harder and tight results are still eluding us in the semi-streaming model where the memory footprint of the algorithm and its access pattern to the input are restricted. Specifically, in the *semi-streaming* model the ground set E is revealed in a stream $e_1, e_2, \ldots, e_{|E|}$ and at time i the algorithm gets access to e_i and can perform computation based on e_i and its current memory but without knowledge of future elements $e_{i+1}, \ldots, e_{|E|}$. The algorithm has independence-oracle access to the matroids M_1 and M_2 restricted to the elements stored in the memory, i.e., for a set of such elements, the algorithm can query whether the set is independent in each matroid.. The goal is to design an algorithm such that (i) the memory usage is near-linear $O((r_1 + r_2) \operatorname{polylog}(r_1 + r_2))$ at any time, where r_1 and r_2 denote the ranks of the input matroids M_1 and M_2, respectively, and (ii) at the end of the stream the algorithm should output a feasible solution $X \subseteq E$, i.e., a subset X that satisfies $X \in I_1$ and $X \in I_2$, of large weight $w(X)$. We remark that the memory requirement $O((r_1 + r_2) \operatorname{polylog}(r_1 + r_2))$ is natural as $r_1 + r_2 = |V|$ when formulating a bipartite matching problem as the intersection of two matroids[3].

The difficulty in designing a good semi-streaming algorithm is that the memory requirement is much smaller than the size of the ground set E and thus the algorithm must intuitively discard many of the elements without knowledge of the future and without significantly deteriorating the weight of the final solution X. The quality of the algorithm is measured in terms of its approximation guarantee: an algorithm is said to have an *approximation guarantee* of α if it is guaranteed to output a solution X, no matter the input and the order of the stream, such that $w(X) \geq \text{OPT}/\alpha$ where OPT denotes the weight of an optimal solution to the instance. As aforementioned, our main result in this paper is a semi-streaming algorithm with an approximation guarantee of $2 + \varepsilon$, for every $\varepsilon > 0$, improving upon the previous best guarantee of $4 + \varepsilon$ [5].

3 The local ratio technique for weighted matroid intersection

In this section, we first present the local ratio algorithm for the weighted matching problem that forms the basis of the semi-streaming algorithm in [14]. We then adapt it to the weighted matroid intersection problem. While the algorithm is fairly natural to adapt to this setting, we give an example in Sect. 3.2.1 that shows that the

---

[3] The considered problem can also be formulated as the problem of finding an independent set in one of the matroids, say M_1, and maximizing a submodular function which would be the (weighted) rank function of M_2. For that problem, [10] recently gave a streaming algorithm with an approximation guarantee of $(2 + \varepsilon)$. However, the space requirement of their algorithm is exponential in the rank of M_1 (which would correspond to exponential in $|V|$ in the matching case) and thus it does not provide a meaningful algorithm for our setting.

same techniques as used for analyzing the algorithm for matchings does not work for matroid intersection. Instead, our analysis, which is presented in Sect. 3.3, deviates from the standard framework for analyzing local ratio algorithms and it heavily relies on a structural property of matroid intersection known as kernels. We remark that the algorithms considered in this section do not have a small memory footprint. We deal with this in Sect. 4 to obtain our semi-streaming algorithm.

3.1 Local-ratio technique for weighted matching

The local ratio algorithm for the weighted matching problem is given in Algorithm 1. The algorithm maintains vertex potentials $w(u)$ for every vertex u, a set S of selected edges, and an auxiliary weight function $g : S \to \mathbb{R}_{\geq 0}$ of the selected edges. Initially the vertex potentials are set to 0 and the set S is empty. When an edge $e = \{u, v\}$ arrives, the algorithm computes how much it gains compared to the previous edges, by taking its weight minus the weight/potential of its endpoints ($g(e) = w(e) - w(u) - w(v)$). If the gain is positive, then we add the edge to S, and add the gain to the weight of the endpoints, that is, we set $w(u) = w(u) + g(e)$ and $w(v) = w(v) + g(e)$.

Algorithm 1 Local ratio algorithm for weighted matching

Input: A stream of the edges of a graph $G = (V, E)$ with a weight function $w : E \to \mathbb{R}_{\geq 0}$.
Output: A matching M.
1: $S \leftarrow \emptyset$
2: $\forall u \in V, w(u) \leftarrow 0$
3: **for** edge $e = (u, v)$ in the stream **do**
4:     **if** $w(u) + w(v) < w(e)$ **then**
5:         $g(e) \leftarrow w(e) - w(u) - w(v)$
6:         $w(u) \leftarrow w(u) + g(e)$
7:         $w(v) \leftarrow w(v) + g(e)$
8:         $S \leftarrow S \cup \{e\}$
9:     **end if**
10: **end for**
11: **return** a maximum weight matching M among the edges stored on the stack S

For a better intuition of the algorithm, consider the example depicted on the top of Fig. 1. The stream consists of four edges e_1, e_2, e_3, e_4 with weights $w(e_1) = 1$ and $w(e_2) = w(e_3) = w(e_4) = 2$. At each time step i, we depict the arriving edge e_i in thick along with its weight; the vertex potentials before the algorithm considers this edge is written on the vertices, and the updated vertex potentials (if any) after considering e_i are depicted next to the incident vertices. The edges that are added to S are solid and those that are not added to S are dashed.

At the arrival of the first edge of weight $w(e_1) = 1$, both incident vertices have potential 0 and so the algorithm adds this edge to S and increases the incident vertex potentials with the gain $g(e_1) = 1$. For the second edge of weight $w(e_2) = 2$, the sum of incident vertex potentials is 1 and so the gain of e_2 is $g(e_2) = 2 - 1$, which in turn causes the algorithm to add this edge to S and to increase the incident vertex potentials by 1. The third time step is similar to the second. At the last time step, edge
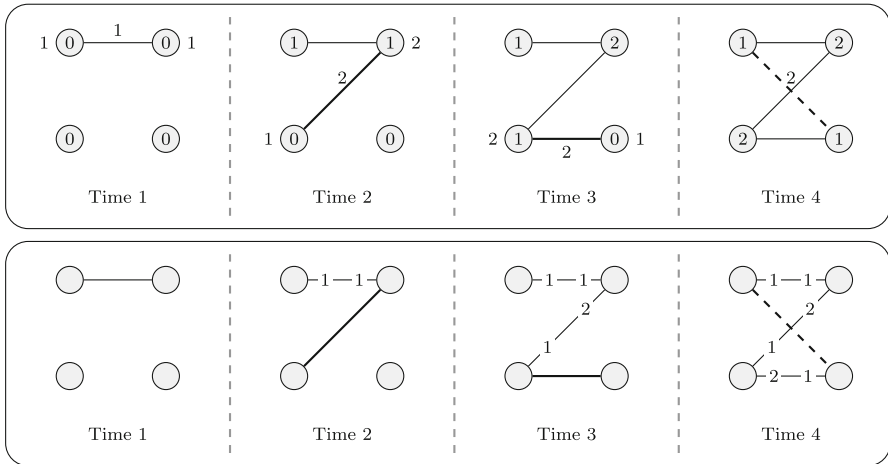
Fig. 1 The top part shows an example execution of the local ratio technique for weighted matchings (Algorithm 1). The bottom part shows how to adapt this (bipartite) example to the language of weighted matroid intersection (Algorithm 2)

e_4 arrives of weight $w(e_4) = 2$. As the incident vertex potentials sum up to 2 the gain of e_4 is not strictly positive and so this edge is *not* added to S and no vertex potentials are updated. Finally, the algorithm returns the maximum weight matching in S which in this case consists of edges $\{e_1, e_3\}$ and has weight 3. Note that the optimal matching of this instance had weight 4 and we thus found a 4/3-approximate solution.

In general, the algorithm has an approximation guarantee of 2. This is proved using a common framework to analyze algorithms based on the local ratio technique: We ignore the weights and greedily construct a matching M by inspecting the edges in S in reverse order, i.e., we first consider the edges that were added last. An easy proof (see e.g. [9]) then shows that the matching M constructed in this way has weight at least half the optimum weight.

In the next section, we adapt the above described algorithm to the context of matroid intersections. We also give an example that the above framework for the analysis fails to give any constant-factor approximation guarantee. Our alternative (tight) analysis of this algorithm is then given in Sect. 3.3.

3.2 Adaptation to weighted matroid intersection

When adapting Algorithm 1 to matroid intersection to obtain Algorithm 2, the first problem we encounter is the fact that matroids do not have a notion of vertices, so we cannot keep a weight/potential for each vertex. To describe how we overcome this issue, it is helpful to consider the case of bipartite matching and in particular the example depicted in Fig. 1. It is well known that the weighted matching problem on a bipartite graph with edge set E and bipartition V_1, V_2 can be modelled as a weighted matroid intersection problem on matroids $M_1 = (E, I_1)$ and $M_2 = (E, I_2)$ where for

---

Algorithm 2 Local ratio for matroid intersection

Input: A stream of the elements of the common ground set of matroids $M_1 = (E, I_1)$, $M_2 = (E, I_2)$.
Output: A set $X \subseteq E$ that is independent in both matroids.
  $S \leftarrow \emptyset$
  for element e in the stream **do**
    calculate $w_i^*(e) = \max \left(\{0\} \cup \{\theta : e \in \text{span}_{M_i} (\{f \in S \mid w_i(f) \geq \theta\})\} \right)$ for $i \in \{1, 2\}$.
    if $w(e) > w_1^*(e) + w_2^*(e)$ **then**
      $g(e) \leftarrow w(e) - w_1^*(e) - w_2^*(e)$
      $w_1(e) \leftarrow w_1^*(e) + g(e)$
      $w_2(e) \leftarrow w_2^*(e) + g(e)$
      $S \leftarrow S \cup \{e\}$
    end if
  end for
  return a maximum weight set $T \subseteq S$ that is independent in M_1 and M_2

$i \in \{1, 2\}$

$$I_i = \{E' \subseteq E \mid \text{each vertex } v \in V_i \text{ is incident to at most one vertex in } E'\}.$$

Instead of keeping a weight for each vertex, we will maintain two weight functions w_1 and w_2, one for each matroid. These weight functions will be set so that the following holds in the special case of bipartite matching: on the arrival of a new edge e, let $T_i \subseteq S$ be an independent set in I_i of selected edges that maximizes the weight function w_i. Then we have that

$$\min_{f \in T_i : T_i \setminus \{f\} \cup \{e\} \in I_i} w_i(f) \qquad \text{if } T_i \cup \{e\} \notin I_i \text{ and 0 otherwise} \tag{1}$$

equals the vertex potential of the incident vertex V_i when running Algorithm 1. It is well-known (e.g. by the optimality of the greedy algorithm for matroids) that the cheapest element f to remove from T_i to make $T_i \setminus \{f\} \cup \{e\}$ an independent set equals the largest weight θ so that the elements of weight at least θ spans e. We thus have that (1) equals

$$\max \left(\{0\} \cup \{\theta : e \in \text{span}_{M_i} (\{f \in S \mid w_i(f) \geq \theta\})\} \right)$$

and it follows that the quantities $w_1^*(e)$ and $w_2^*(e)$ in Algorithm 2 equal the incident vertex potentials in V_1 and V_2 of Algorithm 1 in the special case of bipartite matching. To see this, let us return to our example in Fig. 1 and let V_1 be the two vertices on the left and V_2 be the two vertices on the right. In the bottom part of the figure, the weight functions w_1 and w_2 are depicted (at the corresponding side of the edge) after the arrival of each edge. At time step 1, e_1 does not need to replace any elements in any of the matroids and so $w_1^*(e_1) = w_1^*(e_2) = 0$. We therefore have that its gain is $g(e_1) = 1$ and the algorithm sets $w_1(e_1) = w_2(e_1) = 1$. At time 2, edge e_2 of weight 2 arrives. It is not spanned in the first matroid whereas it is spanned by edge e_1 of weight 1 in the second matroid. It follows that $w_1^*(e_2) = 0$ and $w_2^*(e_2) = w_2(e_1) = 1$ and so e_2 has positive gain $g(e_2) = 1$ and it sets $w_1(e_2) = 1$ and $w_2(e_2) = w_2(e_1) + 1 = 2$. The third time step is similar to the second. At the last time step, e_4 of weight 2 arrives.

However, since it is spanned by e_1 with $w_1(e_1) = 1$ in the first matroid and by e_3 with $w_2(e_3) = 1$ in the second matroid, its gain is 0 and it is thus not added to the set S. Note that throughout this example, and in general for bipartite graphs, Algorithm 2 is identical to Algorithm 1. One may therefore expect that the analysis of Algorithm 1 also generalizes to Algorithm 2. We explain next that this is not the case for general matroids.

3.2.1 Counter example to same approach in analysis

We give a simple example showing that the greedy selection (as done in the analysis for Algorithm 1 for weighted matching) does not work for matroid intersection. Still, it turns out that the set S generated by Algorithm 2 always contains a 2-approximation but the selection process is more involved.

Lemma 1 *There exist two matroids $M_1 = (E, I_1)$ and $M_2 = (E, I_2)$ on a common ground set E and a weight function $w : E \rightarrow \mathbb{R}_{\geq 0}$ such that a greedy algorithm that considers the elements in the set S in the reverse order of when they were added by Algorithm 2 does not provide any constant-factor approximation.*

Proof The example consists of the ground set $E = \{a, b, c, d\}$ with weights $w(a) = 1, w(b) = 1 + \varepsilon, w(c) = 2\varepsilon, w(d) = 3\varepsilon$ for a small $\varepsilon > 0$ (the approximation guarantee will be at least $\Omega(1/\varepsilon)$). The matroids $M_1 = (E, I_1)$ and $M_2 = (E, I_2)$ are defined by

– a subset of E is in I_1 if and only if it does not contain $\{a, b\}$; and
– a subset of E is in I_2 if and only if it contains at most two elements.

To see that M_1 and M_2 are matroids, note that M_1 is a partition matroid with partitions $\{a, b\}, \{c\}, \{d\}$, and M_2 is the 2-uniform matroid (alternatively, one can easily check that M_1 and M_2 satisfy the definition of a matroid).

Now consider the execution of Algorithm 2 when given the elements of E in the order a, b, c, d:

– Element a has weight 1, and $\{a\}$ is independent both in M_1 and M_2, so we set $w_1(a) = w_2(a) = g(a) = 1$ and a is added to S.
– Element b is spanned by a in M_1 and not spanned by any element in M_2. So we get $g(b) = w(b) - w_1^*(b) - w_2^*(b) = 1 + \varepsilon - 1 - 0 = \varepsilon$. As $\varepsilon > 0$, we add b to S, and set $w_1(b) = w_1(a) + \varepsilon = 1 + \varepsilon$ and $w_2(b) = \varepsilon$.
– Element c is not spanned by any element in M_1 but is spanned by $\{a, b\}$ in M_2. As b has the smallest w_2 weight, $w_2^*(c) = w_2(b) = \varepsilon$. So we have $g(c) = 2\varepsilon - w_1^*(c) - w_2^*(c) = 2\varepsilon - 0 - \varepsilon = \varepsilon > 0$, and we set $w_1(c) = \varepsilon$ and $w_2(c) = 2\varepsilon$ and add c to S.
– Element d is similar to c. We have $g(d) = 3\varepsilon - 0 - 2\varepsilon = \varepsilon > 0$ and so we set $w_1(d) = \varepsilon$ and $w_2(d) = 3\varepsilon$ and add d to S.

As the algorithm selected all the elements, we have $S = E$. It follows that the greedy algorithm on S (in the reverse order of when elements were added) will select d and c, after which the set is a maximal independent set in M_2. This gives a weight of 5ε, even though a and b both have weight at least 1, which shows that this algorithm does not guarantee any constant factor approximation. □

3.3 Analysis of Algorithm 2

We prove that Algorithm 2 has an approximation guarantee of 2.

Theorem 1 *Let S be the subset generated by Algorithm 2 on a stream E of elements, matroids $M_1 = (E, I_1)$, $M_2 = (E, I_2)$ and weight function $w : E \to \mathbb{R}_{\geq 0}$. Then there exists a subset $T \subseteq S$ independent in M_1 and in M_2 whose weight $w(T)$ is at least $w(S^*)/2$, where S^* denotes an optimal solution to the weighted matroid intersection problem.*

Throughout the analysis we fix the input matroids $M_1 = (E, I_1)$, $M_2 = (E, I_2)$, the weight function $w : R \to \mathbb{R}_{\geq 0}$, and the order of the elements in the stream. While Algorithm 2 only defines the weight functions w_1 and w_2 for the elements added to the set S, we extend them in the analysis by, for $i \in \{1, 2\}$, letting $w_i(e) = w_i^*(e)$ for the elements e not added to S.

We now prove Theorem 1 by showing that $g(S) \geq w(S^*)/2$ ((Lemma 3) and that there is a solution $T \subseteq S$ such that $w(T) \geq g(S)$ ((Lemma 4). In the proof of both these lemmas, we use the following properties of the computed set S.

Lemma 2 *Let S be the set generated by Algorithm 2 and $S' \subseteq S$ any subset. Consider one of the matroids M_i with $i \in \{1, 2\}$. There exists a subset $T' \subseteq S'$ that is independent in M_i, i.e., $T' \in I_i$, and $w_i(T') \geq g(S')$. Furthermore, the maximum weight independent set in M_i with respect to w_i over the whole ground set E can be selected to be a subset of S, i.e. $T_i \subseteq S$, and it satisfies $w_i(T_i) = g(S)$.*

Proof Consider matroid M_1 (the proof is identical for M_2) and fix $S' \subseteq S$. The set $T_1' \subseteq S'$ that is independent in M_1 and that maximizes $w_1(T_1')$ satisfies

$$w_1(T_1') = \int_0^\infty \mathrm{rank}(\{e \in T_1' \mid w_1(e) \geq \theta\})\, d\theta = \int_0^\infty \mathrm{rank}(\{e \in S' \mid w_1(e) \geq \theta\})\, d\theta \,.$$

The second equality follows from the fact that the greedy algorithm that considers the elements in decreasing order of weight is optimal for matroids and thus we have $\mathrm{rank}(\{e \in T_1' \mid w_1(e) \geq \theta\}) = \mathrm{rank}(\{e \in S' \mid w_1(e) \geq \theta\})$ for any $\theta \in \mathbb{R}$.

Now index the elements of $S' = \{e_1, e_2, \ldots, e_\ell\}$ in the order they were added to S by Algorithm 2 and let $S_j' = \{e_1, \ldots, e_j\}$ for $j = 0, 1, \ldots, \ell$ (where $S_0' = \emptyset$). By the above equalities and by telescoping,

$$w_1(T_1') = \sum_{i=1}^\ell \int_0^\infty \left(\mathrm{rank}(\{e \in S_i' \mid w_1(e) \geq \theta\}) - \mathrm{rank}(\{e \in S_{i-1}' \mid w_1(e) \geq \theta\}) \right) d\theta \,.$$

We have that $\mathrm{rank}(\{e \in S_i' \mid w_1(e) \geq \theta\}) - \mathrm{rank}(\{e \in S_{i-1}' \mid w_1(e) \geq \theta\})$ equals 1 if $w(e_i) \geq \theta$ and $e_i \notin \mathrm{span}(\{e \in S_{i-1}' \mid w_1(e) \geq \theta\})$ and it equals 0 otherwise. Therefore, by the definition of $w_1^*(\cdot)$, the gain $g(\cdot)$ and $w_1(e_i) = w_1^*(e_i) + g(e_i)$ in Algorithm 2 we have

$$w_1(T_1') = \sum_{i=1}^{\ell} \left[w_1(e_i) - \max \left(\{0\} \cup \{\theta : e_i \in \mathrm{span} \left(\{f \in S_{i-1}' \mid w_i(f) \geq \theta\} \right) \} \right) \right]$$

$$\geq \sum_{i=1}^{\ell} g(e_i) = g(S') .$$

The inequality holds because S_{i-1}' is a subset of the set S at the time when Algorithm 2 considers element e_i. Moreover, if $S' = S$, then S_{i-1}' equals the set S at that point and so we then have $w_1^*(e_i) = \max \left(\{0\} \cup \{\theta : e_i \in \mathrm{span} \left(\{f \in S_{i-1}' \mid w_i(f) \geq \theta\} \right) \} \right)$, which implies that the above inequality holds with equality in that case. We can thus also conclude that a maximum weight independent set $T_1 \subseteq S$ satisfies $w_1(T_1) = g(S)$. Finally, we can observe that T_1 is also a maximum weight independent set over the whole ground set since we have $\mathrm{rank}(\{e \in S \mid w_1(e) \geq \theta\}) = \mathrm{rank}(\{e \in E \mid w_1(e) \geq \theta\})$ for every $\theta > 0$, which holds because, by the extension of w_1, an element $e \notin S$ satisfies $e \in \mathrm{span}(\{f \in S : w_1(f) \geq w_1(e)\})$. $\qquad \square$

We can now relate the gain of the elements in S with the weight of an optimal solution.

Lemma 3 *Let S be the subset generated by Algorithm 2. Then $g(S) \geq w(S^*)/2$.*

Proof We first observe that $w_1(e) + w_2(e) \geq w(e)$ for every element $e \in E$. Indeed, for an element $e \in S$, we have by definition $w(e) = g(e) + w_1^*(e) + w_2^*(e)$, and $w_i(e) = g(e) + w_i^*(e)$, so $w_1(e) + w_2(e) = 2g(e) + w_1^*(e) + w_2^*(e) = w(e) + g(e) > w(e)$. In the other case, when $e \notin S$ then $w_1^*(e) + w_2^*(e) \geq w(e)$, and $w_i(e) = w_i^*(e)$, so automatically, $w_1(e) + w_2(e) \geq w(e)$.

The above implies that $w_1(S^*) + w_2(S^*) \geq w(S^*)$. On the other hand, by Lemma 2, we have $w_i(T_i) \geq w_i(S^*)$ (since T_i is a max weight independent set in M_i with respect to w_i) and $w_i(T_i) = g(S)$, thus $g(S) \geq w_i(S^*)$ for $i = 1, 2$. $\qquad \square$

We finish the Proof of Theorem 1 by proving that there is a $T \subseteq S$ independent in both M_1 and M_2 such that $w(T) \geq g(S)$. As described in Sect. 3.2.1, we cannot select T using the greedy method. Instead, we select T using the concept of kernels studied in [8].

Lemma 4 *Let S be the subset generated by Algorithm 2. Then for any subset $S' \subseteq S$, there exists a subset $T \subseteq S'$ independent in M_1 and in M_2 such that $w(T) \geq g(S')$.*

Proof Consider one of the matroids M_i with $i \in \{1, 2\}$ and define a total order $<_i$ on E such that $e <_i f$ if $w_i(e) > w_i(f)$ or if $w_i(e) = w_i(f)$ and e appeared later in the stream than f. The pair $(M_i, <_i)$ is known as an ordered matroid. We further say that a subset E' of E dominates element e of E if $e \in E'$ or there is a subset $C_e \subseteq E'$ such that $e \in \mathrm{span}(C_e)$ and $c < e$ for all elements c of C_e. The set of elements dominated by E' is denoted by $D_{M_i}(E')$. Note that if E' is an independent set, then the greedy algorithm that considers the elements of $D_{M_i}(E')$ in the order $<_i$ selects exactly the elements E'.

Theorem 2 in [8] says that for two ordered matroids $(M_1, <_1)$, $(M_2, <_2)$ there always is a set $K \subseteq E$, which is referred to as a $M_1 M_2$-kernel, such that

– K is independent in both M_1 and in M_2; and
– $D_{M_1}(K) \cup D_{M_2}(K) = E$.

We use the above result on M_1 and M_2 restricted to the elements in S'. Specifically we select $T \subseteq S'$ to be the kernel such that $D_{M_1}(T) \cup D_{M_2}(T) = S'$. Let $S_1 = D_{M_1}(T)$ and $S_2 = D_{M_2}(T)$. By Lemma 2, there exists a set $T' \subseteq S_1$ independent in M_1 such that $w_1(T') \geq g(S_1)$. As noted above, the greedy algorithm that considers the element of S_1 in the order $<_i$ (decreasing weights) selects exactly the elements in T. It follows by the optimality of the greedy algorithm for matroids that T is a maximum weight independent set in S_1 for M_1 with weight function w_1, which in turn implies $w_1(T) \geq g(S_1)$. In the same way, we also have $w_2(T) \geq g(S_2)$. By definition, for any $e \in S'$, we have $w(e) = w_1(e) + w_2(e) - g(e)$. Together, we have $w(T) = w_1(T) + w_2(T) - g(T) \geq g(S_1) + g(S_2) - g(T)$. As elements from T are in both S_1 and S_2, and all other elements are in at least one of both sets, we have $g(S_1) + g(S_2) \geq g(S') + g(T)$, and thus $w(T) \geq g(S')$. □

4 Making the algorithm memory efficient

We now modify Algorithm 2 to only select elements with a significant gain, parametrized by $\alpha > 1$, and delete elements if we have too many in memory, parametrized by a real number y. If α is close enough to 1 and y is large enough, then Algorithm 3 is very close to Algorithm 2, and allows for a similar analysis. This method is very similar to the one used in [14] and [9], but our analysis is quite different.

More precisely, we take an element e only if $w(e) > \alpha(w_1^*(e) + w_2^*(e))$ instead of $w(e) > w_1^*(e) + w_2^*(e)$, and we delete elements if the ratio between two g weights becomes larger than y ($\frac{g(e)}{g(e')} > y$). For technical purposes, we also need to keep independent sets T_1 and T_2 which maximize the weight functions w_1 and w_2 respectively. If an element with small g weight is in T_1 or T_2, we do not delete it, as this would modify the w_i-weights and selection of coming elements. We show that this algorithm is a semi-streaming algorithm with an approximation guarantee of $(2 + \varepsilon)$ for an appropriate selection of the parameters (see Lemma 6 for the space requirement and Theorem 2 for the approximation guarantee).

Lemma 5 *Let S be the subset generated by Algorithm 3 with $\alpha \geq 1$ and $y = \infty$. Then $w(S^*) \leq 2\alpha g(S)$.*

Proof We define $w_\alpha : E \to \mathbb{R}$ by $w_\alpha(e) = w(e)$ if $e \in S$ and $w_\alpha(e) = \frac{w(e)}{\alpha}$ otherwise. By construction, Algorithms 2 and 3 give the same set S, and the same weight function g for this modified weight function. By Lemma 3, $w_\alpha(S^*) \leq 2g(S)$. On the other hand, $w(S^*) \leq \alpha w_\alpha(S^*)$. □

Lemma 6 *Let S be the subset generated by Algorithm 3 with $\alpha = 1 + \varepsilon$ and $y = \frac{\min(r_1, r_2)}{\varepsilon^2}$ and S^* be a maximum weight independent set, where r_1 and r_2 are the ranks of M_1 and M_2 respectively. Then $w(S^*) \leq 2(1 + 2\varepsilon + o(\varepsilon))g(S)$. Furthermore, at any point of time, the size of S is at most $r_1 + r_2 + \min(r_1, r_2) \log_\alpha(\frac{\alpha y}{\varepsilon})$.*

Proof We first prove that the generated set S satisfies $w(S^*) \leq 2(1+2\varepsilon+o(\varepsilon))g(S)$ and we then verify the space requirement of the algorithm, i.e., that it is a semi-streaming algorithm.

Let us call S' the set of elements selected by Algorithm 3, including the elements deleted later. By Lemma 5, we have $2\alpha g(S') \geq w(S^*)$, so if we prove that $g(S') - g(S) \leq \alpha\varepsilon g(S) = (\varepsilon + o(\varepsilon))g(S)$, we are done. We set $i \in \{1, 2\}$ to be the index of the matroid with smaller rank.

In our analysis, it will be convenient to think that the algorithm maintains the maximum weight independent set T_i of M_i throughout the stream. At the arrival of an element e that is added to S, we have that the set T_i is updated as follows. If $T_i \cup \{e\} \in I_i$ then e is simply added to T_i. Otherwise, before updating T_i, there is an element $e^* \in T_i$ such that $w_i(e^*) = w_i^*(e)$ and $T_i \backslash \{e^*\} \cup \{e\}$ is maximum weight independent set in M_i with respect to w_i. Thus we can speak of elements which are *replaced* be another element in T_i. By construction, if e replaces f in T_i, then $w_i(e) > \alpha w_i(f)$.

We can now divide the elements of S' into stacks in the following way: If e replaces an element f in T_i, then we add e on top of the stack containing f, otherwise we create a new stack containing only e. At the end of the stream, each element $e \in T_i$ is in a different stack, and each stack contains exactly one element of T_i, so let us call S_e' the stack containing e whenever $e \in T_i$. We define S_e to be the restriction of S_e' to S. In particular, each element from S' is in exactly one S_e' stack, and each element from S is in exactly one S_e stack. For each stack S_e', we set $e_{del}(S_e')$ to be the highest weight element of S_e' which was removed from S. By construction, $g(S_e') - g(S_e) \leq w_i(e_{del}(S_e'))$. On the other hand, $w_i(f) < \frac{1+\varepsilon}{\varepsilon}g(f)$ for any element $f \in S'$ (otherwise we would not have selected it), so $g(S_e') - g(S_e) < \frac{1+\varepsilon}{\varepsilon}g(e_{del}(S_e'))$. As $e_{del}(S_e')$ was removed from S, we have $g(e_{del}(S_e')) < \frac{g_{max}}{y}$ where $g_{max} = \max_{e \in S} g(e)$. As there are exactly r_i stacks, we get $g(S') - g(S) < r_i \frac{g_{max}\varepsilon^2(1+\varepsilon)}{r_i\varepsilon} = \varepsilon(1 + \varepsilon)g_{max} \leq (\varepsilon + o(\varepsilon))g(S)$.

We now have to prove that the algorithm fits the semi-streaming criteria. In fact, the size of S never exceeds $r_1 + r_2 + r_i \log_\alpha(\frac{\alpha y}{\varepsilon})$. By the pigeonhole principle, if S has at least $r_i \log_\alpha(\frac{\alpha y}{\varepsilon})$ elements, then there is at least one stack S_e which has at least $\log_\alpha(\frac{\alpha y}{\varepsilon})$ elements. By construction, the w_i weight increases by a factor of at least α each time we add an element on the same stack, so the w_i weight difference between the lowest and highest element on the biggest stack would be at least $\frac{\alpha y}{\varepsilon}$. As $w_i(f) < \frac{1+\varepsilon}{\varepsilon}g(f)$, the g weight difference would be at least y, and we would remove the lowest element, unless it was in T_1 or T_2. □

Theorem 2 *Let S be the subset generated by running Algorithm 3 with $\alpha = 1 + \varepsilon$ and $y = \frac{\min(r_1, r_2)}{\varepsilon^2}$. Then there exists a subset $T \subseteq S$ independent in M_1 and in M_2 such that $w(T) \geq g(S)$. Furthermore, T is a $2(1 + 2\varepsilon + o(\varepsilon))$-approximation for the intersection of two matroids.*

Proof Let S^* be a maximum weight independent set. By Lemma 6, we have $2(1 + 2\varepsilon + o(\varepsilon)g(S) \geq w(S^*)$. Let S' be the set of elements selected by Algorithm 3, including the elements deleted later. As long as we do not delete elements from T_1 or T_2, Algorithm 2 restricted to S' will select the same elements, with the same weights, so we can consider S' to be generated by Algorithm 2. Since $S \subseteq S'$, we now observe that by Lemma 4, we can find an independent set $T \subseteq S$ such that $w(T) \geq g(S)$. □

Fig. 2 Consider the example on a bipartite graph where edges arrive in the order e_a, e_b, e_c. It is easy to see that the set S formed by Algorithm 3 contains only the edge e_a of weight 1 whereas the optimal matching consists of taking edges e_b, e_c of combined weight 2α
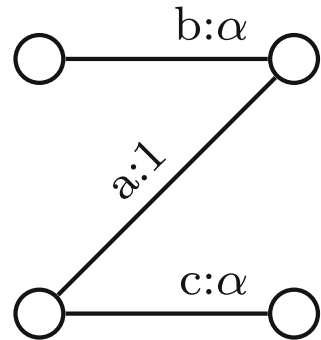
Remark 1 It is easy to construct examples where the set S only contains a 2α-approximation (for an example, see Fig. 2 involving a bipartite graph), so our analysis is tight up to ε.

Remark 2 The techniques of this section can also be used in the case when the ranks of the matroids are unknown. Specifically, the algorithm can maintain the stacks created in the Proof of Lemma 6 and allow for an error $\varepsilon/2$ in the first two stacks created, an error of $\varepsilon/4$ in the next 4 stacks, and in general an error of $\varepsilon/2^i$ in the next 2^i stacks by having a y value specific to each stack. The idea of constructing such a geometric sequence is to have a total error of at most ε. We explain this in detail in Appendix A.

Algorithm 3 Semi-streaming adaptation of Algorithm 2

Input: A stream of the elements and 2 matroids (which we call M_1, M_2) on the same ground set E, a real number $\alpha > 1$ and a real number y.

Output: A set $X \subseteq E$ that is independent in both matroids.

    Whenever we write an assignment of a variable with subscript i, it means we do it for $i = 1, 2$.

    $S \leftarrow \emptyset$

    for element e in the stream **do**

        calculate $w_i^*(e) = \max\left(\{0\} \cup \{\theta : e \in \mathrm{span}_{M_i}\left(\{f \in S \mid w_i(f) \geq \theta\}\right)\}\right)$.

        if $w(e) > \alpha(w_1^*(e) + w_2^*(e))$ **then**

            $g(e) \leftarrow w(e) - w_1^*(e) - w_2^*(e)$

            $S \leftarrow S \cup \{e\}$

            $w_i(e) \leftarrow g(e) + w_i^*(e)$

            Let T_i be a maximum weight independent set of M_i with respect to w_i.

            Let $g_{max} = \max_{e \in S} g(e)$

            Remove all elements $e' \in S$, such that $y \cdot g(e') < g_{max}$ and $e' \notin T_1 \cup T_2$ from S.

        end if

    end for

    return a maximum weight set $T \subseteq S$ that is independent in M_1 and M_2

---

5 Extension to submodular functions

In this section, we consider the problem of submodular matroid intersection in the semi-streaming model. We first give the definition of a submodular function and then formally define our problem.

Definition 1 (*Submodular function*) A set function $f : 2^E \to \mathbb{R}$ is submodular if it satisfies that for any two sets $A, B \subseteq E$, $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$. For any two sets $A, B \subseteq E$, let $f(A \mid B) := f(A \cup B) - f(B)$. For any element e and set $A \subseteq E$, let $f(e \mid A) := f(A \cup \{e\} \mid A)$. Now, an equivalent and more intuitive definition for f to be submodular is that for any two sets $A \subseteq B \subseteq E$, and $e \in E \setminus B$, it holds that $f(e \mid A) \geq f(e \mid B)$. The function f is called monotone if for any element $e \in E$ and set $A \subseteq E$, it holds that $f(e \mid A) \geq 0$.

Given the above definition, we can formally define our problem now. Here, we are given an oracle access to two matroids $M_1 = (E, I_1)$, $M_2 = (E, I_2)$ on a common ground set E and an oracle access to non-negative submodular function $f : 2^E \to \mathbb{R}_{\geq 0}$ on the powerset of the elements of the ground set. The goal is to find a subset $X \subseteq E$ that is independent in both matroids, i.e., $X \in I_1$ and $X \in I_2$, and whose weight $f(X)$ is maximized.

Our Algorithm 4 is a straightforward generalization of Algorithms 1 and 2 of [11]. Since, the weight of an element e now depends on the underlying set that it would be added to, we (naturally) define the weight of e to be the additional value e provides after adding it to set S, i.e. $w(e) = f(e \mid S)$. If e provides S a good enough value, i.e, $f(e \mid S) \geq \alpha(w_1^*(e) + w_2^*(e))$, we add it to set S but with a probability q now. This probability q is the most important difference between Algorithms 3 and 4. This is a trick that we borrow from the Algorithm 1 of [11] which is useful when f is non-monotone because of the following Lemma 2.2 of [1].

Lemma 7 *(Lemma 2.2 in [1]) Let $h : 2^E \to \mathbb{R}_{\geq 0}$ be a non-negative submodular function, and let S be a a random subset of E containing every element of M with probability at most q(not necessarily independently), then $E[h(S)] \geq (1-q)h(\emptyset)$.*

In our proof, we can relate the weight of the set that we pick and the value $f(S^* \cup S_f)$ where S_f denotes the elements in the stack when algorithm stops and S^* denotes the set of optimum elements. If the function f is monotone, this is sufficient as $f(S^* \cup S_f) \geq f(S^*)$. This, however, is not true if function f is non-monotone. Here, one can use the Lemma 7 with the function $h(T) = f(T \cup S^*)$. This enables us to conclude that $\mathbb{E}[f(S^* \cup S_f)] \geq (1-q)f(S^*)$.

5.1 Analysis of Algorithm 4

We extend the analysis of Sect. 4 by using ideas from [11] to analyze our algorithm. Before going into the technical details, we give a brief overview of our analysis. For sake of intuition, we assume that the Algorithm 4 does not delete elements and also does not skip elements with probability $1 - q$. Then, due to the fact that the weight of an element e is the additional value it provides to the current set S, one can relate

---

Algorithm 4 Extension of Algorithm 3 to submodular functions

Input: A stream of the elements and 2 matroids (which we call M_1, M_2) on the same ground set E, a submodular function $f : 2^E \mapsto \mathbb{R}$, a real number $\alpha \geq 1$, a real number q such that $0 \leq q \leq 1$ and a real number y.

Output: A set $X \subseteq E$ that is independent in both matroids.
Whenever we write an assignment of a variable with subscript i, it means we do it for $i = 1, 2$.
$S \leftarrow \emptyset$
for element e in the stream **do**
    calculate $w_i^*(e) = \max \left(\{0\} \cup \{\theta : e \in \text{span}_{M_i} (\{f \in S \mid w_i(f) \geq \theta\})\} \right)$.
    if $f(e \mid S) > \alpha(w_1^*(e) + w_2^*(e))$ **then**
        with probability $1 - q$, **continue**;   {//skip e with probability $1 - q$.}
        $g(e) \leftarrow f(e \mid S) - w_1^*(e) - w_2^*(e)$
        $S \leftarrow S \cup \{e\}$
        $w_i(e) \leftarrow g(e) + w_i^*(e)$
        Let T_i be a maximum weight independent set of M_i with respect to w_i.
        Let $g_{max} = \max\limits_{e \in S} g(e)$
        Remove all elements $e' \in S$, such that $y \cdot g(e') < g_{max}$ and $e' \notin T_1 \cup T_2$ from S.
    end if
end for
return a maximum weight set $T \subseteq S$ that is independent in M_1 and M_2

---

the weight of the independent set picked with the weight of the optimal solution given the set S_f i.e., $f(S^* \mid S_f)$ by basically using the analysis of the previous section. However, this is not enough as the weight of the optimal solution is $f(S^*)$. But, we can still relate the gain of S_f to $f(S_f)$ similar to [11], which helps us relate $f(S^* \cup S_f)$ and weight of our solution. In order to extend it to the case when elements are skipped with probability $1 - q$, we show the above to hold in expectation similar to [11], which is helpful for dealing with non-monotone functions because of Lemma 7. Finally, we remark that one can use an analysis similar to Sect. 4, to show that the effect of deleting elements does not affect the weight of solution by a lot.

Let S_f denote the set S generated when the algorithm stops and S'_f denote the union of S_f and the elements that were deleted by the algorithm. For sake of analysis, we define the weight function $w : E \to \mathbb{R}$ of an element e to be the additional value it provided to the set S when it appeared in the stream, i.e., $w(e) = f(e \mid S)$. Like before, we extend the definition of weight functions w_1 and w_2 for an element e that is not added to S as $w_i(e) = w_i^*(e)$ for $i \in \{1, 2\}$. We note here that all the functions defined above are random variables which depend on the internal randomness of the algorithm. Unless we explicitly mention it, we generally talk about statements with respect to any fixed realization of the internal random choices of the algorithm.

In our analysis, we will prove properties about our algorithm that are already proven for Algorithms 2 and 3 in the previous sections. Our proof strategy will be simply running Algorithm 2 or 3 with the appropriate weight function which will mimic running our original algorithm. Hence, we will prove these statements in a black-box fashion. A weight function that we will use repeatedly in our proofs is $w' : E \to \mathbb{R}_{\geq 0}$ where $w'(e) = w(e)$ if $e \in S'_f$, otherwise $w'(e) = 0$. This basically has the effect of discarding elements not in S'_f i.e, elements that were never picked by the algorithm either because they did not provide a good enough value or because they did but were still skipped.

Lemma 8 *Consider the set S'_f which is the union of S_f generated by the Algorithm 4 and the elements it deletes. Then a maximum weight independent set in M_i for $i \in \{1, 2\}$ over the whole ground set E can be selected to be a subset of S'_f, i.e. $T_i \subseteq S'_f$ and it satisfies $w_i(T_i) = g(S'_f)$.*

Proof Consider running the Algorithm 2 with weight function w'. Notice that doing this generates a stack containing exactly the elements in the set S'_f and exactly the same functions w_1, w_2 and g. Now by applying Lemma 2, we get our result. □

We prove the following lemma similar to [11] which relates the gain of elements in S'_f to the weight of the optimal solution given the set S'_f i.e, $f(S^* \mid S'_f)$. Notice that the lemma below holds only in expectation for $q \neq 1$.

Lemma 9 *Denote the set S'_f which is the union of S_f generated by the Algorithm 4 with $q \in \{1/(2\alpha + 1), 1\}$ and the elements it deletes. Then, $\mathbb{E}[f(S^* \mid S'_f)] \leq 2\alpha\mathbb{E}[g(S'_f)]$.*

Proof We first prove the lemma for $q = 1$ as the proof is easier than that for $q = 1/(2\alpha + 1)$. Consider running the Algorithm 2 with weight function $w'' : E \to \mathbb{R}_{\geq 0}$ defined as follows. If $e \in S'_f$, $w''(e) = w(e)$, else $w''(e) = w(e)/\alpha$. Notice that doing this generates a stack containing exactly the elements in the set S'_f and exactly the same functions w_1, w_2 and g. Now by applying Lemma 3, we get that $w(S^*) \leq 2\alpha g(S'_f)$. By submodularity, we get $f(S^* \mid S'_f) \leq 2\alpha g(S'_f)$.

Now, we prove the lemma for $q = 1/(2\alpha + 1)$. We first define $\lambda : E \to \mathbb{R}$ for an element $e \in E$ as $\lambda(e) = f(e \mid S'_f)$. Notice that, by submodularity of f and definition of λ, we have $f(S^* \mid S'_f) \leq \lambda(S^*)$. Hence, it suffices to prove $\mathbb{E}[\lambda(S^*)] \leq 2\alpha\mathbb{E}[g(S'_f)]$. We prove this below.

Let the event that the element $e \in E$ does not give us a good enough value i.e, it satisfies $\alpha(w_1^*(e) + w_2^*(e)) \geq w(e)$ be R_e. We have two cases to consider now.

1. The first is when R_e is true. Then, for any fixed choice of randomness of the algorithm for which R_e is true, we argue as follows. By definition, $w_i(e) = w_i^*(e)$. Hence, $\alpha(w_1(e) + w_2(e)) \geq w(e)$. Also, $w(e) = f(e \mid S)$ where S is the stack when e appeared in the stream. As $S \subseteq S'_f$, by submodularity and definition of λ, we get that $w(e) \geq \lambda(e)$. Hence, we also get that $\alpha\mathbb{E}[w_1(e) + w_2(e)|R_e] \geq \mathbb{E}[\lambda(e)|R_e]$.

2. The second is when R_e is false. Then, for any fixed choice of randomness of the algorithm for which R_e is false, we argue as follows. Here, e is picked with probability q given the set S at the time e appeared in the stream. If we pick e, then $w_1(e) + w_2(e) = g(e) + w_1^*(e) + g(e) + w_2^*(e) = 2w(e) - w_1^*(e) - w_2^*(e)$. Otherwise, if we do not pick e, then $w_1(e) + w_2(e) = w_1^*(e) + w_2^*(e)$. Hence, the expected value of $w_1(e) + w_2(e)$ satisfies,

$$\mathbb{E}[w_1(e) + w_2(e)|\neg R_e, S] = 2qw(e) + (1 - 2q)(w_1^*(e) + w_2^*(e)) \geq 2qw(e).$$

The last inequality follows as we have $q = 1/(2\alpha + 1) \leq 1/2$. By the choice of q and submodularity, we get that $\alpha\mathbb{E}[w_1(e) + w_2(e)|\neg R_e, S] \geq 2q\alpha w(e) = (1-q)w(e) \geq (1-q)\lambda(e)$. By law of total expectation and conditioned on R_e not taking place we get, $\alpha\mathbb{E}[w_1(e) + w_2(e)|\neg R_e] \geq \mathbb{E}[\lambda(e)|\neg R_e]$.

Finally by the law of total expectation and the points 1 and 2, we obtain that $\alpha \mathbb{E}[w_1(e) + w_2(e)] \geq \mathbb{E}[\lambda(e)]$ holds for any element $e \in E$. Applying this to elements of S^*, we get that $\alpha \mathbb{E}[w_1(S^*) + w_2(S^*)] \geq \mathbb{E}[\lambda(S^*)]$. On the other hand, by Lemma 8, we have $w_i(T_i) \geq w_i(S^*)$ (since T_i is a max weight independent set in M_i with respect to w_i) and $w_i(T_i) = g(S'_f)$, thus $g(S'_f) \geq w_i(S^*)$ for $i = 1, 2$. Hence, we get that $\mathbb{E}[\lambda(S^*)] \leq 2\alpha \mathbb{E}[g(S'_f)]$. $\qquad \square$

Since, we would like the relate the gain of elements in S'_f to the optimal solution we bound the value of $f(S'_f)$ in terms of the gain below similar to [11].

Lemma 10 *Consider the set S'_f which is the union of S_f generated by the Algorithm 4 and the elements it deletes. Then, $g(S'_f) \geq (1 - 1/\alpha) f(S'_f)$.*

Proof By definition, any element $e \in S'_f$, should have satisfied $w(e) \geq \alpha(w_1^*(e) + w_2^*(e))$. Hence, $g(e) \geq w(e) - w(e)/\alpha$. Summing over all elements in S'_f, we get $g(S'_f) \geq (1 - 1/\alpha)w(S'_f) \geq f(S'_f)$ where last inequality (not an equality as S'_f also contains deleted elements) follows by definition of w and submodularity of f. $\quad \square$

Our algorithm only has the set S_f and not S'_f which also includes the deleted elements. Hence, in our next lemma, we prove that the gain of elements in these two sets is roughly the same.

Lemma 11 *Consider the set S'_f which is the union of S_f generated by running the Algorithm 4 with $\alpha > 1$, $y = \min(r_1, r_2)/\delta^2$ for any δ, such that $0 < \delta \leq \alpha - 1$ and the elements it deletes. Here, r_i is the rank of M_i for $i \in \{1, 2\}$. Then, $g(S'_f) - g(S_f) \leq \delta \alpha g(S_f)$. Moreover, at any point during the execution, S contains at most $r_1 + r_2 + \min(r_1, r_2) \log_\alpha(\frac{\alpha y}{\alpha - 1})$ elements.*

Proof Consider running the Algorithm 3 with weight function w'. Notice that doing this generates a stack containing exactly the elements as in the set S_f, exactly the same set of deleted elements and exactly the same functions w_1, w_2 and g. Moreover, this generates the exact same stacks as the Algorithm 4 at every point of execution. Now by the Proof of Lemma 6, we get our result. $\qquad \square$

Lastly, we prove that there exists a set T that is independent in both matroids and has a weight at least the gain of the elements in S_f.

Lemma 12 *Let S_f be the subset generated by Algorithm 4. Then there exists a subset $T \subseteq S$ independent in M_1 and in M_2 such that $w(T) \geq g(S_f)$.*

Proof Consider running the Algorithm 3 with weight function w'. Recall that for any element $e \in S'_f$, $w'(e) = w(e)$, otherwise $w'(e) = 0$. Notice that doing this generates a stack containing exactly the elements as in the set S_f and exactly the same functions w_1, w_2 and g. The result follows by Theorem 2. $\qquad \square$

Now, we have all the lemmas to prove our main theorem which we state below.

Theorem 3 *The subset S_f generated by Algorithm 4 with $\alpha > 1$, $q \in \{1/(2\alpha + 1), 1\}$ and $y = \min(r_1, r_2)/\delta^2$ for any δ, such that $0 < \delta \leq \alpha - 1$ contains a $(4\alpha^2 - 1)/(2\alpha - 2) + O(\delta)$ approximation in expectation for the intersection of two matroids with respect to a non-monotone submodular function f. This is optimized by taking $\alpha = 1 + \sqrt{3}/2$, resulting in an approximation ratio of $4 + 2\sqrt{3} + O(\delta) \sim 7.464$. Moreover, the same algorithm run with $q = 1$ and $y = \min(r_1, r_2)/\delta^2$ is $(2\alpha + \alpha/(\alpha - 1)) + O(\delta)$ approximate if f is monotone. This is optimized by taking $\alpha = 1 + 1/\sqrt{2}$, which yields a $3 + 2\sqrt{2} + O(\delta) \sim 5.828$ approximation.*

Proof By Lemmas 9 and 10, we have that $2\alpha \mathbb{E}[g(S'_f)] \geq \mathbb{E}[f(S^* \mid S'_f)]$ and $g(S'_f)(\alpha/(\alpha - 1)) \geq f(S'_f)$. Combining them, we get,

$$(2\alpha + \alpha/(\alpha - 1))\mathbb{E}[g(S'_f)] \geq \mathbb{E}[f(S'_f) + f(S^* \mid S'_f)] = \mathbb{E}[f(S^* \cup S'_f)].$$

By Lemma 11, we also get that $g(S'_f) - g(S_f) \leq \delta\alpha g(S_f)$. This gives us that

$$(2\alpha + \alpha/(\alpha - 1))(1 + \delta\alpha)\mathbb{E}[g(S_f)] \geq \mathbb{E}[f(S^* \cup S'_f)].$$

Now, by Lemma 12, there exists a subset $T \subseteq S_f$ independent in M_1 and M_2 such that $w(T) \geq g(S_f)$. By definition of w, and submodularity of f, we get that $f(T) \geq w(T)$. This in turn implies, $f(T) \geq g(S_f)$. This gives us that

$$(2\alpha + \alpha/(\alpha - 1))(1 + \delta\alpha)\mathbb{E}[f(T)] \geq \mathbb{E}[f(S^* \cup S'_f)].$$

Notice that the above inequality also holds if $q = 1$ as all the above arguments also work if $q = 1$. Hence, if f is monotone, we get $f(S^* \cup S_f) \geq f(S^*)$ which gives us our desired inequality by rearranging terms. However, if f is non-monotone one has to work a little more which we show below.

To deal with the case when f is non-monotone, we use Lemma 7 and take $h(T) = f(S^* \cup T)$ for any $T \subseteq E$ within the lemma statement, to get that $\mathbb{E}[f(S^* \cup S'_f)] \geq (1 - q)f(S^*)$ as every element of E appears in S'_f with probability at most q. Putting everything together, we get that

$$(2\alpha + \alpha/(\alpha - 1))(1 + \delta\alpha)\mathbb{E}[f(T)] \geq (1 - q)f(S^*).$$

Now, substituting the value of $q = 1/(2\alpha + 1)$ and rearranging terms, we get the desired inequality. $\qquad\square$

Remark 3 We can exactly match the approximation ratios in [11] i.e, without the extra additive factor of $O(\delta)$ by not deleting elements. Moreover, S stores at most $O(\min(r_1, r_2) \log_\alpha |E|)$ elements at any point if we assume that values of f are polynomially bounded in $|E|$, an assumption that the authors in [11] make.

6 More than two matroids

We can easily extend Algorithm 3 to the intersection of k matroids (see Algorithm 5 for details). Most results remain true, in particular, we can have $kg(S) \geq (1+\varepsilon)w(S^*)$ by carefully selecting α and y. The only part which does not work is the selection of the independent set from S. Indeed, matroid kernels are very specific to two matroids. We now prove that a similar approach fails, by proving that the logical generalization of kernels to 3 matroids where one tries to define the order given by the w_i weights is wrong and that a counter-example can arise from Algorithm 5. Thus, any attempt to find a $k + \varepsilon$ approximation using our techniques must bring some fundamentally new idea. Still, we conjecture that the generated set S contains such an approximation.

Proposition 1 *There exists a set S and 3 matroids $(S, I_1), (S, I_2), (S, I_3)$ such that there does not exist a set $T \subseteq S$ such that $S = D_{M_1}(T) \cup D_{M_2}(T) \cup D_{M_3}(T)$ (see Lemma 4 for a definition of $D_{M_i}(T)$) and T is independent in M_1, M_2 and M_3 where $<_i$ is given by w_i generated by Algorithm 5 (for α sufficiently small).*

Proof We set $S = \{a, x, y, z, b\}$, which are given in this order to Algorithm 5. We now define I_1, I_2, I_3 in the following way. A set of 2 elements is in I_i if and only if:

- In I_1 if it is not $\{a, x\}$
- In I_2 if it is not $\{a, y\}$
- In I_3 if it is not $\{a, z\}$

  A set of 3 elements is in I_i if and only if each of its subsets of 2 elements is in I_i and:
- In I_1 if it contains z
- In I_2 if it contains x
- In I_3 if it contains y

A set of 4 elements is not in I_i.

Let us verify that these constraints correspond to matroids. As the problem is symmetrical, it is sufficient to verify that M_1 is a matroid. The 3 element independent sets in M_1 are exactly $\{y, z, b\}$, $\{x, z, b\}$, $\{x, y, z\}$, $\{a, z, b\}$ and $\{a, y, z\}$. Now we consider $X, Y \in I_1$ with $|X| < |Y|$. We should find $e \in Y \backslash X$ such that $X \cup \{e\} \in I_1$. If $X = \emptyset$, take any element from Y. If X is a singleton, then there are two cases: either it is one of $X \subseteq \{a, x\}$, or it is not. In any case, Y contains at most one element from $\{a, x\}$. As it contains at least two elements, Y has to contain an element from $\{y, z, b\}$. In the first case, we can add any of these to X to get an independent set. In the second case, $X \subseteq \{y, z, b\}$, so we can add any element to X and it will remain independent, so just pick any element from $Y \backslash X$. If X contains two elements, then Y is one of the sets from the list above. In particular, it contains z. If $z \notin X$, then we can add z to X. Otherwise, either $X \subseteq \{y, z, b\}$, in which case we can add any element, or X is $\{a, z\}$ or $\{x, z\}$. In either case, Y must contain an element from $\{y, b\}$, which we can add to X.

We now set the weights $w(a) = 1$, $w(x) = w(y) = w(z) = 3$ and $w(b) = 8$ and run Algorithm 5.

Algorithm 5 Extension of Algorithm 3 to k matroids

Input: A stream of the elements and k matroids (which we call $M_1, \ldots, M_k$) on the same ground set E, a real number $\alpha > 1$ and a real number y.
Output: A set $S \subseteq E$ of "saved" elements.
  When we write an assignment of a variable with subscript i, it means we do it for $i = 1, \ldots, k$.
  $S \leftarrow \emptyset$
  for element e in the stream **do**
    calculate $w_i^*(e) = \max \left(\{0\} \cup \{\theta : e \in \operatorname{span}_{M_i}(\{f \in S \mid w_i(f) \geq \theta\})\} \right)$.

    if $w(e) > \alpha \sum_{i=1}^{k} w_i^*(e)$ **then**
      $g(e) \leftarrow w(e) - \sum_{i=1}^{k} w_i^*(e)$
      $S \leftarrow S \cup \{e\}$
      $w_i(e) \leftarrow g(e) + w_i^*(e)$
      Let T_i be a maximum weight independent set of M_i with respect to w_i.
      Let $g_{max} = \max_{e \in S} g(e)$
      Remove all elements $e' \in S$, such that $y \cdot g(e') < g_{max}$ and $e' \notin \bigcup_{i=1}^{k} T_i$ from S.
    end if
  end for

- Element a has weight 1, and $\{a\}$ is independent in M_1, M_2 and M_3, so we set $w_1(a) = w_2(a) = w_3(a) = g(a) = 1$ and a is added to S.
- Element x is spanned by a in M_1, and not spanned by any element in M_2 and M_3, so we get $g(x) = w(x) - w_1^*(x) - w_2^*(x) - w_3^*(x) = 3 - 1 - 0 - 0 = 2$. As $2 > 0$, we add x to S. We also set $w_1(x) = 3$ and $w_2(x) = w_3(x) = 2$.
- Element y and z are very similar to x.
- Element b is spanned in all three matroids by the elements of w_i weight at least 2. On the other hand, b is not spanned in any matroid by the elements of w_i weight strictly bigger than 2, so $w_i^*(b) = 2$ for $i = 1, 2, 3$, thus $g(b) = 8 - 2 - 2 - 2 = 2$ and $w_i(b) = 2 + 2 = 4$ for every i.

To recapitulate, we have $w_1(a) = 1$, $w_1(x) = 3$, $w_1(y) = w_1(z) = 2$, $w_1(b) = 4$ and the w_2 and w_3 weights are similar, with y respectively z being heavier.

Let us assume for a contradiction that T is a solution to the problem.

T must contain b, as it is the heaviest element in every matroid.

If T contains a, then it cannot contain any of x, y, z, otherwise it would not be independent in one of the matroids, so we would have $T \subseteq \{a, b\}$. But x has to be in at least one $D_{M_i}(T)$, and the set $\{x, b\}$ is independent in every matroid, and has a bigger weight than $\{a, b\}$, so x would not be in $D_{M_i}(T)$. Thus T cannot contain a.

As the problem is symmetrical for $\{x, y, z\}$, it is sufficient to test $T = \{z, b\}$, $T = \{y, z, b\}$ and $T = \{x, y, z, b\}$. The last two are not in I_2, so the only remaining possibility is $T = \{z, b\}$. But then y is not in D_{M_1} or D_{M_3} because $\{z, b, y\}$ is independent in M_1 and M_3, and it is not in D_{M_2} because $w_2(y) > w_2(z) \Leftrightarrow y <_2 z$ and $\{y, b\}$ is independent in M_2. As y is not in any D_{M_i}, this concludes the proof. □

Remark 4 In the example of Proposition 1, we have $g(S) = w(a) + w(b)$, and $\{a, b\}$ is independent in all 3 matroids, so this does not contradict Conjecture 1.

Conjecture 1 The stack S generated by Algorithm 2 contains a k approximation for any k.

In the case $k = 2$, this corresponds to Theorem 1. For any k, one can easily find examples were S does not contain more than a k approximation, but we were unable to find an example were it does not contain a k approximation.

A Extending Algorithm 3 when matroid ranks are unknown

In this section, we extend the Algorithm 3 to the case where the the matroid ranks i.e, r_i are unknown. Since r_i is not known, we can not set $y = \frac{\min(r_1, r_2)}{\varepsilon^2}$. The idea is to guess the rank of one of the matroids, say M_1 and adapt the y value that would be assigned to newly added elements as we update our guess. The y values are set in a way that the errors arising from deleting elements specific to a single y value form a geometrically decreasing sequence.

More concretely, we set a y value specific to each element $e \in S$ using the notion of stacks introduced in the Proof of Lemma 6. To recap, on the arrival of element e that is added to our set S initially (may be deleted later), the following two things might happen to the maximum weight independent set with respect to w_1 i.e, T_1. Either it replaces an element e' in T_1, or it is added to T_1. In the former case, we say that e is added to the stack that contains e'. In the latter, we say that we create a new stack and add e to it. In Algorithm 6, we use s to denote the number of stacks at any point during the execution. The y value of an element is decided based on the stack it belongs to. Specifically, the y value of the elements in the first two stacks is set to $\frac{4}{\varepsilon^2}$, the next four stacks to $\frac{16}{\varepsilon^2}$ and in general a value of $\frac{4^i}{\varepsilon^2}$ for the next 2^i stacks. This is done in Algorithm 6 by the function $z : \mathbb{N} \mapsto \mathbb{R}$ where $z(1) = z(2) = \frac{4}{\varepsilon^2}$, $z(3) = z(4) = z(5) = z(6) = \frac{16}{\varepsilon^2}$ and in general a value of $\frac{4^i}{\varepsilon^2}$ for the next 2^i numbers. The values are set in such a way so that the error introduced in each bundle of stacks forms a geometrically decreasing sequence. Using arguments used in the Proof of Lemma 6, the error introduced in the ith bundle of stacks that contains 2^i stacks is at most $\frac{2^i g_{max} \varepsilon^2 (1+\varepsilon)}{4^i \varepsilon} = \frac{g_{max} \varepsilon (1+\varepsilon)}{2^i}$. Summing this over all i gives us that the error is at most $g_{max} \varepsilon (1 + \varepsilon)$ giving us the same error we obtained in the Proof of Lemma 6. By an analysis similar to the one in Proof of Lemma 6, we can prove that the size of S never exceeds $r_1 + r_2 + \max(r_1, r_2) \log_\alpha(\frac{\alpha y_f}{\varepsilon})$ where y_f is the value in the last bundle of stacks formed. By simple calculation, y_f is at most $\frac{\max(r_1, r_2)^2}{\varepsilon^2}$.

Algorithm 6 Extension of Algorithm 3 to unknown r_i

Input: A stream of the elements and 2 matroids (which we call M_1, M_2) on the same ground set E, a real number $\alpha > 1$ and a function $z : \mathbb{N} \mapsto \mathbb{R}$.

Output: A set $X \subseteq E$ that is independent in both matroids.

Whenever we write an assignment of a variable with subscript i, it means we do it for $i = 1, 2$.

$S \leftarrow \emptyset, T_i \leftarrow \emptyset$

$s = 0$ {//initialize no. of stacks to zero.}

for element e in the stream **do**

    calculate $w_i^*(e) = \max \Big(\{0\} \cup \{\theta : e \in \text{span}_{M_i} (\{f \in S \mid w_i(f) \geq \theta\})\} \Big)$.

    if $w(e) > \alpha(w_1^*(e) + w_2^*(e))$ **then**

        $g(e) \leftarrow w(e) - w_1^*(e) - w_2^*(e)$

        $S \leftarrow S \cup \{e\}$

        $w_i(e) \leftarrow g(e) + w_i^*(e)$

        Let H_i be a maximum weight independent set of M_i with respect to w_i.

        if $\{e\} \cup T_1 = H_1$ **then**

            $s = s + 1$

            $y(e) = z(s)$ {//H_1 is formed by adding e to T_1}

        else

            $\{e'\} = T_1 \backslash (H_1 \backslash \{e\})$ {//H_1 is formed by replacing e' with e in T_1}

            $y(e) = y(e')$

        end if

        $T_i = H_i$

        Let $g_{max} = \max\limits_{e \in S} g(e)$

        Remove all elements $e' \in S$, such that $y(e') \cdot g(e') < g_{max}$ and $e' \notin T_1 \cup T_2$ from S.

    end if

end for

return a maximum weight set $T \subseteq S$ that is independent in M_1 and M_2

References

1. Buchbinder, N., Feldman, M., Naor, J., Schwartz, R.: *Submodular maximization with cardinality constraints*, In: Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms, SIAM, pp. 1433–1452 (2014)

2. Bar-Yehuda, R., Bendel, K., Freund, A., Rawitz, D.: Local ratio: A unified framework for approximation algorithms. in memoriam: Shimon even 1935-2004. ACM Computing Surveys (CSUR) **36**, 422–463 (2004)

3. Bar-Yehuda, R., Even, S.: A local-ratio theorem for approximating the weighted vertex cover problem, Analysis and Design of Algorithms for Combinatorial Problems (G. Ausiello and M. Lucertini, eds.), North-Holland Mathematics Studies, vol. 109, North-Holland, pp. 27 – 45 (1985)

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

5. Crouch, M., Stubbs, D.M.: Improved streaming algorithms for weighted matching, via unweighted matching. Leibniz Int. Proc. Inform. LIPIcs **28**, 96–104 (2014)

6. Edmonds, J.: Matroid intersection, Discrete Optimization I. In: Annals of Discrete Mathematics, vol. 4, pp. 39–49. Elsevier, Amsterdam, Netherlands (1979)

7. Feigenbaum, J., Kannan, S., McGregor, A., Suri, S., Zhang, J.: On graph problems in a semi-streaming model. Theoret. Comput. Sci. **348**(2–3), 207–216 (2005)

8. Fleiner, T.: A Matroid Generalization of the Stable Matching Polytope. In: International Conference on Integer Programming and Combinatorial Optimization, pp. 105–114. Springer, Berlin (2001)

9. Ghaffari, M., Wajc, D.: Simplified and Space-Optimal Semi-Streaming (2+epsilon)-Approximate Matching, 2nd Symposium on Simplicity in Algorithms (SOSA 2019) (Dagstuhl, Germany) (Jeremy T. Fineman and Michael Mitzenmacher, eds.), OpenAccess Series in Informatics (OASIcs), vol. 69, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, pp. 13:1–13:8 (2018)

10. Huang, C.-C., Kakimura, N., Mauras, S., Yoshida, Y.: Approximability of monotone submodular function maximization under cardinality and matroid constraints in the streaming model, CoRR arXiv:2002.05477 (2020)
11. Levin, R., Wajc, D.: *Streaming submodular matching meets the primal-dual method*, In: Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA), SIAM, pp. 1914–1933 (2021)
12. McGregor, A.: *Finding graph matchings in data streams*, Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques (Berlin, Heidelberg) (Chandra Chekuri, Klaus Jansen, José D. P. Rolim, and Luca Trevisan, eds.), Springer Berlin Heidelberg, pp. 170–181 (2005)
13. Muthukrishnan, S.: Data streams: algorithms and applications. Found. Trends Theor. Comput. Sci. **1**(2), 117–236 (2005)
14. Paz, A., Schwartzman, G.: A $(2+\varepsilon)$-approximation for maximum weight matching in the semi-streaming model. ACM Transactions on Algorithms (TALG) **15**(2), 1–15 (2018)
15. Schrijver, A.: Combinatorial Optimization: Polyhedra and Efficiency. In: Algorithms and Combinatorics, Springer, Berlin (2003)