Faster Matroid Partition Algorithms

Tatsuya Terao
Research Institute for Mathematical Sciences, Kyoto University, Japan

Abstract

In the matroid partitioning problem, we are given k matroids $\mathcal{M}_1 = (V, \mathcal{I}_1), \dots, \mathcal{M}_k = (V, \mathcal{I}_k)$ defined over a common ground set V of n elements, and we need to find a partitionable set $S \subseteq V$ of largest possible cardinality, denoted by p. Here, a set $S \subseteq V$ is called partitionable if there exists a partition $(S_1, \dots, S_k)$ of S with $S_i \in \mathcal{I}_i$ for $i = 1, \dots, k$. In 1986, Cunningham [7] presented a matroid partition algorithm that uses $O(np^{3/2} + kn)$ independence oracle queries, which was the previously known best algorithm. This query complexity is $O(n^{5/2})$ when $k \leq n$.

Our main result is to present a matroid partition algorithm that uses $\tilde{O}(k^{1/3}np + kn)$ independence oracle queries, which is $\tilde{O}(n^{7/3})$ when $k \leq n$. This improves upon previous Cunningham's algorithm. To obtain this, we present a new approach *edge recycling augmentation*, which can be attained through new ideas: an efficient utilization of the binary search technique by Nguyễn [25] and Chakrabarty-Lee-Sidford-Singla-Wong [5] and a careful analysis of the number of independence oracle queries. Our analysis differs significantly from the one for matroid intersection algorithms, because of the parameter k. We also present a matroid partition algorithm that uses $\tilde{O}((n + k)\sqrt{p})$ rank oracle queries.

2012 ACM Subject Classification Theory of computation → Discrete optimization; Theory of computation → Algorithm design techniques; Mathematics of computing → Matroids and greedoids

Keywords and phrases Matroid Partition, Matroid Union, Combinatorial Optimization

Digital Object Identifier 10.4230/LIPIcs.ICALP.2023.104

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2303.05920

Funding This work was partially supported by the joint project of Kyoto University and Toyota Motor Corporation, titled "Advanced Mathematical Science for Mobility Society".

Acknowledgements The author thanks Yusuke Kobayashi for his generous support and helpful comments on the manuscript. The author also thanks the three anonymous reviewers for their valuable comments.

1 Introduction

The *matroid partitioning problem*[1] is one of the most fundamental problem in combinatorial optimization. The problem is sometimes introduced as an important matroid problem along with the *matroid intersection problem*; see [28, Section 41–42] and [21, Section 13.5–6]. In the problem, we are given k matroids $\mathcal{M}_1 = (V, \mathcal{I}_1), \dots, \mathcal{M}_k = (V, \mathcal{I}_k)$ defined over a common ground set V of n elements, and the objective is to find a partitionable set $S \subseteq V$ of largest possible cardinality, denoted by p. Here, we call a set $S \subseteq V$ partitionable if there exists a partition $(S_1, \dots, S_k)$ of S with $S_i \in \mathcal{I}_i$ for $i = 1, \dots, k$. This problem has a number of applications such as matroid base packing, packing and covering of trees and forests, Shannon switching game. There are much more applications; see [28, Section 42].

[1] The *matroid partitioning problem* is sometimes called simply *matroid partition*. Matroid partition is also called *matroid union* or *matroid sum*.

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 set $S \subseteq V$ and outputs whether $S \in \mathcal{I}$ or not. Some recent studies for fast matroid intersection algorithms also consider a more powerful oracle called *rank oracle*, which takes as input a set $S \subseteq V$ and outputs the size of the maximum cardinality independent subset of S. In the design of efficient algorithms, the goal is to minimize the number of such oracle accesses in a matroid partition algorithm. We consider both independence oracle model and rank oracle model, and present the best query algorithms for both oracle models.

The matroid partitioning problem is closely related to the matroid intersection problem. Actually, the matroid partitioning problem and the matroid intersection problem are polynomially equivalent; see [9, 11].

In the *matroid intersection problem*, we are given two matroids $\mathcal{M}' = (V, \mathcal{I}'), \mathcal{M}'' = (V, \mathcal{I}'')$ defined over a common ground set V of n elements, and the objective is to find a common independent set $S \in \mathcal{I}' \cap \mathcal{I}''$ of largest possible cardinality, denoted by r.

Starting the work of Edmonds [8, 10, 11] in the 1960s, algorithms with polynomial query complexity for the matroid intersection problem have been studied [1–5, 7, 22, 23]. Nguyễn [25] and Chakrabarty-Lee-Sidford-Singla-Wong [5] independently presented a new excellent binary search technique that can find edges in the exchange graph and presented a first combinatorial algorithm that uses $\tilde{O}(nr)$ independence oracle queries[2]. Chakrabarty et al. [5] also presented a $(1 - \epsilon)$ approximation matroid intersection algorithm that uses $\tilde{O}(n^{1.5}/\epsilon^{1.5})$ independence oracle queries. Blikstad-van den Brand-Mukhopadhyay-Nanongkai [4] developed a fast algorithm to solve a graph reachability problem, and broke the $\tilde{O}(n^2)$-independence-oracle-query bound by combining this with previous exact and approximation algorithms. Blikstad [2] improved the independence query complexity of the approximation matroid intersection algorithm. This leads to a randomized matroid intersection algorithm that uses $\tilde{O}(nr^{3/4})$ independence oracle queries, which is currently the best algorithm for the matroid intersection problem for the full range of r. This also leads to a deterministic matroid intersection algorithm that uses $\tilde{O}(nr^{5/6})$ independence oracle queries, which is currently the best deterministic algorithm for the matroid intersection problem for the full range of r.

We can solve the matroid partitioning problem by using the reduction to the matroid intersection problem. A well-known reduction reduces the matroid partition to the matroid intersection whose ground set size is kn. Here, one of the input matroids of this matroid intersection is the direct sum of the k matroids. This leads to a matroid partition algorithm using too many independence oracle queries. Even if we use the currently best algorithm for matroid intersection, the naive reduction leads to a matroid partition algorithm that uses $\tilde{O}(k^2np^{3/4})$ independence oracle queries. Since the matroid partition problem itself is an important problem with several applications, it is meaningful to focus on the query-complexity of the matroid partitioning problem.

A direct algorithm for the matroid partitioning problem was first given by Edmonds in 1968 [8]. Algorithms with polynomial query complexity for the matroid partitioning problem have been studied in the literature [3, 7, 12–14, 20, 27].

Cunningham [7] designed a matroid partition algorithm that uses $O(np^{3/2} + kn)$ independence oracle queries. Cunningham uses a *blocking flow* approach, which is similar to Hopcroft-Karp's bipartite matching algorithm or Dinic's maximum flow algorithm. The independence query complexity of Cunningham's algorithm is $O(n^{5/2})$ when $k \leq n$. Note

[2] The $\tilde{O}$ notation omits factors polynomial in $\log n$.

that $p \leq n$ obviously holds. This was the best algorithm for the matroid partitioning problem for nearly four decades. We study faster matroid partition algorithms by using techniques that were recently developed for fast matroid intersection algorithms.

Our first result is the following theorem, which is obtained by combining Cunningham's technique and the binary search technique by Nguyễn [25] and Chakrabarty et al. [5].

▶ **Theorem 1** (Details in Theorem 14). *There is an algorithm that uses $\tilde{O}(kn\sqrt{p})$ independence oracle queries and solves the matroid partitioning problem.*

The independence query complexity of the algorithm given in Theorem 1 improves upon the one of Cunningham's algorithm [7] when k is small. However, when $k = \Theta(n)$, the independence query complexity of the algorithm given in Theorem 1 is $\tilde{O}(n^{5/2})$, and this query complexity is not strictly less than the one in Cunningham's algorithm.

The setting where $k > n$ is unnatural since there must exist a matroid whose independent set is not involved in the optimal partition. Thus, in this paper, we mainly focus on the case where $k \leq n$. Under this assumption, we sometimes bound the number of queries by a function on a single variable n, where we recall that $p \leq n$. This makes it easy to compare the query complexity of different algorithms.

Our second result is to obtain an algorithm that uses $o(kn\sqrt{p})$ independence oracle queries when k is large. It uses $o(n^{5/2})$ independence oracle queries when $k \leq n$.

▶ **Theorem 2** (Details in Theorem 18). *There is an algorithm that uses $\tilde{O}(k^{1/3}np + kn)$ independence oracle queries and solves the matroid partitioning problem.*

This is the main contribution of this paper. The independence query complexity of the algorithm given in Theorem 2 improves the one of the algorithm given in Theorem 1 when $k = \omega(p^{3/4})$. The independence query complexity of the algorithm given in Theorem 2 is $\tilde{O}(n^{7/3})$ when $k \leq n$. This improves the algorithm by Cunningham [7] and our algorithm given in Theorem 1. It should be emphasized here that this is the first improvement since 1986. We note that this algorithm requires $O(k^{2/3}np)$ time complexity other than independence oracle queries.

We also consider the query complexity in the rank oracle model. Note that the rank oracle is at least as powerful as the independence oracle.

▶ **Theorem 3** (Details in the full version of this paper). *There is an algorithm that uses $\tilde{O}((n + k)\sqrt{p})$ rank oracle queries and solves the matroid partitioning problem.*

The rank query complexity of the algorithm given in Theorem 3 is $\tilde{O}(n^{3/2})$ when $k \leq n$.

1.1 Technical Overview

Cunningham's matroid partition algorithm. The auxiliary graph called *exchange graph* plays an important role in almost all combinatorial algorithms for matroid intersection. In matroid intersection algorithms, we begin with an empty set and repeatedly increase the size of the independent set by augmenting along shortest paths in the exchange graph. In the same way, Knuth [20] and Greene-Magnanti [14] give matroid partition algorithms by using the auxiliary graph with $O(np)$ edges, which we call *compressed exchange graph*.

To improve the running time, Cunningham [7] developed *blocking flow* approach for matroid partition and intersection, which is akin to bipartite matching algorithm by Hopcroft-Karp [17]. The blocking flow approach is applied in each phase of the algorithm. In Hopcroft-Karp's bipartite matching algorithm, we find a maximal set of vertex-disjoint shortest paths and augment along these paths simultaneously. In contrast to this, in a matroid partition algorithm, the augmentations can not be done in parallel, since one augmentation can change the compressed exchange graph. Cunningham showed that we can find multiple augmenting paths of the same length and run all the augmentations in one phase. In Cunningham's matroid partition algorithm, one phase uses only $O(np)$ independence oracle queries (each edge is queried only once in one phase).

Cunningham showed that the number of different lengths of shortest augmenting paths during the algorithm is $O(\sqrt{p})$ and then the number of phases is $O(\sqrt{p})$. Therefore, Cunningham's matroid partition algorithm uses $O(np^{3/2} + kn)$ independence oracle queries in total (enumerating all edges entering sink vertices uses $O(kn)$ independence oracle queries). We note that this query complexity is $O(n^{5/2})$ when $k \leq n$.

Combining blocking flow approach and binary search subroutine. To develop the matroid partition algorithm, given in Theorem 1, that uses $\tilde{O}(kn\sqrt{p})$ independence oracle queries, we combine the blocking flow approach proposed by Cunningham [7] and the binary search procedure proposed by Nguyễn [25] and Chakrabarty et al. [5]. By using the binary search procedure, we obtain an algorithm that uses $\tilde{O}(kn)$ independence oracle queries and performs a breadth first search in the compressed exchange graph. We also obtain an algorithm that uses $\tilde{O}(kn)$ independence oracle queries and runs all the augmentations in a single phase. Since Cunningham showed that the number of phases is $O(\sqrt{p})$, we can easily obtain a matroid partition algorithm that uses $\tilde{O}(kn\sqrt{p})$ independence oracle queries. Our algorithm does not contain technical novelty in a sense that this algorithm is obtained by simply combining Cunningham's technique and the binary search technique by Nguyễn and Chakrabarty et al. Nevertheless, this result is important in a sense that we improve the independence query complexity of a matroid partition algorithm.

Edge Recycling augmentation. In a breadth first search, we need to check, for all vertices v and all indices $i \in [k]$, whether there exists an edge from a vertex v to a vertex $u \in S_i$ in the compressed exchange graph. Then, independence query complexity of a breadth first search of the compressed exchange graph seems to be $\Omega(kn)$, even if we use the binary search procedure. It is not clear whether we can develop a matroid partition algorithm that runs a breadth first search $o(\sqrt{p})$ times, and so, algorithms by the blocking flow approach are now stuck at $\Omega(kn\sqrt{p})$ independence oracle queries. In the setting where $k = \Theta(n)$ and $p = \Theta(n)$, algorithms by the blocking flow approach are stuck at $\Omega(n^{5/2})$ independence oracle queries even if we use the excellent binary search procedure.

In order to break this $O(n^{5/2})$-independence-oracle-query bound, we introduce a new approach *edge recycling augmentation* and develop a matroid partition algorithm whose independence query complexity is sublinear in k. Then we present a matroid partition algorithm that uses $\tilde{O}(n^{7/3})$ independence oracle queries when $k \leq n$.

Our new approach edge recycling augmentation is applied in each phase of the algorithm in the same way as the blocking flow approach. In one phase of edge recycling augmentation, we first compute the edge set E^* in the compressed exchange graph, which uses $O(np)$ independence oracle queries. Then we simply repeat to run a breadth first search and find a shortest path in the compressed exchange graph. This breadth first search is performed by using the information of E^*. The precomputation of E^* may seem too expensive since we have the excellent binary search tool to find edges in the compressed exchange graph. However, we can *recycle* some edges in E^* during the repetition of breadth first searches, which plays an important role in an analysis of our new approach. Note that, edge recycling augmentation runs a breadth first search before every augmentation, while the blocking flow approach runs a breadth first search only once in the beginning of each phase.

Our crucial observation is that all edges entering a vertex in S_i are not changed unless S_i was updated by the augmentation. Then, even after some augmentations, we can use E^* to find edges entering a vertex $u \in S_i$ such that S_i was not updated by the augmentation. This observation is peculiar to the matroid partition. In a breadth first search of the edge recycling augmentation approach, we use the binary search procedure only to find edges entering $u \in S_i$ such that S_i was updated by the augmentation. In one phase, we repeat to run a breadth first search so that the total number of the binary search procedure calls is $O(np)$.

We combine the blocking flow approach algorithm and the edge recycling augmentation approach algorithm. By a careful analysis of independence query complexity, we obtain a matroid partition algorithm that uses $\tilde{O}(k^{1/3}np + kn)$ independence oracle queries.

Note that this edge recycling augmentation approach differs significantly from existing fast matroid intersection algorithms. The key technical contribution of this paper is to introduce this new approach.

1.2 Related Work

Blikstad-Mukhopadhyay-Nanongkai-Tu [3] introduced a new oracle model called dynamic oracle and developed a matroid partitioning algorithm that uses $\tilde{O}((n + r\sqrt{r}) \cdot \mathrm{poly}(k))$ dynamic rank queries, where $r = \max_i \max_{S_i \in \mathcal{I}_i} |S_i|$. Blikstad et al. also obtained an algorithm to solve the k-fold matroid union problem in $\tilde{O}(n\sqrt{r})$ time and dynamic rank queries, which is the special case of the matroid partitioning problem where all matroids $\mathcal{M}_1, \dots, \mathcal{M}_k$ are identical. Quanrud [27] developed an algorithm that solves the k-fold matroid union problem and uses $\tilde{O}(n^{3/2})$ independence oracle queries for the full range of r and k. Quanrud also considered the k-fold matroid union problem in the more general settings where the elements have integral and real-valued capacities.

For certain special matroids, faster matroid partition algorithms are known. For linear matroids, Cunningham [7] presented an $O(n^3 \log n)$-time algorithm that solves the matroid partitioning problem on $O(n)$ matrices that have n columns and at most n rows. For graphic matroids, the k-forest problem is a special case of the matroid partitioning problem. In the problem, we are given an undirected graph and a positive integer k, and the objective is to find a maximum-size union of k forests. Gabow-Westermann [13] presented an $O(\min\{k^{3/2}\sqrt{nm(m + n\log n)}, k^{1/2}m\sqrt{m + n\log n}, kn^2 \log k, \frac{m^2}{k} \log k\})$-time algorithm to solve the k-forest problem, where n and m denote the number of vertices and edges, respectively. Blikstad et al. [3] and Quanrud [27] independently obtained an $\tilde{O}(m + (kn)^{3/2})$ time algorithm to solve the k-forest problem.

Kawase-Kimura-Makino-Hanna [19] studied matroid partitioning problems for various objective functions.

For the weighted matroid intersection, Huang-Kakimura-Kamiyama [18] developed a technique that transforms any unweighted matroid intersection algorithm into an algorithm that solves the weighted case with an $O(W)$ factor. Huang et al. also presented a $(1 - \epsilon)$ approximation weighted matroid intersection algorithm that uses $\tilde{O}(nr^{3/2}/\epsilon)$ independence oracle queries. Chekuri-Quanrud [6] improved the independence query complexity and presented a $(1 - \epsilon)$ approximation weighted matroid intersection algorithm that uses $O(nr/\epsilon^2)$ independence oracle queries, which can be improved by applying more recent faster approximation unweighted matroid intersection algorithm by Chakrabarty et al. [5] and Blikstad [2]. Tu [29] gave a weighted matroid intersection algorithm that uses $\tilde{O}(nr^{3/4} \log W)$ rank oracle queries, which also uses the binary search procedure proposed by Nguyễn [25] and Chakrabarty et al. [5].

For matroids of rank $n/2$, Harvey [15] showed a lower bound of $(\log_2 3)n - o(n)$ independence oracle queries for matroid intersection. Blikstad-Mukhopadhyay-Nanongkai-Tu [3] showed super-linear $\Omega(n \log n)$ query lower bounds for matroid intersection and partitioning problem in their dynamic-rank-oracle and the independence oracle models.

1.3 Paper Organization

In Section 2, we introduce the notation and the known results for matroid partition and intersection. Next, in Section 3, we present our matroid partition algorithm using blocking flow approach. Then, in Section 4, we present our new approach edge recycling augmentation and our faster matroid partition algorithm for large k. Finally, in Section 5 we conclude by mentioning several open problems relevant to our work.

2 Preliminaries

2.1 Matroids

Notation. For a positive integer a, we denote $[a] = \{1, \ldots, a\}$. For a finite set X, let $\#X$ and $|X|$ denote the cardinality of X, which is also called the *size* of X. We will often write $A+x := A \cup \{x\}$ and $A-x := A \setminus \{x\}$. We will also write $A+B := A \cup B$ and $A-B := A \setminus B$, when no confusion can arise.

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

(Downward closure) 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 $x \in S \setminus S'$ such that $S' + x \in \mathcal{I}$.

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

Matroid Intersection. Given two matroids $\mathcal{M}' = (V, \mathcal{I}')$, $\mathcal{M}'' = (V, \mathcal{I}'')$, we define their *intersection* by $(V, \mathcal{I}' \cap \mathcal{I}'')$. The *matroid intersection problem* asks us to find the largest common independent set, whose cardinality we denote by r. Note that the intersection of matroids is not a matroid in general and the problem to find a maximum common independent set of more than two matroids is NP-hard.

Matroid Partition (Matroid Union). Given k matroids $\mathcal{M}_1 = (V, \mathcal{I}_1), \ldots, \mathcal{M}_k = (V, \mathcal{I}_k)$, $S \subseteq V$ is called *partitionable* if there exists a partition $(S_1, \ldots, S_k)$ of S such that $S_i \in \mathcal{I}_i$ for $i \in [k]$. The *matroid partitioning problem* asks us to find the largest partitionable set, whose cardinality we denote by p. Let $\tilde{\mathcal{I}}$ be the family of partitionable subset of V. Then, $(V, \tilde{\mathcal{I}})$ is called the *union* or *sum* of k matroids $\mathcal{M}_1 \ldots, \mathcal{M}_k$. Note that Nash-Williams Theorem [24] states that the union $(V, \tilde{\mathcal{I}})$ of the k matroids is also a matroid.

Oracles. Throughout this paper, we assume that we can only access a matroid $\mathcal{M} = (V, \mathcal{I})$ through an oracle. Given a subset $S \subseteq V$, an *independence* oracle outputs whether $S \in \mathcal{I}$ or not. Given a subset $S \subseteq V$, a *rank* oracle outputs $\mathrm{rank}_{\mathcal{M}}(S)$. Since one query of the rank oracle can determine whether a given subset is independent, the rank oracle is more powerful than the independence oracle.

Binary Search Technique. Chakrabarty-Lee-Sidford-Singla-Wong [5] showed that the following procedure can be implemented efficiently by using binary search in the independence oracle model. (This was developed independently by Nguyễn [25].) Given a matroid $\mathcal{M} = (V, \mathcal{I})$, an independent set $S \in \mathcal{I}$, an element $v \in V \setminus S$, and $B \subseteq S$, the objective is to find an element $u \in S$ that is exchangeable with v (that is, $S + v - u \in \mathcal{I}$) or conclude there is no such an element. We skip the proof in this paper; see [5, Section 3] for a proof.

▶ **Lemma 4** (Edge Search via Binary search, Chakrabarty et al. [5], Nguyễn [25]). *There exists an algorithm* `FindOutEdge` *which, given a matroid $\mathcal{M} = (V, \mathcal{I})$, an independent set $S \in \mathcal{I}$, an element $v \in V \setminus S$, and $B \subseteq S$, finds an element $u \in B$ such that $S + v - u \in \mathcal{I}$ or otherwise determine that no such element exists, and uses $O(\log |B|)$ independence queries.*

2.2 Techniques for Matroid Intersection

Here we provide known results about the matroid intersection.

▶ **Definition 5** (Exchange Graph). *Consider a common independent set $S \in \mathcal{I}' \cap \mathcal{I}''$. 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*

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

▶ **Lemma 6** (Shortest Augmenting Path). *Let $s, v_1, v_2, \ldots, v_{l-1}, t$ be a shortest (s, t)-path in the exchange graph $G(S)$ relative to a common independent set $S \in \mathcal{I}' \cap \mathcal{I}''$. Then $S' = S + v_1 - v_2 + \cdots - v_{l-2} + v_{l-1} \in \mathcal{I}' \cap \mathcal{I}''$.*

In a matroid intersection algorithm, we begin with an empty set S. Then we repeat to find an augmenting path in the exchange graph $G(S)$ and to update the current set S. If there is no (s, t)-path in the exchange graph $G(S)$, then S is a common independent set of maximum size. If there is an (s, t)-path in the exchange graph $G(S)$, then we pick a shortest path and obtain a common independent set $S' \in \mathcal{I}' \cap \mathcal{I}''$ of $|S| + 1$ elements.

Cunningham's matroid intersection algorithm [7] and recent faster matroid intersection algorithms [2, 4, 5, 25] rely on the following lemma.

▶ **Lemma 7** (Cunningham [7]). *For any two matroids $\mathcal{M}' = (V, \mathcal{I}')$ and $\mathcal{M}'' = (V, \mathcal{I}'')$, if the length of the shortest augmenting path in exchange graph $G(S)$ relative to a common independent set $S \in \mathcal{I}' \cap \mathcal{I}''$ is at least d, then $|S| \geq (1 - \frac{O(1)}{d}) \cdot r$, where r is the size of a largest common independent set.*

Cunningham's [7] matroid intersection algorithm by the blocking flow approach relies on the following monotonicity lemma.

▶ **Lemma 8** (Monotonicity Lemma, [5, 7, 16, 26]). *For any two matroids $\mathcal{M}' = (V, \mathcal{I}')$ and $\mathcal{M}'' = (V, \mathcal{I}'')$, suppose we obtain a common independent set $S' \in \mathcal{I}' \cap \mathcal{I}''$ by augmenting $S \in \mathcal{I}' \cap \mathcal{I}''$ along a shortest augmenting path in $G(S)$. Note that $|S'| > |S|$. Let d denote the distance in $G(S)$ and d' denote the distance in $G(S')$. Then for all $v \in V$,*
- **(i)** *If $d(s, v) < d(s, t)$, then $d'(s, v) \geq d(s, v)$. If $d(v, t) < d(s, t)$, then $d'(v, t) \geq d(v, t)$.*
- **(ii)** *If $d(s, v) \geq d(s, t)$, then $d'(s, v) \geq d(s, t)$. If $d(v, t) \geq d(s, t)$, then $d'(v, t) \geq d(s, t)$.*

2.3 Compressed Exchange Graph for Matroid Partition

The matroid partitioning problem can be solved by a matroid intersection algorithm. Let $\hat{V} = V \times [k]$, and define

$$\begin{aligned}\hat{\mathcal{I}}' &= \{\hat{I} \subseteq \hat{V} \mid \forall v \in V, \#\{i \in [k] \mid (v,i) \in \hat{I}\} \leq 1\},\\ \hat{\mathcal{I}}'' &= \{\hat{I} \subseteq \hat{V} \mid \forall i \in [k], \{v \in V \mid (v,i) \in \hat{I}\} \in \mathcal{I}_i\}.\end{aligned}$$

Then, $\hat{\mathcal{M}}' = (\hat{V}, \hat{\mathcal{I}}')$ is a partition matroid. Since $\hat{\mathcal{M}}'' = (\hat{V}, \hat{\mathcal{I}}'')$ is the direct sum of matroids $(V, \mathcal{I}_i)$ for all $i \in [k]$, it is also a matroid. Then, the family of partitionable subsets of V can be represented as

$$\{S \subseteq V \mid \exists \pi \colon S \to [k], \{(v, \pi(v)) \mid v \in S\} \in \hat{\mathcal{I}}' \cap \hat{\mathcal{I}}''\}.$$

Therefore, we can solve the matroid partitioning problem by computing a common independent set of maximum size in $\mathcal{I}'$ and $\mathcal{I}''$. However, we might use too many independence oracle queries when solving the matroid partitioning problem by using this reduction to the matroid intersection problem. This is due to the following reasons. When solving the matroid intersection problem that was reduced by the matroid partitioning problem, the size of the ground set of that matroid intersection problem is $O(kn)$, and then the number of edges in the exchange graph is $O(knp)$, which depends heavily on k. Furthermore, since we consider the total query complexity of the independence oracle of each matroid $\mathcal{M}_i = (V, \mathcal{I}_i)$ for all $i \in [k]$, the query complexity of the independence query of the matroid $\hat{\mathcal{M}}'' = (\hat{V}, \hat{\mathcal{I}}'')$ also depends heavily on k.

Then, to improve the running time, Knuth [20] and Greene-Magnanti [14] give a matroid partition algorithm that uses the following auxiliary graph with $O(np)$ edges, which we call *compressed exchange graph*.

▶ **Definition 9** (Compressed Exchange Graph [7, 14, 20]). *Consider a partition $(S_1, \dots, S_k)$ of $S \subseteq V$ such that $S_i \in \mathcal{I}_i$ for all $i \in [k]$. The* compressed exchange graph *is defined as a directed graph $G(S_1, \dots, S_k) = (V \cup \{s, t_1, \dots, t_k\}, E)$, with $s, t_1, \dots, t_k \notin V$ and $E = E' \cup E_s \cup E_t$, where*

$$\begin{aligned}E' &= \{(v,u) \mid \exists i \in [k], u \in S_i, S_i + v \notin \mathcal{I}_i, S_i + v - u \in \mathcal{I}_i\},\\ E_s &= \{(s,v) \mid v \in V \setminus S\}, \textit{and}\\ E_t &= \bigcup_{i=1}^{k} \{(v, t_i) \mid v \in V \setminus S_i, S_i + v \in \mathcal{I}_i\}.\end{aligned}$$

We set $T = \{t_1, \dots, t_k\}$.

In the matroid partition algorithm, we begin with an empty set S and initialize $S_i = \emptyset$ for all $i \in [k]$. If there is no vertex in T which is reachable from s in the compressed exchange graph $G(S_1, \dots, S_k)$, then S is a partitionable set of maximum size. If there is a path from s to T in the compressed exchange graph, then we pick a shortest path $s, v_1, \dots, v_{l-1}, t_j$. Then we can obtain a partitionable set $S' = S + v_1$ and a partition $(S'_1, \dots, S'_k)$ of S' such that $S'_i \in \mathcal{I}_i$ for all $i \in [k]$. The validity of the algorithm follows from the following two lemmas, which we use throughout this paper. Cunningham [7] showed these lemmas by using the equivalence of the compressed exchange graph for the matroid partition and the exchange graph for the reduced matroid intersection; see [28, Theorem 42.4] for a direct proof that does not use the reduction to the matroid intersection.

▶ **Lemma 10.** *Given a partition $(S_1, \dots, S_k)$ of S such that $S_i \in \mathcal{I}_i$ for all $i \in [k]$, there exists a partitionable set S' whose size is at least $|S|+1$ if and only if there is a vertex $t_j \in T$ that is reachable from s in the compressed exchange graph $G(S_1, \dots, S_k)$.*

▶ **Lemma 11** (Shortest Augmenting Path)**.** *Let $s, v_1, v_2, \dots, v_{l-1}, t_j$ be a shortest (s, T)-path in the compressed exchange graph $G(S_1, \dots, S_k)$. Then $S' = S + v_1$ is a partitionable set.*

We can construct a partition $(S'_1, \dots, S'_k)$ of S' from a partition $(S_1, \dots, S_k)$ of S and an augmenting path in the compressed exchange graph by using the following procedure `Update` (Algorithm 1).

Algorithm 1 `Update`.

Input: a partition $(S_1, \dots, S_k)$ of S $(\subseteq V)$ such that $S_i \in \mathcal{I}_i$ for all $i \in [k]$. an augmenting path $s, v_1, \dots, v_{l-1}, t_j$.
Output: a partition $(S'_1, \dots, S'_k)$ of S' $(\subseteq V)$ such that $S'_i \in \mathcal{I}_i$ for all $i \in [k]$ and $S' = S + v_1$.

For all $i \in [k]$, set $S'_i \leftarrow S_i$
For all $v \in S$, denote by $\pi(v)$ the index such that $v \in S_{\pi(v)}$
for $i \in [l-2]$ **do**
  $S'_{\pi(v_{i+1})} \leftarrow S'_{\pi(v_{i+1})} + v_i - v_{i+1}$
$S'_j \leftarrow S'_j + v_{l-1}$
return a partition $(S'_1, \dots, S'_k)$ of S'

Cunningham [7] observes that the equivalence between the exchange graph for the matroid intersection of two matroids $\hat{\mathcal{M}}' = (\hat{V}, \hat{\mathcal{I}}')$ and $\hat{\mathcal{M}}'' = (\hat{V}, \hat{\mathcal{I}}'')$ and the compressed exchange graph for the matroid partition of k matroids $(V, \mathcal{I}_1), \dots, (V, \mathcal{I}_k)$ to prove the Lemmas 12 and 13 and to develop an efficient algorithm for matroid partition that employs the blocking flow approach. For a fixed partition $(S_1, \dots, S_k)$ of S and an element $v \in S$, let $\pi(v)$ be the index such that $v \in S_{\pi(v)}$. We also denote by $\hat{S}$ the set $\{(v, \pi(v)) \in \hat{V} \mid v \in S\}$. A path $s, v_1, v_2, \dots, v_{l-1}, t_j$ in the compressed exchange graph for the matroid partition corresponds to a path $s, (v_1, \pi(v_2)), (v_2, \pi(v_2)), (v_2, \pi(v_3)), \dots, (v_{l-1}, \pi(v_{l-1})), (v_{l-1}, j), t$ in the exchange graph for the matroid intersection. Then, for all elements $v \in S$, we have $d_{G(S_1,\dots,S_k)}(s, v) = 1 + \frac{1}{2} d_{G(\hat{S})}(s, (v, \pi(v)))$ and $d_{G(S_1,\dots,S_k)}(v, T) = \frac{1}{2} d_{G(\hat{S})}((v, \pi(v)), t)$. We also have $d_{G(S_1,\dots,S_k)}(s, T) = 1 + \frac{1}{2} d_{G(\hat{S})}(s, t)$.

Cunningham [7] uses the following two lemmas to develop an efficient matroid partition algorithm by using blocking flow approach. These lemmas can be shown from the correspondence between the exchange graph and the compressed exchange graph. We also use these two lemmas in our fast matroid partition algorithms.

▶ **Lemma 12** (Cunningham [7])**.** *Given a partition $(S_1, \dots, S_k)$ of S such that $S_i \in \mathcal{I}_i$ for all $i \in [k]$. If the length of a shortest augmenting path in the compressed exchange graph $G(S_1, \dots, S_k)$ is at least d, then $|S| \geq (1 - \frac{O(1)}{d}) \cdot p$, where p is the size of largest partitionable set.*

▶ **Lemma 13** (Monotonicity Lemma [5, 7, 16, 26])**.** *Suppose we obtain a partition $(S'_1, \dots, S'_k)$ of S' by augmenting a partition $(S_1, \dots, S_k)$ of S along a shortest augmenting path in $G(S_1, \dots, S_k)$. Note that $|S'| > |S|$. let d denote the distance in $G(S_1, \dots, S_k)$ and d' denote the distance in $G(S'_1, \dots, S'_k)$. Then for all $v \in V$,*
(i) *If $d(s, v) < d(s, T)$, then $d'(s, v) \geq d(s, v)$. If $d(v, T) < d(s, T)$, then $d'(v, T) \geq d(v, T)$.*
(ii) *If $d(s, v) \geq d(s, T)$, then $d'(s, v) \geq d(s, T)$. If $d(v, T) \geq d(s, T)$, then $d'(v, T) \geq d(s, T)$.*

As we will see later, we use the binary search technique given in Lemma 4 to find edges in the compressed exchange graph under the independence oracle model. Note that the procedure FindOutEdge($\mathcal{M}_i, S_i, v, B$) gives us an efficient way to find edges from the vertex v to a vertex $u \in B(\subseteq S_i)$ in the compressed exchange graph.

3 Blocking Flow Algorithm

In this section, we provide our matroid partition algorithms in the independence oracle model, which is obtained by simply combining the *blocking flow* approach proposed by Cunningham [7] and the binary search search procedure proposed by Nguyễn [25] and Chakrabarty-Lee-Sidford-Singla-Wong [5]. In the full version of this paper, we also present a fast matroid partition algorithm using blocking flow approach in the rank oracle model.

3.1 Blocking Flow Algorithm using Independence Oracle

In this subsection we present our matroid partition algorithm using the blocking flow approach in the independence oracle model. We show the following theorem, which implies Theorem 1.

▶ **Theorem 14.** *There is an algorithm that uses $O(kn\sqrt{p}\log p)$ independence oracle queries and solves the matroid partitioning problem.*

This result improves upon the previously known matroid partition algorithm by Cunningham [7] when $k = o(p)$.

For the proof, we first provide the procedure GetDistanceIndependence (Algorithm 2) that efficiently finds distances from s to every vertex in the compressed exchange graph. This algorithm simply runs a breadth first search by using the procedure FindOutEdge.

Algorithm 2 GetDistanceIndependence.

Input: a partition $(S_1, \dots, S_k)$ of S $(\subseteq V)$ such that $S_i \in \mathcal{I}_i$ for all $i \in [k]$
Output: $d \in \mathbb{R}^{V \cup \{s\} \cup T}$ such that for $v \in V \cup \{s\} \cup T$, d_v is the distance from s to v in $G(S_1, \dots, S_k)$

$d_s \leftarrow 0$
For all $v \in V \setminus S$ let $d_v \leftarrow 1$
For all $v \in S$ let $d_v \leftarrow \infty$
For all $i \in [k]$ let $d_{t_i} \leftarrow \infty$
$Q \leftarrow \{v \in V \setminus S\}$ // Q : queue
For all $i \in [k]$ let $B_i \leftarrow S_i$
while $Q \neq \emptyset$ **do**
  Let v be the element added to Q earliest
  $Q \leftarrow Q - v$
  for $i \in [k]$ *with* $d_{t_i} = \infty$ **do**
    if $v \notin S_i$ *and* $S_i + v \in \mathcal{I}_i$ **then**
      $d_{t_i} \leftarrow d_v + 1$
  for $i \in [k]$ *with* $v \notin S_i$ **do**
    while u = FindOutEdge($\mathcal{M}_i, S_i, v, B_i$) *satisfies* $u \neq \emptyset$ **do**
      $Q \leftarrow Q + u$
      $d_u \leftarrow d_v + 1$
      $B_i \leftarrow B_i - u$
return d

▶ **Lemma 15** (Breadth First Search using Independence Oracle). *Given a partition* $(S_1, \ldots, S_k)$ *of* S $(\subseteq V)$ *such that* $S_i \in \mathcal{I}_i$ *for all* $i \in [k]$*, the procedure* `GetDistanceIndependence` *(Algorithm 2) outputs* $d \in \mathbb{R}^{V \cup \{s\} \cup T}$ *such that, for* $v \in V \cup \{s\} \cup T$*,* d_v *is the distance from* s *to* v *in the compressed exchange graph* $G(S_1, \ldots, S_k)$*. The procedure* `GetDistanceIndependence` *uses* $O(kn \log p)$ *independence oracle queries.*

Proof. The procedure `GetDistanceIndependence` simply performs a breadth first search in the compressed exchange graph $G(S_1, \ldots, S_k)$. Thus, the procedure `GetDistanceIndependence` correctly computes distances from s in $G(S_1, \ldots, S_k)$. Note that each vertex $v \in V$ is added to Q at most once and each vertex $v \in S$ is removed from $B_{\pi(v)}$ at most once. Thus, the number of independence oracle queries used in Line 11 is $O(kn)$. The number of `FindOutEdge` calls that do not output $\emptyset$ is $O(p)$, and the number of `FindOutEdge` calls that output $\emptyset$ is $O(kn)$. Hence, by Lemma 4, the number of independence oracle queries used in Line 14 is $O(kn \log p)$, which completes the proof. ◀

Next we provide our augmentation subroutine for our faster matroid partition algorithm. We implement Cunningham's [7] blocking flow approach for matroid partition by using the binary search procedure proposed by Nguyễn [25] and Chakrabarty et al. [5]. This algorithm is similar to Chakrabarty et al.'s matroid intersection algorithm in the rank oracle model [5]. The implementation is described as `BlockFlowIndependence` in the full version of this paper.

In the procedure `BlockFlowIndependence`, given a partition $(S_1, \ldots, S_k)$ of S, we first compute the distances from s to every vertex in the compressed exchange graph $G(S_1, \ldots, S_k)$ using `GetDistanceIndependence` (Algorithm 2). By using these distances, we divide V into sets $L_1, L_2, \ldots,$ where each L_i has all vertices v such that the distance from s to v is i in the compressed exchange graph $G(S_1, \ldots, S_k)$. Then we search a path $s, a_1, a_2, \ldots, a_{d_T - 1}, a_{d_T}$ in the compressed exchange graph $G(S_1, \ldots, S_k)$, where $a_i \in L_i$ for all $i \in [d_T - 1]$. If we found such a path, we augment a partition $(S_1, \ldots, S_k)$ of S and remove a_i from L_i for all $i \in [d_T - 1]$. Then we search a new path again until no (s, T)-path of length d_T can be found. During the search for such a path, if the procedure concludes that some vertex in L_i is not on such a path, then it removes the vertex from L_i. Note that we write $d_T = \min(d_{t_1}, \ldots, d_{t_k})$.

▶ **Lemma 16** (Blocking Flow using Independence Oracle). *Given a partition* $(S_1, \ldots, S_k)$ *of* S $(\subseteq V)$ *such that* $S_i \in \mathcal{I}_i$ *for all* $i \in [k]$*, the procedure* `BlockFlowIndependence` *outputs a partition* $(S'_1, \ldots, S'_k)$ *of* S' $(\subseteq V)$ *such that* $S'_i \in \mathcal{I}_i$ *for all* $i \in [k]$ *and* $|S'| > |S|$ *and* $d_{G(S'_1, \ldots, S'_k)}(s, T) \geq d_{G(S_1, \ldots, S_k)}(s, T) + 1$*, or a partition* $(S_1, \ldots, S_k)$ *of* S *if no such* S' *exists. The procedure* `BlockFlowIndependence` *uses* $O(kn \log p)$ *independence oracle queries.*

We provide a proof of Lemma 16 in the full version of this paper.

Now we provide a proof of Theorem 14 by using Lemma 12. In our matroid partition algorithm, we simply apply `BlockFlowIndependence` repeatedly until no (s, T)-path can be found.

Proof of Theorem 14. In our algorithm, we start with $S = \emptyset$ and initialize $S_i = \emptyset$ for all $i \in [k]$. Then we apply `BlockFlowIndependence` repeatedly to augment the current partition $(S_1, \ldots, S_k)$ of S until no (s, T)-path can be found in the compressed exchange graph $G(S_1, \ldots, S_k)$.

Since each execution of `BlockFlowIndependence` strictly increases $d_{G(S_1, \ldots, S_k)}(s, T)$, we have $d_{G(S_1, \ldots, S_k)}(s, T) = \Omega(\sqrt{p})$ after $O(\sqrt{p})$ executions of `BlockFlowIndependence`. Lemma 12 implies that, if $d_{G(S_1, \ldots, S_k)}(s, T) = \Omega(\sqrt{p})$, then $|S| \geq p - O(\sqrt{p})$. Then the total number of `BlockFlowIndependence` executions is $O(\sqrt{p}) + O(\sqrt{p}) = O(\sqrt{p})$ in the entire matroid partition algorithm. Lemma 16 implies that one execution of `BlockFlowIndependence` uses $O(kn \log p)$ independence oracle queries, which completes the proof. ◀

In the same way as Chakrabarty et al.'s matroid intersection algorithm in the rank oracle model [5], we easily obtain the following theorem.

▶ **Theorem 17.** *For any $\epsilon > 0$, there is an algorithm that uses $O(kn\epsilon^{-1}\log p)$ independence oracle queries and finds a $(1-\epsilon)$ approximation of the largest partitionable set of k matroids.*

Proof. Similar to the proof of Theorem 14, we start with $S = \emptyset$ and initialize $S_i = \emptyset$ for all $i \in [k]$ and apply `BlockFlowIndependence` repeatedly to augment the current partition $(S_1, \ldots, S_k)$ of S. The only difference is that we apply `BlockFlowIndependence` only ϵ^{-1} times, which uses $O(kn\epsilon^{-1}\log p)$ independence oracle queries.

Each execution of `BlockFlowIndependence` strictly increases $d_{G(S_1,\ldots,S_k)}(s,T)$. Thus, after ϵ^{-1} executions of `BlockFlowIndependence`, we have $d_{G(S_1,\ldots,S_k)}(s,T) = \Omega(\epsilon^{-1})$. Lemma 12 implies that, if $d_{G(S_1,\ldots,S_k)}(s,T) = \Omega(\epsilon^{-1})$, then $|S| \geq p - O(p\epsilon)$, which completes the proof. ◀

4 Faster Algorithm for Large k

In this section, we present an algorithm that uses $o(kn\sqrt{p})$ independence oracle queries when k is large. In subsection 3.1, we have presented the algorithm (Algorithm 2), which runs a breadth first search in the compressed exchange graph and uses $O(kn\log p)$ independence oracle queries. In the evaluation of the independence query complexity of the matroid partition algorithm by the *blocking flow* approach given in section 3, a key observation is that the number of different lengths of shortest augmenting paths during the algorithm is $O(\sqrt{p})$. For now, it is not clear whether we can obtain a matroid partition algorithm that runs a breadth first search $o(\sqrt{p})$ times. Then the blocking flow approaches are now stuck at $\Omega(kn\sqrt{p})$ independence oracle queries. To overcome this barrier and improve upon the algorithm that uses $O(kn\sqrt{p}\log p)$ independence oracle queries given in Theorem 14, we introduce a new approach called *edge recycling augmentation*, which can perform breadth first searches with fewer total independence oracle queries. Our new approach can be attained through new ideas: an efficient utilization of the binary search procedure `FindOutEdge` and a careful analysis of the number of independence oracle queries by using Lemma 12. By combining an algorithm by the blocking flow approach and an algorithm by the edge recycling augmentation approach, we obtain the following theorem, which implies Theorem 2.

▶ **Theorem 18.** *There is an algorithm that uses $O(k^{1/3}np\log p + kn)$ independence oracle queries and solves the matroid partitioning problem. When $k \leq n$, the number of queries is $\tilde{O}(n^{7/3})$.*

This theorem implies that we obtain a matroid partition algorithm that uses $o(kn\sqrt{p})$ independence oracle queries when $k = \omega(p^{3/4})$. We note that this algorithm requires $O(k^{2/3}np)$ time complexity other than independence oracle queries.

In Section 4.1, we present our new approach edge recycling augmentation, and in Section 4.2, we present our faster matroid partition algorithm for large k and give a proof of Theorem 18.

4.1 Edge Recycling Augmentation

In order to select appropriate parameters for our algorithm, we have to determine the value of p. However, the size p of a largest partitionable set is unknown before running the algorithm. Instead of using the exact value of p, we use a $\frac{1}{2}$-approximation $\bar{p}$ for p (that is $\bar{p} \leq p \leq 2\bar{p}$), which can be computed using $O(kn)$ independence oracle queries. It is well known that a

$\frac{1}{2}$-approximate solution for the matroid intersection problem can be found by the following simple greedy algorithm; see [21, Proposition 13.26]. We begin with an empty set. For each element in the ground set, we check whether adding it to the set would result in a common independent set. If it does, we add it to the set. Finally, we obtain a maximal common independent set. We convert this algorithm into the following $\frac{1}{2}$-approximation algorithm (Algorithm 3) for the matroid partitioning problem by utilizing the reduction from matroid partition to the intersection of two matroids $\hat{\mathcal{M}}' = (\hat{V}, \hat{\mathcal{I}}')$ and $\hat{\mathcal{M}}'' = (\hat{V}, \hat{\mathcal{I}}'')$ given in subsection 2.3.

Algorithm 3 $\frac{1}{2}$-`ApproximationMatroidPartition`.

For all $i \in [k]$ let $S_i \leftarrow \emptyset$
for $i \leftarrow 1$ **to** k **do**
  for $v \in V \setminus \left(\bigcup_{j=1}^{i-1} S_j\right)$ **do**
    if $S_i + v \in \mathcal{I}_i$ **then**
      $S_i \leftarrow S_i + v$
return $\bar{p} = |\bigcup_{i=1}^{k} S_i|$

Now we present our new approach *Edge Recycling Augmentation*. Our new approach edge recycling augmentation is applied in each phase of the algorithm. One phase of edge recycling augmentation is described as `EdgeRecyclingAugmentation` (Algorithm 5).

In `EdgeRecyclingAugmentation`, we first compute the edges $E^*(\subseteq V \times S)$ in the compressed exchange graph $G(S_1, \ldots, S_k)$, which uses $O(np)$ independence oracle queries. Note that the compressed exchange graph may be changed by augmentations, that is, augmentations may add or delete several edges in the compressed exchange graph, and so, taking one augmenting path may destroy the set E^* of the edges. However, we notice that we can *recycle* some part of the edge set E^* after the augmentations, which is peculiar to the matroid partition.

In `EdgeRecyclingAugmentation`, we simply repeat to run a breadth first search and then to augment the partitionable set. Unlike `GetDistanceIndependence` (Algorithm 2) in Section 3.1, our BFS *recycles* the precomputed edge set E^*. In one phase, we keep a set J of all indices i such that S_i was updated by the augmentations. Our crucial observation is that no edges, in the compressed exchange graph, entering a vertex in S_i are changed by the augmentations unless augmenting paths contain a vertex in $S_i \cup \{t_i\}$. In contrast to `GetDistanceIndependence` that uses the binary search procedure `FindOutEdge` for all indices $i \in [k]$, our new BFS procedure uses `FindOutEdge` only for indices $i \in J$. We can use E^* to search edges entering a vertex in S_i with $i \notin J$. Then, the BFS based on the ideas described above can be implemented as `EdgeRecyclingBFS` (Algorithm 4).

We also provide a new significant analysis of the number of independence oracle queries in entire our matroid partition algorithm. In `EdgeRecyclingAugmentation`, we repeat to run the breadth first search `EdgeRecyclingBFS` so that the total calls of the binary search procedure is $O(np)$. Then, the number of independence oracle queries used by `EdgeRecyclingBFS` in one call of `EdgeRecyclingAugmentation` is almost equal to the one used by the precomputation of E^*. Hence, one call of `EdgeRecyclingAugmentation` uses $\tilde{O}(np)$ independence oracle queries. The number of calls of `EdgeRecyclingBFS` in `EdgeRecyclingAugmentation` depends on how many edges can not be recycled. Thus, to determine the number of calls of `EdgeRecyclingBFS`, we use the value *sum* in the implementation of `EdgeRecyclingAugmentation` (Algorithm 5). In the entire matroid partition algorithm, we apply `EdgeRecyclingAugmentation` repeatedly. Then, we can obtain a matroid partition algorithm that uses $\tilde{O}(np^{3/2} + kn)$ independence oracle queries. Furthermore, by combining this with the blocking flow approach, the number of total calls of `EdgeRecyclingAugmentation` in the entire matroid partition algorithm can be $O(k^{1/3})$. This leads to obtain a matroid partition algorithm that uses $\tilde{O}(k^{1/3}np + kn)$ independence oracle queries. This analysis differs significantly from that of existing faster matroid intersection algorithms.

For $i \in [k]$, let $F_i (\subseteq V)$ denote the set of vertices adjacent to $t_i \in T$. We first compute the set F_i for all $i \in [k]$ using $O(kn)$ independence oracle queries. Note that, after one augmentation, they can be updated using only $O(n)$ independence oracle queries.

In the following two lemmas, we show the correctness and the independence query complexity of the procedure `EdgeRecyclingAugmentation` (Algorithm 5).

▶ **Lemma 19.** *Given a partition $(S_1, \dots, S_k)$ of $S (\subseteq V)$ such that $S_i \in \mathcal{I}_i$ for all $i \in [k]$, the procedure* `EdgeRecyclingAugmentation` *(Algorithm 5) outputs a partition $(S'_1, \dots, S'_k)$ of $S' (\subseteq V)$ such that $S'_i \in \mathcal{I}_i$ for all $i \in [k]$ and $|S'| \geq |S|$.*

Proof. To prove the correctness of `EdgeRecyclingAugmentation`, we prove the following invariants at the beginning of any iteration of the while loop.

(i) For all $i \in [k] \setminus J$ and all $(v, u) \in V \times S_i$, we have $(v, u) \in E^*$ if and only if $S_i + v - u \in \mathcal{I}_i$ and $S_i + v \notin \mathcal{I}_i$.

(ii) For all $i \in [k]$ and all $v \in V$, we have $v \in F_i$ if and only if $S_i + v \in \mathcal{I}_i$ and $v \notin S_i$.

(iii) For all $i \in [k]$, we have $S_i \in \mathcal{I}_i$.

The invariant is true before the execution of `EdgeRecyclingAugmentation`. Now, assume that the invariants (i)–(iii) hold true at the beginning of an iteration of the while loop. Let a partition $(S_1^{\mathrm{old}}, \dots, S_k^{\mathrm{old}})$ of S^{old} be the partition before the execution of Line 13 and a partition $(S_1^{\mathrm{new}}, \dots, S_k^{\mathrm{new}})$ of S^{new} be the partition after the execution of Line 13. For all $i \in [k] \setminus J$, we have $S_i^{\mathrm{old}} = S_i^{\mathrm{new}}$. Then, invariant (i) remains true. For all $i \in [k] \setminus \{j\}$, we have $|S_i^{\mathrm{old}}| = |S_i^{\mathrm{new}}|$. Hence, for all $i \in [k] \setminus \{j\}$ and all $v \notin S_i^{\mathrm{old}} \cup S_i^{\mathrm{new}}$, we have $S_i^{\mathrm{new}} + v \in \mathcal{I}_i$ if and only if $S_i^{\mathrm{old}} + v \in \mathcal{I}_i$; see [28, Corollary 39.13a] for a proof. Furthermore, for all $i \in [k] \setminus \{j\}$, we have $S_i^{\mathrm{old}} + v \notin \mathcal{I}_i$ for all $v \in S_i^{\mathrm{new}} \setminus S_i^{\mathrm{old}}$ and $S_i^{\mathrm{new}} + v \notin \mathcal{I}_i$ for all $v \in S_i^{\mathrm{old}} \setminus S_i^{\mathrm{new}}$; see [7, Section 5]. Then, invariant (ii) remains true. The procedure `EdgeRecyclingBFS` simply finds a BFS-tree rooted at s by a breadth first search. Thus, if the invariants (i)–(iii) are true, then the procedure `EdgeRecyclingBFS` correctly computes BFS-tree rooted at s. Then, the path P that `EdgeRecyclingBFS` outputs in Line 5 is a shortest augmenting path. Hence, by Lemma 11, invariant (iii) remains true. ◀

▶ **Lemma 20.** *The procedure* `EdgeRecyclingAugmentation` *(Algorithm 5) uses $O(np \log p)$ independence oracle queries.*

Proof of Lemma 20. The number of independence oracle queries used in Line 3 is $O(np)$. Furthermore, the number of independence oracle queries used in Line 14 is $O(np)$, because the number of iterations of the while loop is bounded by p.

Now we show that the number of `FindOutEdge` calls in the entire procedure `EdgeRecyclingAugmentation` is $O(np)$.

In the procedure `EdgeRecyclingBFS`$((S_1, \dots, S_k), E^*, J, \bigcup_{i=1}^{k} F_i)$, each vertex $v \in V$ is added to Q at most once and each vertex $v \in S$ is removed from $B_{\pi(v)}$ at most once, where $\pi(v)$ is the index such that $v \in S_{\pi(v)}$. This means that the number of `FindOutEdge` calls that do not output $\emptyset$ is bounded by p, and the number of `FindOutEdge` calls that output $\emptyset$ is bounded by $n \cdot |J|$. Then, the number of `FindOutEdge` calls in the procedure `EdgeRecyclingBFS` is $O(p + n \cdot |J|)$.

Algorithm 4 `EdgeRecyclingBFS`.

Input: a partition $(S_1, \dots, S_k)$ of S $(\subseteq V)$ such that $S_i \in \mathcal{I}_i$ for all $i \in [k]$, a set $E \subseteq V \times S$, a set $J \subseteq [k]$, a set $F = \{v \in V \mid \exists i \in [k], v \notin S_i, S_i + v \in \mathcal{I}_i\}$.

Output: An augmenting (s, T)-path in $G(S_1, \dots, S_k)$ if one exists.

$Q \leftarrow \{v \in V \setminus S\}$ // Q : `queue`
$B_i \leftarrow S_i$ for all $i \in [k]$
while $Q \neq \emptyset$ **do**
    Let v be the element added to Q earliest
    $Q \leftarrow Q - v$
    if $v \in F$ **then**
        return the shortest augmenting path in the BFS-tree.
    for $i \in J$ **do**
        while $u =$ `FindOutEdge`$(\mathcal{M}_i, S_i, v, B_i)$ *satisfies* $u \neq \emptyset$ **do**
            $Q \leftarrow Q + u$
            $B_i \leftarrow B_i - u$
    for $i \in [k] \setminus J$ **do**
        for $u \in B_i$ *such that* $(v, u) \in E$ **do**
            $Q \leftarrow Q + u$
            $B_i \leftarrow B_i - u$
return NO PATH EXISTS

Algorithm 5 `EdgeRecyclingAugmentation`.

Input: a partition $(S_1, \dots, S_k)$ of S $(\subseteq V)$ such that $S_i \in \mathcal{I}_i$ for all $i \in [k]$, sets $F_i = \{v \in V \setminus S_i \mid S_i + v \in \mathcal{I}_i\}$ for all $i \in [k]$

Output: a partition $(S'_1, \dots, S'_k)$ of S' $(\subseteq V)$ such that $S'_i \in \mathcal{I}_i$ for all $i \in [k]$ and $|S'| \geq |S|$.

$\mathit{sum} \leftarrow 0$
$J \leftarrow \emptyset$
$E^* \leftarrow \{(v, u) \in V \times S \mid \exists i \in [k], u \in S_i, S_i + v \notin \mathcal{I}_i, S_i + v - u \in \mathcal{I}_i\}$
while $\mathit{sum} < 2\bar{p}$ **do**
    $P \leftarrow$ `EdgeRecyclingBFS`$((S_1, \dots, S_k), E^*, J, \bigcup_{i=1}^{k} F_i)$
    if $P =$ NO PATH EXISTS **then**
        break
    For $v \in S$ denote by $\pi(v)$ the index such that $v \in S_{\pi(v)}$
    Denote by $V(P) = \{s, v_1, \dots, v_{l-1}, t_j\}$ the vertices in the path P
    for $i \leftarrow 2$ **to** $l - 1$ **do**
        $J \leftarrow J + \pi(v_i)$
    $J \leftarrow J + j$
    $(S_1, \dots, S_k) \leftarrow$ `Update`$((S_1, \dots, S_k), P)$
    $F_j \leftarrow \{v \in V \mid v \notin S_j$ and $S_j + v \in \mathcal{I}_j\}$
    $\mathit{sum} \leftarrow \mathit{sum} + |J|$
return $(S_1, \dots, S_k)$

Suppose that the procedure `EdgeRecyclingBFS` is called for $J = J_1, J_2, \ldots, J_c$ in the procedure `EdgeRecyclingAugmentation`. Obivously, $c \leq 2\bar{p} = O(p)$. Furthermore, by the condition of the while loop in `EdgeRecyclingAugmentation`, $\sum_{i=1}^{c} |J_i| = O(p)$. Thus, the number of `FindOutEdge` calls in the entire procedure `EdgeRecyclingAugmentation` is $O\left(\sum_{i=1}^{c}(p + n \cdot |J_i|)\right)$, which is $O(np)$. Hence, by Lemma 4, the number of independence oracle queries by `FindOutEdge` in the entire procedure `EdgeRecyclingAugmentation` is $O(np \log p)$, which completes the proof. ◀

At this point, we can obtain a matroid partition algorithm that uses $O(np^{3/2} \log p + kn)$ independence oracle queries. In the algorithm, we first compute $F_i = \{v \in V \setminus S_i \mid S_i + v \in \mathcal{I}_i\}$ for all $i \in [k]$. Next, we apply `EdgeRecyclingAugmentation` repeatedly to augment the current partition $(S_1, \ldots, S_k)$ of S until no (s, T)-path can be found in the compressed exchange graph $G(S_1, \ldots, S_k)$. As we will show later in Lemma 22, the number of independence oracle queries in this algorithm is $O(np^{3/2} \log p + kn)$. In the next subsection, we improve this by combining the algorithm by the blocking flow approach and the algorithm by the edge recycling augmentation approach.

4.2 Going Faster for Large k by Combining Blocking Flow and Edge Recycling Augmentation

We have already presented two algorithms to solve the matroid partitioning problem in the independence oracle model. We combine the algorithm by the *blocking flow* approach and the one by the *edge recycling augmentation* approach. When the distance from s to T in the compressed exchange graph is small, we use the blocking flow approach. On the other hand, when the distance from s to T in the compressed exchange graph is large, we use the edge recycling augmentation approach. The implementation is described as Algorithm 6. Then we obtain a matroid partitioning algorithm that uses $o(kn\sqrt{p})$ independence oracle queries when $k = \omega(p^{3/4})$. This improves upon the algorithm given in Theorem 14 that uses only the blocking flow approach.

Algorithm 6 Faster Matroid Partition Algorithm for Large k.

1 Compute a $\frac{1}{2}$-approximation $\bar{p}$ for p by running $\frac{1}{2}$-`ApproximationMatroidPartition` (Algorithm 3) and determine the value of d.
2 For all $i \in [k]$ let $S_i \leftarrow \emptyset$
3 Apply `BlockFlowIndependence` repeatedly to augment the current partition $(S_1, \ldots, S_k)$ of S until the distance from s to T in the compressed exchange graph $G(S_1, \ldots, S_k)$ is at least d.
4 For all $i \in [k]$ let $F_i \leftarrow \{v \in V \setminus S_i \mid S_i + v \in \mathcal{I}_i\}$
5 Apply `EdgeRecyclingAugmentation` (Algorithm 5) repeatedly to augment the current partition $(S_1, \ldots, S_k)$ of S and to update F_j with $j \in [k]$ until no (s, T)-path can be found in the compressed exchange graph $G(S_1, \ldots, S_k)$.

The algorithm is parametrized by an integer d which we set in the end. To analyze the independence query complexity of Algorithm 6, we first show that Line 3 uses $\tilde{O}(knd)$ independence oracle queries and Line 5 uses $\tilde{O}\left(\frac{p^{3/2}n}{d^{1/2}}\right)$ independence oracle queries.

▶ **Lemma 21.** *Line 3 of Algorithm 6 uses $O(knd \log p)$ independence oracle queries.*

Proof. Lemma 16 implies that the distance from s to T in the compressed exchange graph increases by at least 1 after the execution of `BlockFlowIndependence`. Consequently, the number of calls of `BlockFlowIndependence` is bounded by d. Furthermore, Lemma 16 implies that the number of independence oracle queries in one call of `BlockFlowIndependence` is $O(kn \log p)$, which completes the proof. ◀

▶ **Lemma 22.** *Line 5 of Algorithm 6 uses* $O\left(\frac{p^{3/2}n}{d^{1/2}} \log p\right)$ *independence oracle queries.*

Proof. Let m denote the number of calls of `EdgeRecyclingAugmentation` in Line 5 of Algorithm 6. By Lemma 20, we only have to show that $m = O\left(\sqrt{\frac{p}{d}}\right)$. For $i \in [m]$, let c_i denote the number of augmenting paths found in the i-th call of `EdgeRecyclingAugmentation`. For $i \in [m-1] \cup \{0\}$, we write $s_i = \sum_{j=i+1}^{m} c_j$.

We first show the following two claims.

▷ Claim 23. There is a positive constant C such that $c_i \geq C\sqrt{s_i}$ for all $i \in [m-1]$.

Proof. Let $i \in [m-1]$. We denote by l_i the length of the augmenting path found in the last `EdgeRecyclingBFS` in the i-th call of `EdgeRecyclingAugmentation`. Lemma 12 implies that $s_i = O\left(\frac{p}{l_i}\right)$. We note that, by Lemma 13, the length of shortest augmenting paths never decreases as the partitionable set size increases.

In the i-th call of `EdgeRecyclingAugmentation`, the sum of the sizes of J is upper bounded by $c_i^2 \cdot l_i$, because the size of J is upper bounded by $c_i \cdot l_i$. Furthermore, by the condition of the while loop in `EdgeRecyclingAugmentation`, the sum of the sizes of J is at least $2\bar{p}(\geq p)$. Thus, we obtain $c_i^2 \cdot l_i \geq p$, and then we have $c_i^2 \geq \frac{p}{l_i}$. Since $s_i = O\left(\frac{p}{l_i}\right)$, we have $\sqrt{s_i} = O(c_i)$, which completes the proof. ◁

▷ Claim 24. For all $i \in [m-1]$, we have $\frac{C}{\sqrt{1+C}} \leq \int_{s_i}^{s_{i-1}} \frac{dx}{\sqrt{x}}$.

Proof. Let $i \in [m-1]$. Since $c_i \geq C\sqrt{s_i}$ by Claim 23, we obtain

$$\begin{aligned}\int_{s_i}^{s_{i-1}} \frac{dx}{\sqrt{x}} = \int_{s_i}^{s_i+c_i} \frac{dx}{\sqrt{x}} \geq \int_{s_i}^{s_i+C\sqrt{s_i}} \frac{dx}{\sqrt{x}} \geq \int_{s_i}^{s_i+C\sqrt{s_i}} \frac{dx}{\sqrt{s_i+C\sqrt{s_i}}} \\ = \frac{C\sqrt{s_i}}{\sqrt{s_i+C\sqrt{s_i}}} = \frac{C}{\sqrt{1+C\frac{1}{\sqrt{s_i}}}} \geq \frac{C}{\sqrt{1+C}},\end{aligned}$$

which completes the proof. ◁

By Claim 24, $m-1 = \sum_{i=1}^{m-1} 1 \leq \frac{\sqrt{1+C}}{C} \sum_{i=1}^{m-1} \int_{s_i}^{s_{i-1}} \frac{dx}{\sqrt{x}} = O\left(\int_{s_{m-1}}^{s_0} \frac{dx}{\sqrt{x}}\right) = O\left(\sqrt{s_0}\right).$

Since Lemma 12 implies that $s_0 = O\left(\frac{p}{d}\right)$, the number of calls of `EdgeRecyclingAugmentation` in Line 5 of Algorithm 6 is $O\left(\sqrt{\frac{p}{d}}\right)$. By Lemma 20, the proof is complete. ◀

In Algorithm 6, we set a parameter d in order to balance the number of independence oracle queries used in Lines 3 and 5. Thus we obtain the following proof.

Proof of Theorem 18. We set $d = \dfrac{\bar{p}}{k^{2/3}}$ and run Algorithm 6. Then, by Lemmas 21 and 22, the number of independence oracle queries used in Lines 3 and 5 is $O(k^{1/3}np\log p)$. Furthermore, the number of independence oracle queries used in Lines 1 and 4 is $O(kn)$, which completes the proof. ◀

Note that Algorithm 6 requires $O\left(np \cdot \dfrac{p}{d}\right) = O(k^{2/3}np)$ time complexity other than independence oracle queries. This is because we use the edge set E^* of size np in `EdgeRecyclingBFS` and the number of total `EdgeRecyclingBFS` calls in Algorithm 6 is $O\left(\dfrac{p}{d}\right)$.

5 Concluding Remarks

By simply combining Cunningham's algorithm [7] and the binary search technique proposed by Nguyễn [25] and Chakrabarty-Lee-Sidford-Singla-Wong [5], we can not break the $O(n^{5/2})$-independence-query bound for the matroid partitioning problem. However, we introduce a new approach *edge recycling augmentation* and break this barrier and obtain an algorithm that $\tilde{O}(n^{7/3})$ independence oracle queries. This result will be a substantial step forward understanding the matroid partitioning problem.

Our key observation is that some edges in the compressed exchange graph will remain the same after an augmentation, and then we need not query again to find them. That is, we can recycle some edges in the compressed exchange graph. This yields a matroid partition algorithm whose independence query complexity is sublinear in k. This idea is quite simple, and we believe that edge recycling augmentation will be useful in the design of algorithms in future.

In a recent breakthrough, Blikstad-van den Brand-Mukhopadhyay-Nanongkai [4] broke the $\tilde{O}(n^2)$-independence-query bound for matroid intersection. Then it is natural to ask whether we can make a similar improvement for the matroid partition algorithm. However, such an improvement is impossible. As one anonymous reviewer pointed out, it is easy to show that the matroid partitioning problem requires $\Omega(kn)$ independence oracle queries, which is $\Omega(n^2)$ when $k = \Theta(n)$.[3] Then, there is a clear difference between these two problems.

We also consider a matroid partition algorithm in the rank oracle model and present a matroid partition algorithm that uses $\tilde{O}(n^{3/2})$ rank oracle queries when $k \le n$. Blikstad et al. [4] asks whether the tight bounds of the matroid intersection problem are the same under independence oracle model and rank oracle model. The same kind of problem is natural for the matroid partitioning problem. Unlike the matroid intersection problem, we believe there exists a difference between independence oracle and rank oracle in terms of query complexity of the matroid partitioning problem.

References

1 Martin Aigner and Thomas A Dowling. Matching theory for combinatorial geometries. *Transactions of the American Mathematical Society*, 158(1):231–245, 1971.

2 Joakim Blikstad. Breaking $O(nr)$ for matroid intersection. In *the 48th International Colloquium on Automata, Languages, and Programming (ICALP2022)*, pages 31:1–31:17, 2021.

[3] Let $M_1, \dots, M_k$ be matroids of rank 1 defined over a common ground set V of n elements. Now, we construct a bipartite graph $G = (L \cup R, E)$ with $|L| = n$, $|R| = k$ where $(v, i) \in E$ if and only if $\{v\}$ is independent in M_i. Here, the maximum size of a partitionable set is equal to the maximum size of a matching in G. It can be viewed as having edge-query access to this graph G, since it does not make sense to use an independence query to a set of size at least 2. It requires $\Omega(kn)$ edge queries to find the size of a maximum matching; see [30] for details.

3 Joakim Blikstad, Sagnik Mukhopadhyay, Danupon Nanongkai Nanongkai, and Ta-Wei Tu. Fast algorithms via dynamic-oracle matroids. In *the 55th ACM Symposium on Theory of Computing (STOC2023), to appear*, 2023.

4 Joakim Blikstad, Jan van den Brand, Sagnik Mukhopadhyay, and Danupon Nanongkai. Breaking the quadratic barrier for matroid intersection. In *the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC2021)*, pages 421–432, 2021.

5 Deeparnab Chakrabarty, Yin Tat Lee, Aaron Sidford, Sahil Singla, and Sam Chiu-wai Wong. Faster matroid intersection. In *the 60th Annual Symposium on Foundations of Computer Science (FOCS2019)*, pages 1146–1168. IEEE, 2019.

6 Chandra Chekuri and Kent Quanrud. A fast approximation for maximum weight matroid intersection. In *the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA2016)*, pages 445–457. SIAM, 2016.

7 William H Cunningham. Improved bounds for matroid partition and intersection algorithms. *SIAM Journal on Computing*, 15(4):948–957, 1986.

8 Jack Edmonds. Matroid partition. *Mathematics of the Decision Sciences*, 11:335, 1968.

9 Jack Edmonds. Submodular functions, matroids, and certain polyhedra. *Combinatorial Structures and Their Applications, R. Guy, H. Hanani, N. Sauer, and J. Schonheim, eds. New York*, pages 69–87, 1970.

10 Jack Edmonds. Matroid intersection. In *Annals of Discrete Mathematics*, volume 4, pages 39–49. Elsevier, 1979.

11 Jack Edmonds. Submodular functions, matroids, and certain polyhedra. In *Combinatorial Optimization—Eureka, You Shrink! Papers Dedicated to Jack Edmonds 5th International Workshop Aussois, France, March 5–9, 2001 Revised Papers*, pages 11–26. Springer, 2003.

12 András Frank and Zoltán Miklós. Simple push-relabel algorithms for matroids and submodular flows. *Japan Journal of Industrial and Applied Mathematics*, 29:419–439, 2012.

13 Harold Gabow and Herbert Westermann. Forests, frames, and games: algorithms for matroid sums and applications. In *the Twentieth Annual ACM Symposium on Theory of Computing (STOC1988)*, pages 407–421, 1988.

14 Curtis Greene and Thomas L Magnanti. Some abstract pivot algorithms. *SIAM Journal on Applied Mathematics*, 29(3):530–539, 1975.

15 Nicholas JA Harvey. Matroid intersection, pointer chasing, and young's seminormal representation of s_n. In *the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA2008)*, pages 542–549, 2008.

16 Julian Haselmayr. Schnitt von matroiden theorie und algorithmen. Master's thesis, University of Augsberg, 2008.

17 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.

18 Chien-Chung Huang, Naonori Kakimura, and Naoyuki Kamiyama. Exact and approximation algorithms for weighted matroid intersection. In *the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA2016)*, pages 430–444. SIAM, 2016.

19 Yasushi Kawase, Kei Kimura, Kazuhisa Makino, and Hanna Sumita. Optimal matroid partitioning problems. *Algorithmica*, 83:1653–1676, 2021.

20 Donald Ervin Knuth. *Matroid partitioning*. Computer Science Department, Stanford University, 1973.

21 Bernhard H Korte and Jens Vygen. *Combinatorial optimization*. Springer, third edition, 2006.

22 Eugene L Lawler. Matroid intersection algorithms. *Mathematical Programming*, 9(1):31–56, 1975.

23 Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. In *the 56th Annual Symposium on Foundations of Computer Science (FOCS2015)*, pages 1049–1065. IEEE, 2015.

24 Crispin St John Alvah Nash-Williams. An application of matroids to graph theory. In *Theory of Graphs – International Symposium – Théorie des graphes – Journées internationales d'étude Rome*, pages 263–265, 1966.

25 Huy L Nguyen. A note on cunningham's algorithm for matroid intersection, 2019. `arXiv:1904.04129`.

26 Christopher Price. Combinatorial algorithms for submodular function minimization and related problems. Master's thesis, University of Waterloo, 2015.

27 Kent Quanrud. Faster exact and approximation algorithms for packing and covering matroids via push-relabel. *CoRR*, 2023. `arXiv:2303.01478`.

28 Alexander Schrijver et al. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer, 2003.

29 Ta-Wei Tu. Subquadratic weighted matroid intersection under rank oracles. In *the 33rd International Symposium on Algorithms and Computation (ISAAC2022)*, pages 63:1–63:14, 2022.

30 Andrew Chi-Chih Yao. Monotone bipartite graph properties are evasive. *SIAM Journal on Computing*, 17(3):517–520, 1988.