

Deterministic $(2/3 - \varepsilon)$ -Approximation of Matroid Intersection Using Nearly-Linear Independence-Oracle Queries

Tatsuya Terao   

Research Institute for Mathematical Sciences, Kyoto University, Japan

Abstract

In the matroid intersection problem, we are given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ defined on the same ground set V of n elements, and the objective is to find a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ of largest possible cardinality, denoted by r . In this paper, we consider a deterministic matroid intersection algorithm with only a nearly linear number of independence oracle queries. Our contribution is to present a deterministic $O(\frac{n}{\varepsilon} + r \log r)$ -independence-query $(2/3 - \varepsilon)$ -approximation algorithm for any $\varepsilon > 0$. Our idea is very simple: we apply a recent $\tilde{O}(n\sqrt{r}/\varepsilon)$ -independence-query $(1 - \varepsilon)$ -approximation algorithm of Blikstad [ICALP 2021], but terminate it before completion. Moreover, we also present a semi-streaming algorithm for $(2/3 - \varepsilon)$ -approximation of matroid intersection in $O(1/\varepsilon)$ passes.

2012 ACM Subject Classification Theory of computation $\rightarrow$ Discrete optimization; Mathematics of computing $\rightarrow$ Matroids and greedoids; Theory of computation $\rightarrow$ Streaming, sublinear and near linear time algorithms

Keywords and phrases Matroid intersection, approximation algorithm, streaming algorithm

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.50

Related Version *Full Version*: <https://arxiv.org/abs/2410.18820>

Funding This work was partially supported by the joint project of Kyoto University and Toyota Motor Corporation, titled “Advanced Mathematical Science for Mobility Society” and by JSPS KAKENHI Grant Number JP24KJ1494.

Acknowledgements The author thanks Yusuke Kobayashi for his generous support and helpful comments on the manuscript.

1 Introduction

The *matroid intersection problem* is one of the most fundamental problems in combinatorial optimization. In this problem, we are given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ defined on the same ground set V of n elements, and the objective is to find a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ of largest possible cardinality, denoted by r . This problem generalizes many important combinatorial optimization problems such as bipartite matching, packing spanning trees, and arborescences in directed graphs. Furthermore, it has also several applications outside of traditional combinatorial optimization such as electrical engineering [51, 55].

To design an algorithm for arbitrary matroids, it is common to consider an oracle model: an algorithm accesses a matroid through an oracle. The most standard and well-studied oracle is an *independence oracle*, which takes as input a subset $S \subseteq V$ and outputs whether S is independent or not. Many studies consider the following research question: how few queries can we solve the matroid intersection problem with?



© Tatsuya Terao;

licensed under Creative Commons License CC-BY 4.0

19th International Symposium on Algorithms and Data Structures (WADS 2025).

Editors: Pat Morin and Eunjin Oh; Article No. 50; pp. 50:1–50:18

Leibniz International Proceedings in Informatics



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

Starting the work of Edmonds [22, 23], many algorithms with polynomial query complexity for the matroid intersection problem have been studied [4, 9, 11, 13–15, 18, 20, 21, 36, 48, 49, 52, 53, 57, 60]. Edmonds [22] developed the first polynomial-query algorithm by reduction to the matroid partitioning problem. This algorithm requires $O(n^4)$ independence oracle queries. More direct algorithms were given by Aigner–Dowling [4] and Lawler [48]. The algorithm of Lawler requires $O(nr^2)$ independence oracle queries. In 1986, Cunningham [21] presented an $O(nr^{3/2})$ -independence-query algorithm by using a blocking flow approach, which is akin to the bipartite matching algorithm of Hopcroft–Karp [35]. Chekuri–Quanrud [20] pointed out that, for any $\varepsilon > 0$, an $O(nr/\varepsilon)$ -independence-query $(1 - \varepsilon)$ -approximation algorithm can be obtained by terminating Cunningham’s algorithm early. This idea comes from the well-known fact that, for the bipartite matching problem, a linear-time $(1 - \varepsilon)$ -approximation algorithm can be obtained by terminating Hopcroft–Karp’s algorithm early. Recently, Nguyễn [52] and Chakrabarty–Lee–Sidford–Singla–Wong [18] independently presented a new binary search technique that can efficiently find edges in the exchange graph and developed a combinatorial $\tilde{O}(nr)$ -independence-query exact algorithm.¹ Chakrabarty et al. also presented a new augmenting sets technique and developed an $\tilde{O}(n^{1.5}/\varepsilon^{1.5})$ -independence-query $(1 - \varepsilon)$ -approximation algorithm. The techniques developed by Nguyễn and Chakrabarty et al. have been used in recent several studies on fast algorithms for other matroid problems [12, 16, 43, 54, 58, 61]. Blikstad–van den Brand–Mukhopadhyay–Nanongkai [15] first broke the $\tilde{O}(n^2)$ -independence-query bound for exact matroid intersection algorithms. Blikstad [11] improved the independence query complexity of a $(1 - \varepsilon)$ -approximation algorithm to $\tilde{O}(n\sqrt{r}/\varepsilon)$. Blikstad’s improvement on the $(1 - \varepsilon)$ -approximation algorithm resulted in a randomized $\tilde{O}(nr^{3/4})$ -independence-query exact algorithm and a deterministic $\tilde{O}(nr^{5/6})$ -independence-query exact algorithm. Recently, Quanrud [53] presented a randomized $\tilde{O}_\varepsilon(n + r^{3/2})$ -independence-query $(1 - \varepsilon)$ -approximation algorithm². Remarkably, Blikstad–Tu [14] very recently presented a randomized $\tilde{O}_\varepsilon(n)$ -independence-query $(1 - \varepsilon)$ -approximation algorithm.³

Both the recent $(1 - \varepsilon)$ -approximation algorithms by Quanrud [53] and Blikstad–Tu [14] are randomized. Therefore, it remains an open question whether a deterministic $\tilde{O}_\varepsilon(n)$ independence-query $(1 - \varepsilon)$ -approximation algorithm can be achieved for almost the entire range of r .⁴ Then, we consider a deterministic matroid intersection algorithm with only a nearly linear number of independence oracle queries. It is well-known that any maximal common independent set is at least half the size of a maximum common independent set (see e.g., [47, Proposition 13.26]). Thus, the natural greedy algorithm – pick an element whenever possible – can be regarded as a deterministic $1/2$ -approximation algorithm using a linear number of independence oracle queries. In fact, even beating the trivial $1/2$ -approximation ratio for deterministic nearly-linear-independence-query algorithms remains an open problem for almost the entire range of r .⁴

In this work, we present a deterministic $(2/3 - \varepsilon)$ -approximation algorithm using a nearly linear number of independence oracle queries.

► **Theorem 1.** *Given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ on the same ground set V , for any $\varepsilon > 0$, there is a deterministic algorithm that finds a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ with $|S| \geq (2/3 - \varepsilon)r$, using $O(\frac{n}{\varepsilon} + r \log r)$ independence oracle queries.*

¹ The $\tilde{O}$ notation omits factors polynomial in $\log n$.

² The $\tilde{O}_\varepsilon$ notation omits factors polynomial in ε and $\log n$.

³ The results in this manuscript were obtained independently of the remarkable result by Blikstad–Tu [14].

⁴ Blikstad–Tu [14] presented a deterministic $(1 - \varepsilon)$ -approximation algorithm using a strictly linear number of independence oracle queries only when $r = \Theta(n)$ (and ε is constant).

Our algorithm uses only a strictly linear number of independence oracle queries when $r = O(n/\log n)$ (and ε is constant). Prior to our work, Guruganesh–Singla [34] presented a randomized $O(n)$ -independence-query algorithm that achieves $(1/2 + \delta)$ -approximation for some small constant $\delta > 0$, which was the best approximation ratio for (possibly randomized) algorithms using only a strictly linear number of independence oracle queries.⁴

We note that the time complexity of our algorithm is dominated by the independence oracle queries. That is, our algorithm in Theorem 1 has time complexity $O((\frac{n}{\varepsilon} + r \log r) \cdot \mathcal{T}_{\text{ind}})$, where $\mathcal{T}_{\text{ind}}$ denotes the maximum running time of a single independence oracle query.

Our idea is very simple: we apply a recent $\tilde{O}(n\sqrt{r}/\varepsilon)$ -independence-query $(1 - \varepsilon)$ -approximation algorithm of Blikstad [11], but terminate it before completion. We observe that a $(2/3 - \varepsilon)$ -approximate solution can be obtained by terminating Blikstad’s algorithm early. As such, our algorithm does not introduce technical novelty, but we believe that it enhances the understanding of algorithms for the matroid intersection problem.

We also implement our algorithm using a *rank oracle*, which takes as input a subset $S \subseteq V$ and outputs the size of the maximum cardinality independent subset of S . Several recent studies have considered fast matroid intersection algorithms in the rank oracle model [18, 49, 60]. We note that the rank oracle is more powerful than the independence oracle, since a single query to the rank oracle can determine whether a given set is independent or not.

Chakrabarty–Lee–Sidford–Singla–Wong [18] presented an $\tilde{O}(n\sqrt{r})$ -rank-query exact algorithm and an $O(\frac{n}{\varepsilon} \log n)$ -rank-query $(1 - \varepsilon)$ -approximation algorithm. Their $(1 - \varepsilon)$ -approximation algorithm requires nearly linear rank oracle queries. Our second result is to obtain a $(2/3 - \varepsilon)$ -approximation algorithm using a strictly linear number of rank oracle queries.

► **Theorem 2.** *Given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ on the same ground set V , for any $\varepsilon > 0$, there is a deterministic algorithm that finds a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ with $|S| \geq (2/3 - \varepsilon)r$, using $O(\frac{n}{\varepsilon})$ rank oracle queries.*

We note that the time complexity of our algorithm is dominated by the rank oracle queries. That is, our algorithm in Theorem 2 has time complexity $O(\frac{n}{\varepsilon} \cdot \mathcal{T}_{\text{rank}})$, where $\mathcal{T}_{\text{rank}}$ denotes the maximum running time of a single rank oracle query.

Surprisingly, our algorithm can also be implemented in the semi-streaming model of computation. The streaming model for graph problems was initiated by Feigenbaum–Kannan–McGregor–Suri–Zhang [27], which is called the semi-streaming model. In this model, an algorithm computes over a graph stream using a limited space of $\tilde{O}(n)$ bits, where n is the number of vertices. The maximum matching problem is one of the most studied problems in the graph streaming setting [1–3, 5–8, 10, 24–27, 29–31, 33, 40–42, 44–46, 50, 59]. The first of these studies was a $(2/3 - \varepsilon)$ -approximation algorithm for the bipartite maximum matching problem by Feigenbaum et al. [27], which uses $\tilde{O}(n)$ bits of space and $O(\log(1/\varepsilon)/\varepsilon)$ passes. In particular, whether a $(1 - \varepsilon)$ approximation can be achieved in $\text{poly}(1/\varepsilon)$ passes in the semi-streaming model for general graphs was a significant open problem until it was solved by Fischer–Mitrović–Uitto [30].

Since the matroid intersection problem is a well-known generalization of the bipartite matching problem, there are also several studies on the matroid intersection problem (and its generalizations) in the streaming setting in the literature [17, 19, 28, 32, 37–39]. For the matroid intersection problem, in the semi-streaming model, we consider algorithms such that the memory usage is $O((r_1 + r_2)\text{polylog}(r_1 + r_2))$, where r_1 and r_2 denote the ranks of matroids $\mathcal{M}_1$ and $\mathcal{M}_2$, respectively. We note that this memory requirement is natural when we formulate the bipartite matching problem as the matroid intersection problem.

In this work, we focus on constant-passes ($\text{poly}(1/\varepsilon)$ -passes) semi-streaming algorithms for the matroid intersection problem, particularly given the significance of constant-passes semi-streaming algorithms for the matching problem.⁵ Recently, Huang–Sellier [37] presented a $(2/3 - \varepsilon)$ -approximation semi-streaming algorithm for the matroid intersection problem in the random-order streaming model.⁶

Our third result is to obtain a $(2/3 - \varepsilon)$ -approximation semi-streaming algorithm with $O(\frac{1}{\varepsilon})$ passes in the adversary-order streaming model.

► **Theorem 3.** *Given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ on the same ground set V , for any $\varepsilon > 0$, there is a deterministic semi-streaming algorithm that finds a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ with $|S| \geq (2/3 - \varepsilon)r$, using $O(\frac{r_1 + r_2}{\varepsilon})$ memory and $O(\frac{1}{\varepsilon})$ passes.*

Our algorithm in Theorem 3 is a generalization of the bipartite matching semi-streaming algorithm of Feigenbaum et al. [27].

2 Preliminaries

2.1 Notation

Here, we provide the basic notations and conventions used throughout the paper.

Set Notation

For a set A and an element x , we will often write $A + x := A \cup \{x\}$ and $A - x := A \setminus \{x\}$. For two sets A and B , we will also write $A + B := A \cup B$ and $A - B := A \setminus B$ when no confusion can arise.

Matroid

A pair $\mathcal{M} = (V, \mathcal{I})$ of a finite set V and a non-empty set family $\mathcal{I} \subseteq 2^V$ is called a *matroid* if the following properties are satisfied.

(Downward closure property) If $S \in \mathcal{I}$ and $S' \subseteq S$, then $S' \in \mathcal{I}$.

(Augmentation property) If $S, S' \in \mathcal{I}$ and $|S'| < |S|$, then there exists $v \in S \setminus S'$ such that $S' + v \in \mathcal{I}$.

A set $S \subseteq V$ is called *independent* if $S \in \mathcal{I}$ and *dependent* otherwise.

Rank

For a matroid $\mathcal{M} = (V, \mathcal{I})$, we define the *rank* of $\mathcal{M}$ as $\text{rank}(\mathcal{M}) = \max\{|S| \mid S \in \mathcal{I}\}$. In addition, for any $S \subseteq V$, we define the *rank* of S as $\text{rank}_{\mathcal{M}}(S) = \max\{|T| \mid T \subseteq S, T \in \mathcal{I}\}$.

⁵ Based on the results from Assadi [5] and Quanrud [53], we believe that it is able to develop a semi-streaming $(1 - \varepsilon)$ -approximation algorithm with similar space usage. However, such an algorithm would require $O(\log(n) \cdot \text{poly}(1/\varepsilon))$ passes. In the context of research on streaming matching algorithms, the distinctions between $O(\text{poly}(1/\varepsilon))$ passes and $O(\log(n) \cdot \text{poly}(1/\varepsilon))$ are well recognized and crucial; see [5, Table 1 in the arXiv version]. We also believe that the result by Blikstad–Tu [14] is unlikely to immediately lead to constant-passes semi-streaming algorithm for the matroid intersection problem. This is because, while Blikstad–Tu used some idea from the constant-passes semi-streaming algorithm of Assadi–Liu–Tarjan [8] for the bipartite matching problem, Blikstad–Tu relies on the technique from Quanrud [53]. Quanrud’s algorithm used some idea from the $O(\log(n) \cdot \text{poly}(1/\varepsilon))$ -passes semi-streaming algorithm by Assadi [5].

⁶ We note that their algorithm requires too many queries to compute the density-based decomposition. Thus, their algorithm does not imply a nearly-linear-independence-query $(2/3 - \varepsilon)$ -approximation algorithm.

Matroid Intersection

Given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ defined on the same ground set V , a *common independent set* S is a set in $\mathcal{I}_1 \cap \mathcal{I}_2$. In the *matroid intersection problem*, the aim is to find a largest common independent set, whose cardinality we denote by r .

2.2 Exchange Graph and Augmenting Path

We provide the standard definitions of *exchange graph* and *augmenting paths* and known lemmas used in recent fast matroid intersection algorithms. Many matroid intersection algorithms use an approach of iteratively finding augmenting paths in the exchange graph.

► **Definition 4** (Exchange Graph). Consider a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ of two matroids $\mathcal{M}_1$ and $\mathcal{M}_2$. The exchange graph is defined as a directed graph $G(S) = (V \cup \{s, t\}, E)$, with $s, t \notin V$ and $E = E' \cup E'' \cup E_s \cup E_t$, where

$$\begin{aligned} E' &= \{(u, v) \mid u \in S, v \in V \setminus S, S - u + v \in \mathcal{I}_1\}, \\ E'' &= \{(v, u) \mid u \in S, v \in V \setminus S, S - u + v \in \mathcal{I}_2\}, \\ E_s &= \{(s, v) \mid v \in V \setminus S, S + v \in \mathcal{I}_1\}, \text{ and} \\ E_t &= \{(v, t) \mid v \in V \setminus S, S + v \in \mathcal{I}_2\}. \end{aligned}$$

► **Lemma 5** (Shortest Augmenting Path; see [56, Theorem 41.2]). Let $s, v_1, v_2, \dots, v_{\ell-1}, t$ be a shortest (s, t) -path in the exchange graph $G(S)$ relative to a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$. Then, $S' = S + v_1 - v_2 + \dots - v_{\ell-2} + v_{\ell-1} \in \mathcal{I}_1 \cap \mathcal{I}_2$.

Cunningham's [21] matroid intersection algorithm and recent fast matroid intersection algorithms [11, 15, 18, 20, 52] rely on the following lemma.

► **Lemma 6** (from [21, Corollary 2.2]). Let S be a common independent set which is not maximum. Then, there exists an (s, t) -path of length at most $\frac{2|S|}{r-|S|} + 2$ in the exchange graph $G(S)$.

Chakrabarty–Lee–Sidford–Singla–Wong [18] presented a new binary search technique that can efficiently find edges in the exchange graph. (This technique was developed independently by Nguyễn [52].)

► **Lemma 7** (Binary Search Technique from [18, 52]). Given a matroid $\mathcal{M} = (V, \mathcal{I})$, an independent set $S \in \mathcal{I}$, an element $v \in V \setminus S$, and $T \subseteq S$, using $O(\log |T|)$ independence oracle queries, we can find an element $u \in T$ such that $S + v - u \in \mathcal{I}$ or otherwise determine that no such element exists. Furthermore, if there exists no such an element, then this procedure uses only one independence oracle query.⁷

Let $\text{FindExchange}(\mathcal{M}, S, v, T)$ be the procedure that implements Lemma 7.

Let D_i be the set of elements $v \in V$ such that the distance from s to v is exactly i in the exchange graph $G(S)$. Chakrabarty et al. [18] showed that the distance layers D_i can be computed efficiently by using the binary search technique. In their algorithm, they compute odd and even layers separately because FindExchange cannot find both incoming and outgoing edges due to restrictions on the allowed queries; see [18, Algorithm 6 and Lemma 19]. Their algorithm requires $O(nr + r^2 \log r)$ independence oracle queries to compute all distance layers $D_1, D_2, \dots, D_{2r+1}$; see [18, Remark just after Proof of Theorem 18]. In

⁷ This is because there exists such an element if and only if $S + v - T \in \mathcal{I}$.

Algorithm 1 GetDistance.

Input: a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$
Output: distance layers $D_1 \subseteq V \setminus S$, $D_2 \subseteq S$, and $D_3 \subseteq V \setminus S$

```

1  $D_1 \leftarrow \emptyset$ ,  $D_2 \leftarrow \emptyset$ ,  $D_3 \leftarrow \emptyset$ 
2 for  $v \in V \setminus S$  with  $S + v \in \mathcal{I}_1$  do
3    $D_1 \leftarrow D_1 + v$ 
4  $Q \leftarrow D_1$ ,  $T \leftarrow S$ 
5 while  $Q$  contains some  $v$  do
6   while  $u = \text{FindExchange}(\mathcal{M}_2, S, v, T)$  satisfies  $u \neq \emptyset$  do
7      $D_2 \leftarrow D_2 + u$ ,  $T \leftarrow T - u$ 
8    $Q \leftarrow Q - v$ 
9 for  $v \in V \setminus (S \cup D_1)$  with  $S + v - D_2 \in \mathcal{I}_1$  do
10   $D_3 \leftarrow D_3 + v$ 

```

our matroid intersection algorithm, we only need to compute D_1 , D_2 , and D_3 . Their algorithm can compute D_1 , D_2 , and D_3 using $O(n + r \log r)$ independence oracle queries. See **GetDistance** (Algorithm 1) for the pseudocode of the algorithm.

► **Lemma 8** (follows from [18, Lemma 19]). *Given a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ of two matroids $\mathcal{M}_1$ and $\mathcal{M}_2$, using $O(n + r \log r)$ independence oracle queries, we can find the distance layers $D_1 \subseteq V \setminus S$, $D_2 \subseteq S$, and $D_3 \subseteq V \setminus S$.*

For completeness, we give a proof of Lemma 8.

Proof. The procedure **GetDistance** (Algorithm 1) simply performs a breadth first search in the exchange graph $G(S)$. Thus, the procedure **GetDistance** correctly computes the distance layers D_1 , D_2 , and D_3 . Here, each element $u \in D_2$ is found by **FindExchange** only once. Thus, the number of **FindExchange** calls that do not output $\emptyset$ is $|D_2| \leq r$, and the number of **FindExchange** calls that output $\emptyset$ is $|D_1| \leq n$. Hence, by Lemma 7, the number of independence oracle queries used in Line 6 is $O(n + r \log r)$. In addition, the number of independence oracle queries used in Lines 2 and 9 is $O(n)$, which completes the proof. ◀

3 Augmenting Sets Technique

In this section, we recall the definition and properties of *augmenting sets*, which were introduced as a generalization of *augmenting paths* by Chakrabarty–Lee–Sidford–Singla–Wong [18]. The augmenting sets technique plays a crucial role in the $(1 - \varepsilon)$ -approximation algorithms of Chakrabarty et al. [18] and Blikstad [11].

In our matroid intersection algorithm, we only need to find an augmenting set in the exchange graph whose shortest (s, t) -path length is 4. Thus, we introduce the definition and properties of augmenting sets when restricted to the case where the length of a shortest augmenting path is 4.

► **Definition 9** (Augmenting Sets from [18, Definition 24]). *Let $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ be such that shortest augmenting paths in $G(S)$ have length 4. We say that a collection of sets $\Pi := (B_1, A_1, B_2)$ form an augmenting set (of width w) in $G(S)$ if the following conditions are satisfied:*

- (a) $B_1 \subseteq D_1$, $A_1 \subseteq D_2$, and $B_2 \subseteq D_3$.
- (b) $|B_1| = |A_1| = |B_2| = w$
- (c) $S + B_1 \in \mathcal{I}_1$

- (d) $S + B_1 - A_1 \in \mathcal{I}_2$
- (e) $S - A_1 + B_2 \in \mathcal{I}_1$
- (f) $S + B_2 \in \mathcal{I}_2$

► **Theorem 10** (from [18, Theorem 25]). *Let $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ be such that shortest augmenting paths in $G(S)$ have length 4. Let $\Pi := (B_1, A_1, B_2)$ be an augmenting set in the exchange graph $G(S)$. Then, the set $S' := S \oplus \Pi := S + B_1 - A_1 + B_2$ is a common independent set.*

Here, we recall the definition and property of *maximal augmenting sets*, which correspond to a maximal collection of shortest augmenting paths such that augmentation along them must increase the (s, t) -distance. In the $(1 - \varepsilon)$ -approximation algorithms of Chakrabarty et al. and Blikstad, they repeatedly find a maximal augmenting set and augment along it.

► **Definition 11** (Maximal Augmenting Sets from [18, Definitions 31 and 35]). *Let $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ be such that shortest augmenting paths in $G(S)$ have length 4. Let $\Pi := (B_1, A_1, B_2)$ and $\tilde{\Pi} := (\tilde{B}_1, \tilde{A}_1, \tilde{B}_2)$ be two augmenting sets. We say $\tilde{\Pi}$ contains Π if $B_1 \subseteq \tilde{B}_1$, $A_1 \subseteq \tilde{A}_1$ and $B_2 \subseteq \tilde{B}_2$. We use the notation $\Pi \subseteq \tilde{\Pi}$ to denote this. An augmenting set Π is called maximal if there exists no other augmenting set $\tilde{\Pi}$ containing Π .*

► **Theorem 12** (from [18, Theorem 36]). *Let $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ be such that shortest augmenting paths in $G(S)$ have length 4. Let Π be a maximal augmenting set in the exchange graph $G(S)$. Then, there is no augmenting path of length at most 4 in $G(S \oplus \Pi)$.*

Here, we recall the definition of *partial augmenting sets*, which are the relaxed form of augmenting sets. In the algorithms of Chakrabarty et al. and Blikstad for finding a maximal augmenting set, they keep track of the partial augmenting set, which is iteratively made close to a maximal augmenting set through some refine procedures.

► **Definition 13** (Partial Augmenting Sets from [18, Definition 37]). *Let $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ be such that shortest augmenting paths in $G(S)$ have length 4. We say that $\Phi := (B_1, A_1, B_2)$ forms a partial augmenting set in $G(S)$ if it satisfies the conditions (a), (c), (e), (f) of an augmenting set, and the following two relaxed conditions:*

- (b) $|B_1| \geq |A_1| \geq |B_2|$
- (d) $\text{rank}_2(S + B_1 - A_1) = \text{rank}_2(S)$

Chakrabarty et al. presented an efficient algorithm to convert any partial augmenting set Φ into an augmenting set Π .

► **Lemma 14** (from [18, Lemma 38]). *Given a partial augmenting set $\Phi = (B_1, A_1, B_2)$, using $O(n)$ independence oracle queries, we can find an augmenting set $\Pi = (B'_1, A'_1, B'_2)$ such that $B'_1 \subseteq B_1$, $A'_1 \subseteq A_1$ and $B'_2 = B_2$.*

Chakrabarty et al. [18] presented an algorithm to find a maximal augmenting set in the exchange graph $G(S)$, where shortest augmenting paths have length $2(\ell + 1)$, using $O(n^{1.5} \sqrt{\ell \log r})$ independence oracle queries; see [18, Proof of Theorem 48 and Proof of Theorem 21]. Blikstad [11] improved the query complexity and presented an algorithm to find a maximal augmenting set using $O(n\sqrt{r \log r})$ independence oracle queries; see [11, Lemma 35 and Proof of Theorem 1].

In Blikstad's algorithm for finding a maximal augmenting set, we keep track of a partial augmenting set and iteratively update it to closer to a maximal augmenting set. To obtain a fast algorithm, Blikstad combines two algorithms **Refine** [11, Algorithm 4] and **RefinePath** [11, Algorithm 5]; see [11, Algorithm 6 and Lemma 35]. Let $p \in [1, r]$ be the parameter that controls the trade-off of the two algorithms. He first applies **Refine** to find a

partial augmenting set that is close enough to a maximal augmenting set, which uses $O(nr/p)$ independence oracle queries. Then, he applies **RefinePath** until the partial augmenting set becomes a maximal augmenting set, which uses $O(np \log r)$ independence oracle queries.

In our algorithm, we apply only the first part **Refine** of Blikstad's algorithm, as formally stated as follows.

► **Lemma 15** (follows from [11, Lemma 35]). *For any $p \in [1, r]$, given a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ and the distance layers D_1, D_2 , and D_3 , using $O(\frac{nr}{p})$ independence oracle queries, we can find a partial augmenting set $\Phi = (B_1, A_1, B_2)$ such that the following properties hold.*

- (i) *There is an augmenting set $\Pi = (B'_1, A'_1, B'_2)$ such that $B'_1 \subseteq B_1$, $A'_1 \subseteq A_1$ and $B'_2 = B_2$.*
- (ii) *We have $|B_1| - |B_2| \leq p$.*
- (iii) *There is a maximal augmenting set $\tilde{\Pi}$ of width at most $|B_1|$ in $G(S)$.*

To obtain Lemma 15, we apply **Refine** [11, Algorithm 4] (see also Algorithm 5) until $|B_1| - |B_2| \leq p$, but at least once. The procedure **Refine** makes the partial augmenting set $\Phi = (B_1, A_1, B_2)$ close to a maximal augmenting set. To become $|B_1| - |B_2| \leq p$, we need to apply **Refine** $O(|S|/p + 1)$ times; see [11, Proof of Lemma 35]. Since each call of **Refine** uses $O(n)$ independence oracle queries (see [11, Lemma 29]), the total number of independence oracle queries is $O(nr/p)$.

► **Remark 16.** Lemma 15 is not explicitly stated in Blikstad's [11] paper. By the property of a partial augmenting set and the argument in [11, Proof of Lemma 35], it is clear that the conditions (i) and (ii) are satisfied. In particular, here we show that the condition (iii) is also satisfied. In the algorithm in [11, Algorithm 6 and Lemma 35] for finding a maximal augmenting set, after applying **Refine** until $|B_1| - |B_2| \leq p$, we apply **RefinePath** [11, Algorithm 5] until the partial augmenting set becomes a maximal augmenting set. While **RefinePath** modifies B_1 , it does not increase $|B_1|$; see [11, the second paragraph of the proof of Lemma 35]. Thus, the partial augmenting set after applying **RefinePath** is a maximal augmenting set of width at most $|B_1|$. Here, let this maximal augmenting set be $\tilde{\Pi}$.

4 $(2/3 - \varepsilon)$ -Approximation Algorithm Using Nearly-Linear Independence-Oracle Queries

In this section, by providing a nearly-linear-independence-query $(2/3 - \varepsilon)$ -approximation algorithm for the matroid intersection problem, we prove Theorem 1, which we restate here.

► **Theorem 1.** *Given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ on the same ground set V , for any $\varepsilon > 0$, there is a deterministic algorithm that finds a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ with $|S| \geq (2/3 - \varepsilon)r$, using $O(\frac{n}{\varepsilon} + r \log r)$ independence oracle queries.*

We use the following lemma to prove Theorem 1.

► **Lemma 17.** *Let $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ be a common independent set such that shortest augmenting paths in $G(S)$ have length at least 6. Then, S is a $2/3$ -approximate solution of the matroid intersection problem.*

Proof. We argue by contraposition. Let $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ be a common independent set with $|S| < \frac{2}{3}r$. Then, we have $\frac{2|S|}{r-|S|} + 2 < 6$. Thus, by Lemma 6, there exists an augmenting path of length at most $\frac{2|S|}{r-|S|} + 2 < 6$ in $G(S)$. This completes the proof. ◀

Now, we give a proof of Theorem 1. See Algorithm 2 for the pseudocode of our algorithm.

■ **Algorithm 2** $(2/3 - \varepsilon)$ -approximation algorithm for the matroid intersection problem.

```

1 Compute a maximal common independent set  $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ .      // Write  $\bar{r} = |S|$ .
2 Compute the distance layers  $D_1, D_2$ , and  $D_3$  in the exchange graph  $G(S)$ .
3 if the  $(s, t)$ -distance in  $G(S)$  is more than 4 then
4   | return  $S$ 
5 Apply Refine (Algorithm 5) until  $|B_1| - |B_2| \leq \varepsilon \bar{r}$ .
6 Apply Lemma 14 for the obtained partial augmenting set  $\Phi = (B_1, A_1, B_2)$  and then
  obtain an augmenting set  $\Pi = (B'_1, A'_1, B'_2)$  such that  $B'_1 \subseteq B_1$ ,  $A'_1 \subseteq A_1$  and
   $B'_2 = B_2$ .
7 return  $S \oplus \Pi$ 

```

Proof of Theorem 1. We give an algorithm to find a $(2/3 - \varepsilon)$ -approximate solution for the matroid intersection problem. In our algorithm, we first compute a maximal common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$, which uses $O(n)$ independence oracle queries. Here, let $\bar{r}$ be the size of this maximal common independent set S . We note that $r \leq 2\bar{r}$ since any maximal common independent set is at least half the size of a maximum common independent set (see e.g., [47, Proposition 13.26]).

Then, we compute the distance layers D_1, D_2 , and D_3 in the exchange graph $G(S)$, which, by Lemma 8, uses $O(n + r \log r)$ independence oracle queries. Here, there is no augmenting path of length 2 in $G(S)$, since S is a maximal common independent set. Thus, shortest augmenting paths in $G(S)$ have length at least 4.

Next, by checking whether $S + v \in \mathcal{I}_2$ for each $v \in D_3$, we check whether the (s, t) -distance is 4 or not, which uses $O(n)$ independence oracle queries. If the (s, t) -distance is more than 4, then S is already a $2/3$ -approximate solution by Lemma 17, so we output S . Otherwise, we apply Lemma 15 in which we set the parameter $p = \varepsilon \bar{r}$, which uses $O(n/\varepsilon)$ independence oracle queries.⁸ Then, we obtain a partial augmenting set $\Phi = (B_1, A_1, B_2)$. By applying Lemma 14 for the obtained partial augmenting set Φ , using $O(n)$ independence oracle queries, we can find an augmenting set $\Pi = (B'_1, A'_1, B'_2)$ such that $B'_1 \subseteq B_1$, $A'_1 \subseteq A_1$ and $B'_2 = B_2$. Then, we output the set $S \oplus \Pi$.

By Theorem 10, the set $S \oplus \Pi$ is a common independent set. Now, we show that the obtained solution $S \oplus \Pi$ is a $(2/3 - \varepsilon)$ -approximate solution.

By the condition (iii) in Lemma 15, there is a maximal augmenting set $\tilde{\Pi} = (\tilde{B}_1, \tilde{A}_1, \tilde{B}_2)$ in $G(S)$ such that $|\tilde{B}_1| \leq |B_1|$. By Theorems 10 and 12 and Lemma 17, $S \oplus \tilde{\Pi}$ is a $2/3$ -approximate solution, so we have $|S \oplus \tilde{\Pi}| \geq \frac{2}{3}r$. Moreover, by the condition (ii) in Lemma 15, we have $|\tilde{B}_1| \leq |B_1| \leq |B_2| + p = |B'_2| + p = |B'_1| + p$. Thus, we have $|S \oplus \tilde{\Pi}| - |S \oplus \Pi| = |\tilde{B}_1| - |B'_1| \leq p = \varepsilon \bar{r} \leq \varepsilon r$. Therefore, we have $|S \oplus \Pi| \geq (2/3 - \varepsilon)r$, which completes the proof. ◀

5 $(2/3 - \varepsilon)$ -Approximation Algorithm Using Linear Rank-Oracle Queries

In this section, by providing a linear-rank-query $(2/3 - \varepsilon)$ -approximation algorithm for the matroid intersection problem, we prove Theorem 2. We show that Algorithm 2 can be implemented as a $(2/3 - \varepsilon)$ -approximation algorithm with linear rank oracle queries. We restate Theorem 2 here.

⁸ We apply **Refine** (Algorithm 5) only $O(1/\varepsilon)$ times.

► **Theorem 2.** *Given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ on the same ground set V , for any $\varepsilon > 0$, there is a deterministic algorithm that finds a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ with $|S| \geq (2/3 - \varepsilon)r$, using $O(\frac{n}{\varepsilon})$ rank oracle queries.*

Chakrabarty–Lee–Sidford–Singla–Wong [18] showed that a $(1 - \varepsilon)$ -approximate solution can be obtained with $O(\frac{n}{\varepsilon} \log n)$ rank oracle queries by using the binary search technique; see [18, Theorem 17]. In their $(1 - \varepsilon)$ -approximation algorithm, they find some edges using the binary search technique. For each edge, this requires $O(\log n)$ rank oracle queries. On the other hand, we show that a $(2/3 - \varepsilon)$ -approximate solution can be obtained with only $O(n/\varepsilon)$ rank oracle queries. In our $(2/3 - \varepsilon)$ -approximation algorithm, we find an almost maximal augmenting set without needing to identify any specific edges. Hence, the rank oracle query complexity of our algorithm is linear without any logarithmic factor.

We use the following lemma to prove Theorem 2.

► **Lemma 18.** *Given a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$, using $O(n)$ rank oracle queries, we can find the distance layers $D_1 \subseteq V \setminus S$, $D_2 \subseteq S$, and $D_3 \subseteq V \setminus S$.*

Proof. Our algorithm for computing D_1, D_2 , and D_3 is almost the same as Algorithm 1. Since a single rank oracle query can determine whether a given subset is independent or not, Algorithm 1 except for computing D_2 can be implemented with $O(n)$ rank oracle queries. Now, we show how D_2 can be computed with $O(n)$ rank oracle queries. For each $u \in S$, we check whether $\text{rank}_{\mathcal{M}_2}(S + D_1 - u) \geq \text{rank}_{\mathcal{M}_2}(S)$ holds or not. If $\text{rank}_{\mathcal{M}_2}(S + D_1 - u) \geq \text{rank}_{\mathcal{M}_2}(S)$ holds, then there exists an element $v \in D_1$ such that $S + v - u \in \mathcal{I}_2$, and consequently $u \in D_2$. Otherwise, there does not exist an element $v \in D_1$ such that $S + v - u \in \mathcal{I}_2$, and consequently $u \notin D_2$. This completes the proof. ◀

Proof of Theorem 2. Now, we show how Algorithm 2 can be implemented with $O(n/\varepsilon)$ rank oracle queries. As mentioned in the proof of Theorem 1, all parts of Algorithm 2, except for Line 2, can be implemented with $O(n/\varepsilon)$ independence oracle queries. Since a single rank oracle query can determine whether a given subset is independent or not, we can implement all parts of Algorithm 2, except for Line 2, with $O(n/\varepsilon)$ rank oracle queries. In addition, by Lemma 18, we can implement Line 2 with $O(n)$ rank oracle queries, which completes the proof. ◀

6 $(2/3 - \varepsilon)$ -Approximation Algorithm in the Semi-Streaming Model

In this section, by providing a $(2/3 - \varepsilon)$ -approximation algorithm for the matroid intersection problem in the semi-streaming model, we prove Theorem 3. We show that Algorithm 2 can also be implemented in the semi-streaming model as a $(2/3 - \varepsilon)$ -approximation algorithm using a constant number of passes. We restate Theorem 3 here.

► **Theorem 3.** *Given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ on the same ground set V , for any $\varepsilon > 0$, there is a deterministic semi-streaming algorithm that finds a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ with $|S| \geq (2/3 - \varepsilon)r$, using $O(\frac{r_1 + r_2}{\varepsilon})$ memory and $O(\frac{1}{\varepsilon})$ passes.*

Before we present how Algorithm 2 can be implemented in the semi-streaming model, we describe the outline of Blikstad’s [11] algorithm of Lemma 15. This is because some of his results will be used in our argument later. In this paper, we skip the proof of the correctness of the algorithm; see [11, Sections 3.1 and 3.3] for the proof. Recall that, in Lemma 15, we apply **Refine** only $O(1/\varepsilon)$ times. See Algorithm 5 for the pseudocode of **Refine**.

■ **Algorithm 3** **RefineAB**(k) (from [11, Algorithm 1]); called **Refine1** in [18, Algorithm 9].

-
- 1 Find maximal $B \subseteq F_{2k+1}$ s.t. $S - A_k + B_{k+1} + B \in \mathcal{I}_1$
 - 2 $B_{k+1} \leftarrow B_{k+1} + B$, $F_{2k+1} \leftarrow F_{2k+1} - B$
 - 3 Find maximal $A \subseteq A_k$ s.t. $S - A_k + B_{k+1} + A \in \mathcal{I}_1$
 - 4 $A_k \leftarrow A_k - A$, $R_{2k} \leftarrow R_{2k} + A$
-

■ **Algorithm 4** **RefineBA**(k) (from [11, Algorithm 2]); called **Refine2** in [18, Algorithm 10].

-
- 1 Find maximal $B \subseteq B_k$ s.t. $S - (D_{2k} - R_{2k}) + B \in \mathcal{I}_2$
 - 2 $R_{2k-1} \leftarrow R_{2k-1} + B_k \setminus B$, $B_k \leftarrow B$
 - 3 Find maximal $A \subseteq F_{2k}$ s.t. $S - (D_{2k} - R_{2k}) + B_k + A \in \mathcal{I}_2$
 - 4 $A_k \leftarrow A_k + F_{2k} \setminus A$, $F_{2k} \leftarrow A$
-

In Blikstad’s algorithm, we maintain three types of elements in each layer (see [11, Section 3.1]):

- *fresh*. Denoted by $F_i \subseteq D_i$. These elements are candidates that could be added to the partial augmenting set.
- *selected*. Denoted by B_1, A_1, B_2 . These elements form the current partial augmenting set $\Pi = (B_1, A_1, B_2)$.
- *removed*. Denoted by $R_i \subseteq D_i$. These elements are deemed useless, and then we can disregard them.

For convenience, we also define imaginary layers D_0 and D_4 with $A_0 = R_0 = F_0 = D_0 = A_2 = R_4 = F_4 = D_4 = \emptyset$.

In **Refine** (Algorithm 5), we iteratively apply **RefineAB** (Algorithm 3) and **RefineBA** (Algorithm 4), and then update the types of elements. Initially, we begin with all elements being *fresh*. Elements can change their types from *fresh* $\rightarrow$ *selected* $\rightarrow$ *removed*, but their types cannot be changed in the other direction. Note that an element of type *fresh* can change to type *removed* without first going through type *selected*.

Now, by providing how Algorithm 2 can be implemented in the semi-streaming model, we give a proof of Theorem 3.

Proof of Theorem 3. We show how Algorithm 2 can be implemented using $O(\frac{r_1+r_2}{\epsilon})$ memory and $O(\frac{1}{\epsilon})$ passes in the semi-streaming model.

In our algorithm, We first compute a maximal common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ using one pass of the stream. Note that we can easily compute a maximal set in a single pass. We store the set S explicitly using $O(r)$ memory.

Then, we compute D_2 in the exchange graph $G(S)$ using one pass of the stream. For each $v \in V \setminus S$, if $S + v \in \mathcal{I}_1$, then we find all elements $u \in S$ such that $S + v - u \in \mathcal{I}_2$ and add them to D_2 .

We store D_2 explicitly using $O(r)$ memory. In our algorithm, we store the types of elements (i.e., F_2, A_1, R_2) in D_2 explicitly using $O(r)$ memory. Whenever an element $v \in V \setminus S$ arrives in the stream, we can determine whether $v \in D_1$, $v \in D_3$, or otherwise, by checking whether $S + v \in \mathcal{I}_1$ and whether there exists an element $u \in D_2$ such that $S + v - u \in \mathcal{I}_1$. Thus, we can maintain the distance layers D_1 and D_3 implicitly.

Next, we apply **Refine** (Algorithm 5) $O(1/\epsilon)$ times. In our implementation of **Refine** in the semi-streaming model, we replace Lines 3–8 in **Refine** with **UpdateABA**(k) (Algorithm 6). By Claim 19, our implementation of **Refine** can also correctly find a desired partial augmenting set.

■ **Algorithm 5** **Refine** (from [11, Algorithms 3 and 4]).

```

1 for  $k = 1, 0$  do
2   RefineBA( $k + 1$ )
3   for  $x \in F_{2k+1}$  do
4     if  $S - A_k + B_{k+1} + x \in \mathcal{I}_1$  then
5       if  $S - A_{k+1} - F_{2k+2} + B_{k+1} + x \in \mathcal{I}_2$  then
6          $B_{k+1} \leftarrow B_{k+1} + x, F_{2k+1} \leftarrow F_{2k+1} - x$ 
7       else
8          $R_{2k+1} \leftarrow R_{2k+1} + x, F_{2k+1} \leftarrow F_{2k+1} - x$ 
9   RefineBA( $k + 1$ )
10  RefineAB( $k$ )

```

■ **Algorithm 6** **UpdateABA**(k) (implementation of Lines 3–8 in **Refine** (Algorithm 5) in the semi-streaming model).

```

1 for  $x \in F_{2k+1}$  do // first pass
2   if  $S - A_k + B_{k+1} + x \in \mathcal{I}_1$  and  $S - A_{k+1} - F_{2k+2} + B_{k+1} + x \in \mathcal{I}_2$  then
3      $B_{k+1} \leftarrow B_{k+1} + x, F_{2k+1} \leftarrow F_{2k+1} - x$ 
4 Store the current  $A_k$  and  $B_{k+1}$  as  $A_k^{(i)}$  and  $B_{k+1}^{(i)}$ , respectively.
   // Assume that this procedure is the  $i$ -th call of UpdateABA( $k$ ) in the
   // entire matroid intersection algorithm.
5 for  $x \in F_{2k+1}$  do // second pass
6   if  $S - A_k + B_{k+1} + x \in \mathcal{I}_1$  then
7      $R_{2k+1} \leftarrow R_{2k+1} + x, F_{2k+1} \leftarrow F_{2k+1} - x$ 

```

▷ **Claim 19.** Even if we replace Lines 3–8 in **Refine** with **UpdateABA**(k) (Algorithm 6), the procedure **Refine** can correctly find a partial augmenting set that satisfies the conditions (i)–(iii) in Lemma 15.

Proof. Consider the order of the elements in F_{2k+1} satisfying the following condition: an element added to B_{k+1} in the first for loop of **UpdateABA**(k) appears before any element that is not added to B_{k+1} in that loop. Suppose that, in the execution of Lines 3–8 in **Refine**, the elements in F_{2k+1} arrive in this order. Then, the changes of types of elements in D_{2k+1} by **UpdateABA**(k) call are exactly same as by the execution of Lines 3–8 in **Refine** (Algorithm 5). Regardless of the order of the elements in the for loop in Line 3, the implementation of **Refine** in Algorithm 5 can find a desired partial augmenting set. Therefore, **Refine** with Lines 3–8 replaced by **UpdateABA**(k) correctly finds a desired partial augmenting set, which completes the proof. ◁

Now, we show that each call of the modified **Refine** can be implemented with $O(1)$ passes. Since we can compute a maximal set in a single pass, both **RefineAB** (Algorithm 3) and **RefineBA** (Algorithm 4) can be implemented with 2 passes. In addition, **UpdateABA** (Algorithm 6) can also be implemented with 2 passes.

Next, we show that we can maintain the types of elements (i.e., $F_1, B_1, R_1, F_3, B_2, R_3$) in D_1 and D_3 implicitly. To do this, we explicitly maintain the following:

- We store the current partial augmenting set (B_1, A_1, B_2) . Since the number of selected elements is always at most $r_1 + r_2$, we use $O(r_1 + r_2)$ memory to store it.

- We store all removed elements whose type has changed from *selected* to *removed*. Only **RefineAB** and **RefineBA** change the types of elements from *selected* to *removed*. In each call of **RefineAB** and **RefineBA**, this change occurs only for at most $r_1 + r_2$ elements. Since the number of **RefineAB** and **RefineBA** calls is $O(1/\varepsilon)$, we use $O((r_1 + r_2)/\varepsilon)$ memory to store them in the entire algorithm.
- We store $A_k^{(i)}$ and $B_{k+1}^{(i)}$ just after the first for loop in **UpdateABA** (Algorithm 6). Since the number of **UpdateABA** calls is $O(1/\varepsilon)$, we use $O((r_1 + r_2)/\varepsilon)$ memory to store them in the entire algorithm.

Here, we note that, for an element whose current type is *removed*, there are only the following two cases.

- The type has changed from *selected* to *removed* by **RefineAB** or **RefineBA**.
- The type has changed from *fresh* to *removed* by **UpdateABA**.

Whenever an element $v \in D_1 \cup D_3$ arrives, we can identify the current type of v in the following way:

- If the current type of v is *selected*, then we can easily identify it.
- If the current type of v is *removed*, then we identify it in the following way:
 - In the case where the type of v has changed from *selected* to *removed* by **RefineAB** or **RefineBA**, we can conclude that the current type of v is *removed*.
 - In the case where the type of v has changed from *fresh* to *removed* by Line 7 in **UpdateABA** (Algorithm 6), we identify it by simulating all previous executions of the second for loop in **UpdateABA**. More precisely, let C be the number of **UpdateABA** called so far. If $v \in D_{2k+1}$ and there is an index $i \leq C$ such that there are $A_k^{(i)}$ and $B_{k+1}^{(i)}$ such that $S - A_k^{(i)} + B_{k+1}^{(i)} + v \in \mathcal{I}_1$, then we conclude that the current type of v is *removed*.
- Otherwise, the current type of v is *fresh*.

Therefore, each call of the modified **Refine** can be implemented with $O(1)$ passes.

After applying **Refine** $O(1/\varepsilon)$ times, we obtain a partial augmenting set $\Phi = (B_1, A_1, B_2)$ such that $|B_1| - |B_2| \leq \varepsilon \bar{r}$. Then, by applying Lemma 14 for the obtained partial augmenting set $\Phi = (B_1, A_1, B_2)$, we obtain an augmenting set $\Pi = (B'_1, A'_1, B'_2)$ such that $B'_1 \subseteq B_1$, $A'_1 \subseteq A_1$ and $B_2 = B'_2$. The algorithm in Lemma 14 can be implemented without an additional pass of the stream, because it only accesses the set S and the partial augmenting set $\Phi = (B_1, A_1, B_2)$; see [18, Proof of Lemma 38].

Finally, we output the set $S \oplus \Pi$. By the same argument as Proof of Theorem 1, the set $S \oplus \Pi$ is a $(2/3 - \varepsilon)$ -approximate solution, which completes the proof. ◀

7 Concluding Remarks

We have observed that a $(2/3 - \varepsilon)$ -approximate solution can be obtained by terminating Blikstad's [11] algorithm early. Then, we obtained a deterministic nearly-linear-independence-query $(2/3 - \varepsilon)$ -approximation algorithm for the matroid intersection problem.

We were also able to use this observation in the streaming model of computation. Then, we obtain a $(2/3 - \varepsilon)$ -approximation constant-pass semi-streaming algorithm for the matroid intersection problem. This result is a generalization of the $(2/3 - \varepsilon)$ -approximation bipartite matching semi-streaming algorithm of Feigenbaum–Kannan–McGregor–Suri–Zhang [27], who initiated the study of matching algorithms in the semi-streaming model. Soon after, McGregor [50] first obtained a $(1 - \varepsilon)$ -approximation constant-pass semi-streaming algorithm

for the maximum matching problem. Since then, there have been many studies for $(1 - \varepsilon)$ -approximation constant-pass semi-streaming algorithms for the maximum matching problem in the literature [1, 8, 24, 25, 30, 31, 40, 42, 59]. Then, it is natural to ask whether we can obtain a $(1 - \varepsilon)$ -approximation constant-pass semi-streaming algorithm for the matroid intersection problem.

In the current Blikstad’s algorithm for finding an augmenting set, it is necessary to perfectly find an augmenting set of length $2k$ before finding an augmenting set of length $2k + 2$. By finding a maximal common independent set, we can perfectly find an augmenting set of length 2. This means that we can start from a state where shortest augmenting paths have length 4 or greater. In our current algorithm, since we do not perfectly find an augmenting set of length 4, we are unable to find an augmenting set of length 6 or greater. Thus, our current algorithm achieves only a $(2/3 - \varepsilon)$ -approximation rather than a $(1 - 1/k - \varepsilon)$ -approximation. However, we believe our study could be an important step to obtain a deterministic nearly-linear-independence-query $(1 - \varepsilon)$ -approximation algorithm.

References

- 1 Kook Jin Ahn and Sudipto Guha. Laminar families and metric embeddings: Non-bipartite maximum matching problem in the semi-streaming model. *arXiv preprint arXiv:1104.4058*, 2011. doi:10.48550/arXiv.1104.4058.
- 2 Kook Jin Ahn and Sudipto Guha. Linear programming in the semi-streaming model with application to the maximum matching problem. *Information and Computation*, 222:59–79, 2013. Announced at ICALP 2011. doi:10.1016/j.ic.2012.10.006.
- 3 Kook Jin Ahn and Sudipto Guha. Access to data and number of iterations: Dual primal algorithms for maximum matching under resource constraints. *ACM Transactions on Parallel Computing (TOPC)*, 4(4):1–40, 2018. Announced at SPAA 2015. doi:10.1145/3154855.
- 4 Martin Aigner and Thomas A Dowling. Matching theory for combinatorial geometries. *Transactions of the American Mathematical Society*, 158(1):231–245, 1971. doi:10.1090/S0002-9947-1971-0286689-5.
- 5 Sepehr Assadi. A simple $(1 - \varepsilon)$ -approximation semi-streaming algorithm for maximum (weighted) matching. In *Proceedings of the 7th Symposium on Simplicity in Algorithms (SOSA 2024)*, pages 337–354. SIAM, 2024. doi:10.1137/1.9781611977936.31.
- 6 Sepehr Assadi and Soheil Behnezhad. Beating two-thirds for random-order streaming matching. In *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, volume 198, pages 19:1–19:13, 2021. doi:10.4230/LIPIcs.ICALP.2021.19.
- 7 Sepehr Assadi, Arun Jambulapati, Yujia Jin, Aaron Sidford, and Kevin Tian. Semi-streaming bipartite matching in fewer passes and optimal space. In *Proceedings of the 33rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2022)*, pages 627–669. SIAM, 2022. doi:10.1137/1.9781611977073.29.
- 8 Sepehr Assadi, S Cliff Liu, and Robert E Tarjan. An auction algorithm for bipartite matching in streaming and massively parallel computation models. In *Proceedings of the 4th Symposium on Simplicity in Algorithms (SOSA 2021)*, pages 165–171. SIAM, 2021. doi:10.1137/1.9781611976496.18.
- 9 Kristóf Bérczi, Tamás Király, Yutaro Yamaguchi, and Yu Yokoi. Matroid intersection under restricted oracles. *SIAM Journal on Discrete Mathematics*, 37(2):1311–1330, 2023. doi:10.1137/22m152579x.
- 10 Aaron Bernstein. Improved bounds for matching in random-order streams. *Theory of Computing Systems*, pages 1–15, 2023. Announced at ICALP 2020. doi:10.1007/s00224-023-10155-7.

- 11 Joakim Blikstad. Breaking $O(nr)$ for matroid intersection. In *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, volume 198, pages 31:1–31:17, 2021. doi:10.4230/LIPIcs.ICALP.2021.31.
- 12 Joakim Blikstad. Sublinear-round parallel matroid intersection. In *Proceedings of the 49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, volume 229, pages 25:1–25:17, 2022. doi:10.4230/LIPIcs.ICALP.2022.25.
- 13 Joakim Blikstad, Sagnik Mukhopadhyay, Danupon Nanongkai, and Ta-Wei Tu. Fast algorithms via dynamic-oracle matroids. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC 2023)*, pages 1229–1242, 2023. doi:10.1145/3564246.3585219.
- 14 Joakim Blikstad and Ta-Wei Tu. Efficient matroid intersection via a batch-update auction algorithm. In *Proceedings of the 8th Symposium on Simplicity in Algorithms (SOSA 2025)*, pages 226–237. SIAM, 2025. doi:10.1137/1.9781611978315.18.
- 15 Joakim Blikstad, Jan van den Brand, Sagnik Mukhopadhyay, and Danupon Nanongkai. Breaking the quadratic barrier for matroid intersection. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC 2021)*, pages 421–432, 2021. doi:10.1145/3406325.3451092.
- 16 Niv Buchbinder and Moran Feldman. Deterministic algorithm and faster algorithm for submodular maximization subject to a matroid constraint. In *Proceedings of the 65th Annual Symposium on Foundations of Computer Science (FOCS 2024)*, pages 700–712. IEEE, 2024. doi:10.1109/FOCS61266.2024.00050.
- 17 Amit Chakrabarti and Sagar Kale. Submodular maximization meets streaming: matchings, matroids, and more. *Mathematical Programming*, 154:225–247, 2015. Announced at IPCO 2014. doi:10.1007/s10107-015-0900-7.
- 18 Deeparnab Chakrabarty, Yin Tat Lee, Aaron Sidford, Sahil Singla, and Sam Chiu-wai Wong. Faster matroid intersection. In *Proceedings of the 60th Annual Symposium on Foundations of Computer Science (FOCS 2019)*, pages 1146–1168. IEEE, 2019. doi:10.1109/FOCS.2019.00072.
- 19 Chandra Chekuri, Shalmoli Gupta, and Kent Quanrud. Streaming algorithms for submodular function maximization. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming (ICALP 2015)*, pages 318–330. Springer, 2015. doi:10.1007/978-3-662-47672-7_26.
- 20 Chandra Chekuri and Kent Quanrud. A fast approximation for maximum weight matroid intersection. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2016)*, pages 445–457. SIAM, 2016. doi:10.1137/1.9781611974331.ch33.
- 21 William H Cunningham. Improved bounds for matroid partition and intersection algorithms. *SIAM Journal on Computing*, 15(4):948–957, 1986. doi:10.1137/0215066.
- 22 Jack Edmonds. Submodular functions, matroids, and certain polyhedra. In *Combinatorial Structures and Their Applications*, pages 69–87, 1970.
- 23 Jack Edmonds. Matroid intersection. In *Annals of Discrete Mathematics*, volume 4, pages 39–49. Elsevier, 1979. doi:10.1016/S0167-5060(08)70817-3.
- 24 Sebastian Eggert, Lasse Kliemann, Peter Munstermann, and Anand Srivastav. Bipartite matching in the semi-streaming model. *Algorithmica*, 63(1):490–508, 2012. doi:10.1007/s00453-011-9556-8.
- 25 Sebastian Eggert, Lasse Kliemann, and Anand Srivastav. Bipartite graph matchings in the semi-streaming model. In *Proceedings of the 17th Annual European Symposium on Algorithms (ESA 2009)*, pages 492–503. Springer, 2009. doi:10.1007/978-3-642-04128-0_44.
- 26 Hossein Esfandiari, MohammadTaghi Hajiaghayi, and Morteza Monemizadeh. Finding large matchings in semi-streaming. In *Proceedings of the 16th International Conference on Data Mining Workshops (ICDMW 2016)*, pages 608–614. IEEE, 2016. doi:10.1109/ICDMW.2016.0092.

- 27 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theoretical Computer Science*, 348(2-3):207–216, 2005. Announced at ICALP 2004. doi:10.1016/j.tcs.2005.09.013.
- 28 Moran Feldman, Amin Karbasi, and Ehsan Kazemi. Do less, get more: Streaming submodular maximization with subsampling. *Advances in Neural Information Processing Systems 31: Proceedings of the 32th Annual Conference on Neural Information Processing Systems (Neurips 2018)*, 31, 2018.
- 29 Moran Feldman and Ariel Szarf. Maximum matching sans maximal matching: A new approach for finding maximum matchings in the data stream model. *Algorithmica*, 86(4):1173–1209, 2024. Announced at APPROX/RANDOM 2022. doi:10.1007/s00453-023-01190-4.
- 30 Manuela Fischer, Slobodan Mitrović, and Jara Uitto. Deterministic $(1 + \varepsilon)$ -approximate maximum matching with poly $(1/\varepsilon)$ passes in the semi-streaming model and beyond. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2022)*, pages 248–260, 2022. doi:10.1145/3519935.3520039.
- 31 Buddhima Gamlath, Sagar Kale, Slobodan Mitrovic, and Ola Svensson. Weighted matchings via unweighted augmentations. In *Proceedings of the 38th Symposium on Principles of Distributed Computing (PODC 2019)*, pages 491–500, 2019. doi:10.1145/3293611.3331603.
- 32 Paritosh Garg, Linus Jordan, and Ola Svensson. Semi-streaming algorithms for submodular matroid intersection. *Mathematical Programming*, 197(2):967–990, 2023. Announced at IPCO 2021. doi:10.1007/s10107-022-01858-9.
- 33 Ashish Goel, Michael Kapralov, and Sanjeev Khanna. On the communication and streaming complexity of maximum bipartite matching. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2012)*, pages 468–485. SIAM, 2012. doi:10.1137/1.9781611973099.41.
- 34 Guru Prashanth Guruganesh and Sahil Singla. Online matroid intersection: Beating half for random arrival. In *Proceedings of the 19th International Conference on Integer Programming and Combinatorial Optimization (IPCO 2017)*, pages 241–253. Springer, 2017. doi:10.1007/978-3-319-59250-3_20.
- 35 John E Hopcroft and Richard M Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973. doi:10.1137/0202019.
- 36 Chien-Chung Huang, Naonori Kakimura, and Naoyuki Kamiyama. Exact and approximation algorithms for weighted matroid intersection. *Mathematical Programming*, 177(1-2):85–112, 2019. Announced at SODA 2016. doi:10.1007/s10107-018-1260-x.
- 37 Chien-Chung Huang and François Sellier. Robust sparsification for matroid intersection with applications. In *Proceedings of the 35th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2024)*, pages 2916–2940. SIAM, 2024. doi:10.1137/1.9781611977912.104.
- 38 Chien-Chung Huang, Theophile Thiery, and Justin Ward. Improved multi-pass streaming algorithms for submodular maximization with matroid constraints. In *Proceedings of the 23th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX 2020) and the 24th International Workshop on Randomized and Computation (RANDOM 2020)*, volume 176, pages 62:1–62:19, 2020. doi:10.4230/LIPIcs.APPROX/RANDOM.2020.62.
- 39 Chien-Chung Huang and Justin Ward. FPT-algorithms for the ℓ -matchoid problem with a coverage objective. *SIAM Journal on Discrete Mathematics*, 37(2):1053–1078, 2023. doi:10.1137/21M1442267.
- 40 Shang-En Huang and Hsin-Hao Su. $(1 - \varepsilon)$ -approximate maximum weighted matching in poly $(1/\varepsilon, \log n)$ time in the distributed and parallel settings. In *Proceedings of the 42nd Symposium on Principles of Distributed Computing (PODC 2023)*, pages 44–54, 2023. doi:10.1145/3583668.3594570.

- 41 Sagar Kale and Sumedh Tirodkar. Maximum matching in two, three, and a few more passes over graph streams. In *Proceedings of the 20th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX 2017) and the 21st International Workshop on Randomized and Computation (RANDOM 2017)*, volume 81, pages 15:1–15:21, 2017. doi:10.4230/LIPIcs.APPROX-RANDOM.2017.15.
- 42 Michael Kapralov. Better bounds for matchings in the streaming model. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2013)*, pages 1679–1697. SIAM, 2013. doi:10.1137/1.9781611973105.121.
- 43 Yusuke Kobayashi and Tatsuya Terao. Subquadratic submodular maximization with a general matroid constraint. In *Proceedings of the 51st International Colloquium on Automata, Languages, and Programming (ICALP 2024)*, volume 297, pages 100:1–100:19, 2024. doi:10.4230/LIPIcs.ICALP.2024.100.
- 44 Christian Konrad. A simple augmentation method for matchings with applications to streaming algorithms. In *Proceedings of the 43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018)*, pages 74–1, 2018. doi:10.4230/LIPIcs.MFCS.2018.74.
- 45 Christian Konrad, Frédéric Magniez, and Claire Mathieu. Maximum matching in semi-streaming with few passes. In *Proceedings of the 15th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX 2012) and the 16th International Workshop on Randomized and Computation (RANDOM 2012)*, pages 231–242. Springer, 2012. doi:10.1007/978-3-642-32512-0_20.
- 46 Christian Konrad, Kheeran K. Naidu, and Arun Steward. Maximum matching via maximal matching queries. In *Proceedings of the 40th International Symposium on Theoretical Aspects of Computer Science (STACS 2023)*, volume 254, pages 41:1–41:22, 2023. doi:10.4230/LIPIcs.STACS.2023.41.
- 47 Bernhard H Korte and Jens Vygen. *Combinatorial optimization*. Springer, third edition, 2006.
- 48 Eugene L Lawler. Matroid intersection algorithms. *Mathematical Programming*, 9(1):31–56, 1975. doi:10.1007/BF01681329.
- 49 Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. In *Proceedings of the 56th Annual Symposium on Foundations of Computer Science (FOCS 2015)*, pages 1049–1065. IEEE, 2015. doi:10.1109/FOCS.2015.68.
- 50 Andrew McGregor. Finding graph matchings in data streams. In *Proceedings of the 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX 2005) and the 9th International Workshop on Randomized and Computation (RANDOM 2005)*, pages 170–181. Springer, 2005. doi:10.1007/11538462_15.
- 51 Kazuo Murota. *Matrices and matroids for systems analysis*. Springer, 2010.
- 52 Huy L Nguyen. A note on Cunningham’s algorithm for matroid intersection. arXiv preprint arXiv:1904.04129, 2019. doi:10.48550/arXiv.1904.04129.
- 53 Kent Quanrud. Adaptive sparsification for matroid intersection. In *Proceedings of the 51st International Colloquium on Automata, Languages, and Programming (ICALP 2024)*, volume 297, pages 118:1–118:20, 2024. doi:10.4230/LIPIcs.ICALP.2024.118.
- 54 Kent Quanrud. Faster exact and approximation algorithms for packing and covering matroids via push-relabel. In *Proceedings of the 35th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2024)*, pages 2305–2336. SIAM, 2024. doi:10.1137/1.9781611977912.82.
- 55 András Recski. *Matroid theory and its applications in electric network theory and in statics*, volume 6. Springer Science & Business Media, 2013. doi:10.1007/978-3-662-22143-3.
- 56 Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer, 2003.

- 57 Maiko Shigeno and Satoru Iwata. A dual approximation approach to weighted matroid intersection. *Operations Research Letters*, 18(3):153–156, 1995. doi:10.1016/0167-6377(95)00047-X.
- 58 Tatsuya Terao. Faster matroid partition algorithms. *ACM Transactions on Algorithms (TALG)*, 21(2):1–26, 2025. Announced at ICALP 2023. doi:10.1145/3707208.
- 59 Sumedh Tirodkar. Deterministic algorithms for maximum matching on general graphs in the semi-streaming model. In *Proceedings of the 38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.FSTTCS.2018.39.
- 60 Ta-Wei Tu. Subquadratic weighted matroid intersection under rank oracles. In *Proceedings of the 33rd International Symposium on Algorithms and Computation (ISAAC 2022)*, volume 248, pages 63:1–63:14, 2022. doi:10.4230/LIPIcs.ISAAC.2022.63.
- 61 Vignesh Viswanathan and Yair Zick. A general framework for fair allocation under matroid rank valuations. In *Proceedings of the 24th ACM Conference on Economics and Computation (EC 2023)*, pages 1129–1152, 2023. doi:10.1145/3580507.3597675.