Breaking $O(nr)$ for Matroid Intersection

Joakim Blikstad
KTH Royal Institute of Technology, Stockholm, Sweden

Abstract

We present algorithms that break the $\tilde{O}(nr)$-independence-query bound for the Matroid Intersection problem *for the full range of* r; where n is the size of the ground set and $r \leq n$ is the size of the largest common independent set. The $\tilde{O}(nr)$ bound was due to the efficient implementations [CLSSW FOCS'19; Nguyễn 2019] of the classic algorithm of Cunningham [SICOMP'86]. It was recently broken for large r ($r = \omega(\sqrt{n})$), first by the $\tilde{O}(n^{1.5}/\varepsilon^{1.5})$-query $(1-\varepsilon)$-approximation algorithm of CLSSW [FOCS'19], and subsequently by the $\tilde{O}(n^{6/5}r^{3/5})$-query exact algorithm of BvdBMN [STOC'21]. No algorithm – even an approximation one – was known to break the $\tilde{O}(nr)$ bound for the full range of r. We present an $\tilde{O}(n\sqrt{r}/\varepsilon)$-query $(1-\varepsilon)$-approximation algorithm and an $\tilde{O}(nr^{3/4})$-query exact algorithm. Our algorithms improve the $\tilde{O}(nr)$ bound and also the bounds by CLSSW and BvdBMN for the full range of r.

2012 ACM Subject Classification Theory of computation → Discrete optimization; Theory of computation → Approximation algorithms analysis; Mathematics of computing → Matroids and greedoids

Keywords and phrases Matroid Intersection, Combinatorial Optimization, Approximation Algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.31

Category Track A: Algorithms, Complexity and Games

Funding This project has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme under grant agreement No 71567.

Acknowledgements I thank Danupon Nanongkai and Sagnik Mukhopadhyay for insightful discussions and their valuable comments throughout the development of this work.

1 Introduction

Matroid Intersection is a fundamental problem in combinatorial optimization that has been studied for more than half a century. The classic version of this problem is as follows: *Given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ over a common ground set V of n elements, find the largest common independent set $S^* \in \mathcal{I}_1 \cap \mathcal{I}_2$ by making independence oracle queries*[1] *of the form "Is $S \in \mathcal{I}_1$?" or "Is $S \in \mathcal{I}_2$?" for $S \subseteq V$.* The size of the largest common independent set is usually denoted by r.

Matroid intersection can be used to model many other combinatorial optimization problems, such as bipartite matching, arborescences, spanning tree packing, etc. As such, designing algorithms for matroid intersection is an interesting problem to study.

In this paper, we consider the task of finding a $(1-\varepsilon)$-approximate solution to the matroid intersection problem, that is finding some common independent set S of size at least $(1-\varepsilon)r$. We show an improvement of approximation algorithms for matroid intersection, and as a consequence also obtain an improvement for the *exact* matroid intersection problem.

[1] There are also other oracle models considered in the literature (e.g. rank-oracles), but in this paper we focus on the independence query model. Whenever we say *query* in this paper, we thus mean *independence query*.

Previous work. Polynomial algorithms for matroid intersection started with the work of Edmond's $O(n^2r)$-query algorithms [6, 7, 8] in the 1960s. Since then, there has been a long line of research e.g. [1, 2, 3, 4, 5, 9, 10]. Cunningham [5] designed a $O(nr^{1.5})$-query blocking-flow algorithm in 1986, similar to that of Hopcroft-Karp's bipartite-matching or Dinic's maximum-flow algorithms. Chekuri and Quanrud [4] pointed out that Cunningham's classic algorithm [5] from 1986 is already a $O(nr/\varepsilon)$-query $(1-\epsilon)$-approximation algorithm. Recently, Chakrabarty-Lee-Sidford-Singla-Wong [3] and Nguyễn [11] independently showed how to implement Cunningham's classic algorithm using only $\tilde{O}(nr)$ independence queries. This is akin to spending $\tilde{O}(n)$ queries to find each of the so-called *augmenting paths.* A fundamental question is whether several augmenting paths can be found simultaneously to break the $\tilde{O}(nr)$ bound.

This question has been answered for large r ($r = \omega(\sqrt{n})$), first by the $\tilde{O}(n^{1.5}/\varepsilon^{1.5})$-query $(1-\epsilon)$-approximation algorithm of Chakrabarty-Lee-Sidford-Singla-Wong[2] [3], and very recently by the randomized $\tilde{O}(n^{6/5}r^{3/5})$-query exact algorithm of Blikstad-v.d.Brand-Mukhopadhyay-Nanongkai [2]. Whether we can break the $O(nr)$-query bound *for the full range of* r remained open *even for approximation algorithms.*

Our results. We break the $O(nr)$-query bound for both *approximation* and *exact* algorithms. We first state our results for approximate matroid intersection.[3]

▶ **Theorem 1** (Approximation algorithm). *There is a deterministic algorithm which given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ on the same ground set V, finds a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ with $|S| \geq (1-\varepsilon)r$, using $O\left(\frac{n\sqrt{r\log r}}{\varepsilon}\right)$ independence queries.*

Plugging Theorem 1 in the framework of [2], we get an improved algorithm – more efficient than the previous state-of-the-art – for exact matroid intersection which we state next.

▶ **Theorem 2** (Exact algorithm). *There is a randomized algorithm which given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ on the same ground set V, finds a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ of maximum cardinality r, and w.h.p.*[4] *uses $O(nr^{3/4}\log n)$ independence queries. There is also a deterministic exact algorithm using $O(nr^{5/6}\log n)$ queries.*

▶ Remark 3. Although we only focus on the query-complexity in this paper, we note that the time-complexity of the algorithms are dominated by query-oracle calls. That is, our approximation algorithm runs in $\tilde{O}(n\sqrt{r}\mathcal{T}_{\text{ind}}/\varepsilon)$ time, and the exact algorithms in $\tilde{O}(nr^{3/4}\mathcal{T}_{\text{ind}})$ (randomized) respectively $\tilde{O}(nr^{5/6}\mathcal{T}_{\text{ind}})$ time (deterministic), where $\mathcal{T}_{\text{ind}}$ denotes the time-complexity of the independence-oracle.

1.1 Technical Overview

Approximation algorithm. Our approximation algorithm (Theorem 1) is a modified version of Chakrabarty-Lee-Sidford-Singla-Wong's $\tilde{O}(n^{1.5}/\varepsilon^{1.5})$-query approximation algorithm [3, Section 6]. The algorithm is based on the ideas of Cunningham's classic blocking-flow

[2] In the same paper they also show a $\tilde{O}(n^2r^{-1}\varepsilon^{-2} + r^{1.5}\varepsilon^{-4.5})$-query algorithm.

[3] The $\tilde{O}(n^2r^{-1}\varepsilon^{-2} + r^{1.5}\varepsilon^{-4.5})$-query algorithm of [3] is the only previous algorithm which is more efficient than our algorithm in some range of r and ε. Actually, since the $\tilde{O}(n^2r^{-1}\varepsilon^{-2} + r^{1.5}\varepsilon^{-4.5})$-query algorithm use the $\tilde{O}(n^{1.5}/\varepsilon^{1.5})$ algorithm as a subroutine, we do get a slightly improved version by using our $\tilde{O}(n\sqrt{r}/\varepsilon)$ algorithm as the subroutine instead: $\tilde{O}(n^2r^{-1}\varepsilon^{-2} + r^{1.5}\varepsilon^{-4})$.

[4] *w.h.p. = with high probability* meaning with probability $1 - n^{-c}$ for some arbitrarily large constant c.

algorithm [5] and runs in $O(1/\varepsilon)$ phases, where in each phase the algorithm seeks to find a *maximal* set of *augmentations* in the *exchange graph.* Given a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$, the *exchange graph* $G(S)$ is a directed bipartite graph (with bipartition $(S + \{s,t\}, V \setminus S)$). Finding a shortest (s,t)-path, called an *augmenting path*, in $G(S)$ means one can increase the size of the common independent set S by 1. Since the exchange graph changes after each augmentation,[5] and we do not know how to find a single augmenting path faster than $\Omega(n)$ queries, the need to find several augmentations in parallel arises. [3, Section 6] introduces the notion of *augmenting sets*: a generalization of the classical *augmenting paths* but where one can perform many augmentations in parallel.

So the revised goal of the algorithm is to, in each phase, efficiently find a *maximal augmenting set* (akin to a *blocking-flow* in bipartite matching or flow algorithms). Towards this goal, the algorithm maintains a relaxed version of augmenting set – called a *partial* augmenting set – and keeps *refining* it to make it "better" (i.e. closer to a maximal augmenting set). Here we give two independent improvements on top of the algorithm of [3]:

1. The algorithm of [3] refines the partial augmenting set by a sequence of operations on two adjacent distance layers in the exchange graph. In our algorithm, we instead consider *three* consecutive layers for our basic refinement procedures. This lets us focus our analysis on what happens in S – the "left" side of the bipartite exchange graph – which contains at most r elements in total (in contrast to [3] where the performance analysis is dependent on all n elements). The number of times we need to run the refinement procedures thus depends on r, instead of n, which makes the algorithm faster when $r = o(n)$.
2. When the partial augmenting set is "close enough" to a maximal augmenting set, [3] falls back to finding the remaining augmenting paths one at a time. In our algorithm, we also change to a different procedure when the partial augmenting set is close enough to maximal. The difference is that, instead of finding arbitrary augmenting paths, we find a special type of *valid paths* with respect to the partial augmenting set, so that these paths can be used to further improve (refine) the partial augmenting set. The number of valid paths we need to find is less than the number of augmenting paths [3] needs to find. This decreases the dependency on ε in the final algorithm.

The first improvement (Item 1) replaces the $\sqrt{n}$ term with a $\sqrt{r}$ term in the query complexity of the algorithm. The second improvement (Item 2) shaves off a $1/\sqrt{\varepsilon}$ term from the query complexity. Together they thus bring down the query complexity from $\tilde{O}(\frac{n\sqrt{n}}{\varepsilon\sqrt{\varepsilon}})$ in [3] to $\tilde{O}(\frac{n\sqrt{r}}{\varepsilon})$ as in our Theorem 1. Note that these two improvements are independent of each other, and can be applied individually.

Exact algorithm. To obtain the exact algorithm (Theorem 2), we use the framework of Blikstad-v.d.Brand-Mukhopadhyay-Nanongkai's $\tilde{O}(n^{6/5}r^{3/5})$-query exact algorithm [2]. The main idea of this algorithm is to combine approximation algorithms – which can efficiently find a common independent set only εr away from the optimal – with a randomized $\tilde{O}(n\sqrt{r})$-query subroutine to find each of the remaining *few, very long* augmenting paths. The $\tilde{O}(n^{6/5}r^{3/5})$-query exact algorithm [2] currently uses Chakrabarty-Lee-Sidford-Singla-Wong's $\tilde{O}(n^{1.5}/\varepsilon^{1.5})$ approximation algorithm [3] as a subroutine. Simply replacing it with our improved approximation algorithm (Theorem 1) yields our $\tilde{O}(nr^{3/4})$-query exact algorithm.

[5] Unlike what happens in augmenting path algorithms for flow and bipartite matching, where the underlying graphs remain the same.

2 Preliminaries

We use the standard definitions of *matroid* $\mathcal{M} = (V, \mathcal{I})$; *rank* $\text{rk}(X)$ for any $X \subseteq V$; *exchange graph* $G(S)$ for a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$; and *augmenting paths* in $G(S)$ throughout this paper. For completeness, we define them below. We also need the notions of *augmenting sets* introduced by [3], which we also define in later this section.

Matroids

▶ **Definition 4** (Matroid). A *matroid* is a tuple $\mathcal{M} = (V, \mathcal{I})$ of a *ground set* V of n elements, and non-empty family $\mathcal{I} \subseteq 2^V$ of *independent sets* satisfying
Downward closure: if $S \in \mathcal{I}$, then $S' \in \mathcal{I}$ for all $S' \subseteq S$.
Exchange property: if $S, S' \in \mathcal{I}$, $|S| > |S'|$, then there exists $x \in S \setminus S'$ such that $S' \cup \{x\} \in \mathcal{I}$.

▶ **Definition 5** (Set notation). We will use $A + x$ and $A - x$ to denote $A \cup \{x\}$ respectively $A \setminus \{x\}$, as is usual in matroid intersection literature. We will also use $\bar{A} := V \setminus A$, $A + B := A \cup B$, and $A - B := A \setminus B$.

▶ **Definition 6** (Matroid rank). The *rank* of $A \subseteq V$, denoted by $\text{rk}(A)$, is the size of the largest (or, equivalently, any maximal) independent set contained in A. It is well-known that the rank-function is submodular, i.e. $\text{rk}(A + x) - \text{rk}(A) \geq \text{rk}(B + x) - \text{rk}(B)$ whenever $A \subseteq B \subseteq V$ and $x \in V \setminus B$.[6] Note that $\text{rk}(A) = |A|$ if and only if $A \subseteq \mathcal{I}$.

▶ **Definition 7** (Matroid Intersection). Given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ over the same ground set V, a *common independent set* S is a set in $\mathcal{I}_1 \cap \mathcal{I}_2$. The *matroid intersection problem* asks us to find the largest common independent set – whose cardinality we denote by r. We use rk_1 and rk_2 to be the rank functions of the corresponding matroids.

The Exchange Graph

Many matroid intersection algorithms, e.g. those in [1, 2, 5, 7, 9, 11], are based on iteratively finding *augmenting paths* in the *exchange graph*.

▶ **Definition 8** (Exchange graph). Given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ over the same ground set, and a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$, the *exchange graph* $G(S)$ is a directed bipartite graph on vertex set $V \cup \{s, t\}$ with the following arcs (or directed edges):
1. (s, b) for $b \in \bar{S}$ when $S + b \in \mathcal{I}_1$.
2. (b, t) for $b \in \bar{S}$ when $S + b \in \mathcal{I}_2$.
3. (a, b) for $b \in \bar{S}, a \in S$ when $S + b - a \in \mathcal{I}_1$.
4. (b, a) for $b \in \bar{S}, a \in S$ when $S + b - a \in \mathcal{I}_2$.

We will denote the set of elements at distance k from s by the distance-layer D_k.

▶ **Definition 9** (Shortest augmenting path). A shortest (s, t)-path $p = (s, b_1, a_1, b_2, a_2, \ldots, a_\ell, b_{\ell+1}, t)$ (with $b_i \in \bar{S}$ and $a_i \in S$) in $G(S)$ is called a *shortest augmenting path*. We can *augment* S along the path p to obtain $S' = S \oplus p = S + b_1 - a_1 + b_2 - a_2 \ldots + b_{\ell+1}$, which is well-known to also be a common independent set (with $|S'| = |S| + 1$) [5]. Conversely, there must exist a shortest augmenting path whenever $|S| < r$.

The following lemma is very useful for $(1 - \varepsilon)$-approximation algorithms since it essentially says that one needs only to consider paths up to length $O(\frac{1}{\varepsilon})$.

[6] Usually denoted as the *diminishing returns* property of submodular functions.

▶ **Lemma 10** (Cunningham [5]). *If the length of the shortest (s,t)-path in $G(S)$ is at least $2\ell+2$, then $|S| \geq (1-O(1/\ell))r$.*

▶ **Lemma 11** (Exchange discovery by binary search [3, 11]). *Suppose $\mathcal{M} = (V, \mathcal{I})$ is a matroid, $Y \subseteq X \in \mathcal{I}$, and $b \notin X$ such that $X + b \notin \mathcal{I}$. Then, using $O(\log |Y|)$ independence queries one can find some $a \in Y$ such that $X + b - a \in \mathcal{I}$ or determine that none exist.*[7]

Augmenting Sets

A generalization of the classical *augmenting paths* – called *augmenting sets* – play a key role in the approximation algorithm of [3], and therefore also in the modified version of this algorithm presented in this paper. In order to efficiently find "good" augmenting sets, the algorithm works with a relaxed form of them instead: *partial* augmenting sets. The following definitions and key properties of (partial) augmenting sets are copied from [3] where one can find the corresponding proofs.

▶ **Definition 12** (Augmenting Sets, from [3, Definition 24]). Let $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ and $G(S)$ be the corresponding exchange graph with shortest (s,t)-path of length $2(\ell+1)$ and distance layers $D_1, D_2, \ldots, D_{2\ell+1}$. A collection of sets $\Pi_\ell := (B_1, A_1, B_2, A_2, \ldots, A_\ell, B_{\ell+1})$ form an *augmenting set* (of *width* w) in $G(S)$ if the following conditions are satisfied:

(a) For $1 \leq k \leq \ell+1$, we have $A_k \subseteq D_{2k}$ and $B_k \subseteq D_{2k-1}$.
(b) $|B_1| = |A_1| = |B_2| = \cdots = |B_{\ell+1}| = w$
(c) $S + B_1 \in \mathcal{I}_1$
(d) $S + B_{\ell+1} \in \mathcal{I}_2$
(e) For all $1 \leq k \leq \ell$, we have $S - A_k + B_{k+1} \in \mathcal{I}_1$
(f) For all $1 \leq k \leq \ell$, we have $S - A_k + B_k \in \mathcal{I}_2$

▶ **Definition 13** (Partial Augmenting Sets, from [3, Definition 37]). We say that $\Phi_\ell := (B_1, A_1, B_2, A_2, \ldots, A_\ell, B_{\ell+1})$ forms a *partial augmenting set* if it satisfies the conditions (a), (c), (d), and (e) of an *augmenting set*, plus the following two relaxed conditions:

(b) $|B_1| \geq |A_1| \geq |B_2| \geq \cdots \geq |B_{\ell+1}|$.
(f) For all $1 \leq k \leq \ell$, we have $\mathrm{rk}_2(S - A_k + B_k) = \mathrm{rk}_2(S)$.

▶ **Theorem 14** (from [3, Theorem 25]). *Let $\Pi_\ell := (B_1, A_1, B_2, A_2, \cdots, B_\ell, A_\ell, B_{\ell+1})$ be the an augmenting set in the exchange graph $G(S)$. Then the set $S' := S \oplus \Pi_\ell := S + B_1 - A_1 + B_2 - \cdots + B_\ell - A_\ell + B_{\ell+1}$ is a common independent set.*[8]

We also need the notion of *maximal* augmenting sets, which naturally correspond to a maximal ordered collection of shortest augmenting paths, where, after augmentation, the (s,t)-distance must have increased. The following are due to [3].

▶ **Definition 15** (Maximal Augmenting Sets, from [3, Definition 35]). Let $\Pi_\ell = (B_1, A_1, B_2, \cdots, B_\ell, A_\ell, B_{\ell+1})$ and $\tilde{\Pi}_\ell = (\tilde{B}_1, \tilde{A}_1, \tilde{B}_2, \cdots, \tilde{B}_\ell, \tilde{A}_\ell, \tilde{B}_{\ell+1})$ be two augmenting sets in $G(S)$. We say $\tilde{\Pi}_\ell$ *contains* Π_ℓ if $B_k \subseteq \tilde{B}_k$ and $A_k \subseteq \tilde{A}_k$, for all k. An augmenting set Π_ℓ is called *maximal* if there exists no other augmenting set $\tilde{\Pi}_\ell$ containing Π_ℓ.

▶ **Theorem 16** (from [3, Theorem 36]). *An augmenting set Π_ℓ is maximal if and only if there is no augmenting path of length at most $2(\ell+1)$ in $G(S \oplus \Pi_\ell)$.*

[7] When $X = S$, we can use this to find edges of type 3 and 4 in the exchange graph.
[8] Note that $|S'| = |S| + w$, where w is the width of Π_ℓ. In particular, an augmenting set with width $w = 1$ is exactly an augmenting path.

3 Improved Approximation Algorithm

Our algorithm closely follows the algorithm of Chakrabarty-Lee-Sidford-Singla-Wong [3, Section 6]. The algorithm runs in phases, where in each phase the algorithm finds a maximal set of augmentations to perform, so that the (s,t)-distance in the exchange graph increases between phases. By Lemma 10, only $O(1/\varepsilon)$ phases are necessary.

In the beginning of a phase, the algorithm runs a breadth-first-search to compute the distance layers $D_1, D_2, \ldots D_{2\ell+1}$ in the exchange graph $G(S)$, where S is the current common independent set. The total number of independence queries, across all phases, for these BFS's can be bounded by $O(n\log(r)/\varepsilon)$. We refer to [3, Algorithm 4, Lemma 19, and Proof of Theorem 21] for how to implement such a BFS efficiently.

After the distance layers have been found, the search for a maximal augmenting set begins. We start by summarizing on a high level how the algorithm of [3] does this in two stages:

1. The first stage keeps track of a *partial* augmenting set which it keeps *refining* by a series of operations on adjacent distance layers in the exchange graph, to make it closer to a *maximal* augmenting set.
2. When we are "close enough" to a *maximum* augmenting set, the second stage handles the last few augmenting paths – for which the first stage slows down – by finding the remaining augmenting paths individually one at a time.

Here we give two independent improvements over the algorithm of [3], one for each stage. The first improvement is to replace the refine operations in the first stage by a new subroutine `RefineABA` (Section 3.1.2) working on *three* consecutive layers instead of two. This allows us to measure progress in terms of r instead of n. The second improvement is for the second stage where we, instead of finding arbitrary augmenting paths, work directly on top of the output of the first stage and find a specific type of *valid paths* with respect to the partial augmenting set, using a new a subroutine `RefinePath` (Section 3.2).

3.1 Implementing a Phase: Refining

The basic refining ideas and procedures in this section are the same as in [3]. The goal is to keep track of a partial augmenting set $\Phi_\ell = (B_1, A_1, B_2, \ldots, A_\ell, B_{\ell+1})$ which is iteratively made "better" through some *refine procedures*. Eventually, the partial augmenting set will become a maximal augmenting set, which concludes the phase. Towards this goal, we maintain three types of elements in each layer:

Selected. Denoted by A_k or B_k. These form the partial augmenting set $\Phi_\ell = (B_1, A_1, B_2, \ldots, A_\ell, B_{\ell+1})$.

Removed. Denoted by R_k. These elements are safe to disregard from further computation (i.e. deemed useless) when refining Φ_ℓ towards a maximal augmenting set.

Fresh. Denoted by F_k. These are the elements that are neither *selected* nor *removed*.

Elements can change their types from *fresh* $\to$ *selected* $\to$ *removed*, but never in the other direction. Initially, we start with all elements being fresh.[9] For convenience, we also define "imaginary" layers D_0 and $D_{2\ell+2}$ with $A_0 = R_0 = F_0 = D_0 = A_{\ell+1} = R_{2\ell+2} = F_{2\ell+2} = D_{2\ell+2} = \emptyset$. The algorithm maintains the following *phase invariants* (which are initially satisfied) during the refinement process:

[9] This differs slightly from [3], where the initially B_1 is greedily picked to be maximal so that $S + B_1 \in \mathcal{I}_1$, while the rest of the elements are fresh.

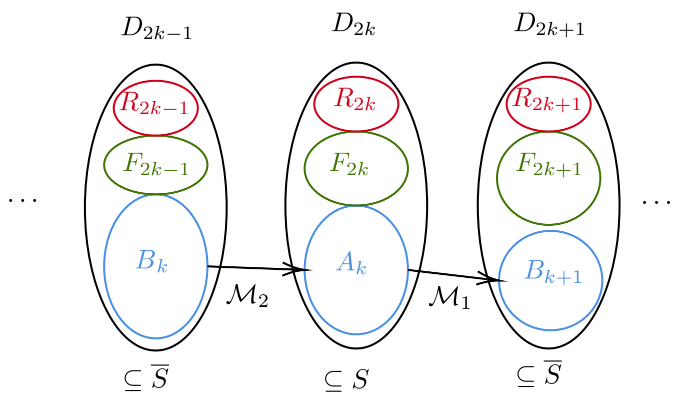

Figure 1 An illustration of a few neighboring layers. Note that $(B_k, R_{2k-1}, F_{2k-1})$ form a partition of odd layer $D_{2k-1} \subseteq \bar{S}$, and (A_k, R_{2k}, F_{2k}) form a partition of even layer $D_{2k} \subseteq S$.

▶ **Definition 17** (Phase Invariants, from [3, Section 6.3.2]). The *phase invariants* are:

(a-b) $\Phi_\ell = (B_1, A_1, B_2, \dots, A_\ell, B_{\ell+1})$ forms a partial augmenting set.[10]

(c) For $1 \le k \le \ell$, for any $X \subseteq B_{k+1} + F_{2k+1} = D_{2k+1} - R_{2k+1}$, if $S - (A_k + R_{2k}) + X \in \mathcal{I}_1$ then $S - A_k + X \in \mathcal{I}_1$. [11]

(d) $\mathrm{rk}_2(W + R_{2k-1}) = \mathrm{rk}_2(W)$ where $W = S - (D_{2k} - R_{2k}) + B_k$.

▶ Remark 18. Invariant (c) essentially says that if R_{2k+1} is "useless", then so is R_{2k}. Similarly, Invariant (d) says that if R_{2k} is "useless", then so is R_{2k-1}. Together they imply that we can safely ignore all the removed elements.

▶ **Lemma 19.** *Suppose that (i) the phase invariants hold; (ii)* $|B_1| = |A_1| = \cdots = |B_{\ell+1}|$*; and (iii)* B_1 *is a maximal subset of* $D_1 \setminus R_1$ *satisfying* $S + B_1 \in \mathcal{I}_1$*. Then* $(B_1, A_1, \dots, B_{\ell+1})$ *is a maximal augmenting set.*

Proof idea. (See [3, Proof of Lemma 44] for a complete proof). If it was not maximal, there exists an augmenting path $(b_1, a_1, \dots, b_{\ell+1})$ in the exchange graph after augmenting along $(B_1, A_1, \dots, B_{\ell+1})$. However, (iii) then says that b_1 must have been removed since it cannot be fresh. But if b_1 is removed, then so was a_1, then so was b_2 etc., by invariants (c) and (d) (this requires a technical, but straightforward, argument). However, $b_{\ell+1}$ cannot have been removed (by invariant (d)), which gives the desired contradiction. ◀

3.1.1 Refining Two Adjacent Layers

We now present the basic refinement procedures from [3], which are operations on neighboring layers. There is some asymmetry in how (odd, even) and (even, odd) layer-pairs are handled, arising from the inherent asymmetry of the independence query between S and $\bar{S}$, but the ideas are the same.

`RefineAB`(k) extends B_{k+1} as much as possible while respecting invariant (a-b) (Lines 1-2). Then a maximal collection of element in A_k which can be "matched" to B_{k+1} is found, and the others elements in A_k are removed (Lines 3-4).

[10] The naming of this invariant as (a-b) is to be consistent with [3] where this condition is split up into two separate items (a) and (b).

[11] An equivalent condition for (c) is: $\mathrm{rk}_1(W - R_{2k}) = \mathrm{rk}_1(W) - |R_{2k}|$, where $W = S - A_k + (D_{2k+1} - R_{2k+1})$.

`RefineBA`(k) finds a maximal subset B_k that can be "matched" to $A_k + F_{2k}$, and removes the other elements of B_k (Lines 1-2). Then A_k is extended with elements from F_{2k} which are the endpoints of the above "matching" (Lines 3-4).

Algorithm 1 `RefineAB`(k). (called `Refine1` in [3, Algorithm 9])

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

Algorithm 2 `RefineBA`(k). (called `Refine2` in [3, Algorithm 10])

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

The following properties of the `RefineAB` and `RefineBA` methods are proven in [3].

▶ **Lemma 20** (from [3, Lemmas 40-42]). *Both `RefineAB` and `RefineBA` preserve the invariants. Also: after `RefineAB`(k) is run, we have $|A_k| = |B_{k+1}|$ (unless $k = 0$). After `RefineBA`(k) is run, we have $|B_k| = |A_k|$ (unless $k = \ell + 1$).*

▶ **Lemma 21** (from [3, Lemma 45]). *`RefineAB` can be implemented with $O(|D_{2k}| + |D_{2k+1}|)$ queries. `RefineBA` can be implemented with $O(|D_{2k-1}| + |D_{2k}|)$ queries.*

▶ **Observation 22.** *Lemma 20 is particularly interesting. It says that at least $|A_k^{old}| - |B_{k+1}^{old}|$ (respectively $|B_k^{old}| - |A_k^{old}|$) elements change type when running `RefineAB` (respectively `RefineAB`).*

▶ Remark 23. Observation 22 is used in [3] to bound the number of times one needs to refine the partial augmenting set. Indeed, every element can only change its type a constant number of times. In a single refinement pass, procedures `RefineAB`(k) and `RefineBA`(k) are called for all k, and we obtain a telescoping sum guaranteeing us that $|B_1^{old}| - |B_{\ell+1}^{old}|$ elements have changed their types. Hence, after $O(\sqrt{n})$ refinement passes we have $|B_1| - |B_{\ell+1}| \leq \sqrt{n}$, and we are "close" to having a maximal augmenting set – only around $\sqrt{n}$ many augmenting paths away. This is essentially what lets [3] obtain their subquadratic $\tilde{O}(n^{1.5}/\text{poly}(\varepsilon))$ algorithm.

3.1.2 Refining Three Adjacent Layers

We are now ready to present the new `RefineABA` method (Algorithm 3), which is **not** present in [3]. This method works similarly to `RefineAB` and `RefineBA`, but on **three** (instead of two) consecutive layers $(D_{2k}, D_{2k+1}, D_{2k+2})$ with the corresponding sets (A_k, B_{k+1}, A_{k+1}).

The motivation for this new procedure is that we can get a stronger version of Observation 22: after running `RefineABA`(k) we want that at least $|A_k^{old}| - |A_{k+1}^{old}|$ element in **even** layers have changed types. Note that there are at most $|S| \leq r$ elements in the even layers (as opposed to n elements in total, which can be much larger), so this means we need to refine the partial augmenting set fewer times when using `RefineABA` compared to when just using `RefineAB` and `RefineBA`. In particular, we will get that after $O(\sqrt{r})$ refinement passes, $|B_1| - |B_{\ell+1}| \leq \sqrt{r}$.

▶ Remark 24. A natural question to ask is if it actually could be the case that only elements in odd layers (i.e. those in $\bar{S}$ which there are up to n many of) change their type (while elements in even layers do not) during the refinement passes in the algorithm of [3] (which only uses the two-layer refinement procedures)? That is, is the new three-layer refinement procedure necessary? The answer is yes. Consider for example the case with 5 layers $B_1 \subseteq D_1; A_1 \subseteq D_2; B_2 \subseteq D_3; A_2 \subseteq D_4; B_3 \subseteq D_5$ where $q := |B_1| = |A_1|$ and $|A_2| = |B_3| = 0$. Refining the consecutive pair (B_1, A_1) or (A_2, B_3) will not do anything. When refining (A_1, B_2) it could be the case that only B_2 increases (say any q-size subset in D_3 can be "matched" with A_1). Similarly, when refining (B_2, A_2) it could be the case that only B_2 decreases (say there is only a single element in D_3 which could be "matched" with anything in the next layer D_4, then it is unlikely that this specific element is already selected in B_2). In this case, we would need to run the two-layer refinement procedures around $|D_3|/q \approx n/q$ times before anything other than B_2 changes. In contrast, the new `RefineABA` method would, when run on (A_1, B_2, A_2), terminate with $|A_1| = |B_2| = |A_2|$ (that is it would have found the "special" element in D_3 the first time it is run).

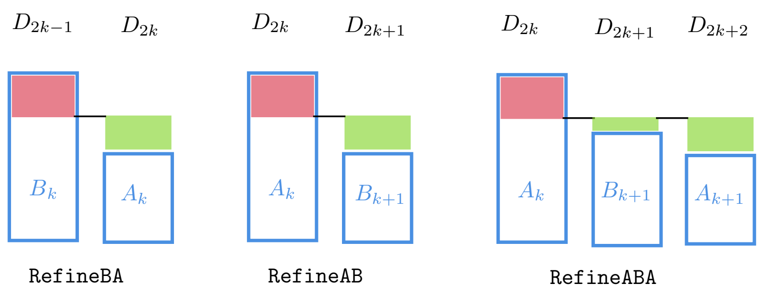

Figure 2 An illustration how the different refine methods change the partial augmenting sets. Newly selected elements are marked in green, while newly removed elements are marked in red.

To explain how `RefineABA` works, let us start with a simple case, namely when $S = \emptyset$, i.e. there is only one layer between s and t in the exchange graph. Here, finding a maximal augmenting set is the same as finding some maximal set B which is independent in both matroids. Running `RefineAB` would extend this B with elements as long as it is independent in the first matroid (ignoring the second matroid), while `RefineBA` would throw away elements from B until it is independent in the second matroid (now ignoring the first matroid). If we just alternate running `RefineAB` and `RefineBA` we would in the worst case need to do this up to n times (which is too expensive). Instead, there is a very simple greedy algorithm that efficiently finds a maximal set B independent in both of the matroids[12]: *for each element, include it in B if this does not break independence for either matroid.* This is akin to how our `RefineABA` method works: it looks at the constraints from both matroids simultaneously (both neighboring layers) and greedily selects B.

In the general case, `RefineABA` can be seen as running `RefineAB` and `RefineBA` simultaneously. The algorithm starts by asserting $|B_{k+1}| = |A_{k+1}|$ (so that $S + B_{k+1} - A_{k+1} \in \mathcal{I}_2$) by running `RefineBA`. So now we have both $S + B_{k+1} - A_k \in \mathcal{I}_1$ and $S + B_{k+1} - A_{k+1} \in \mathcal{I}_2$, and the algorithm proceeds to greedily extend B_{k+1} while it is still consistent with both the previous layer A_k and the next layer $A_{k+1} + F_{2k+2}$. Some care has to be taken here to also mark elements as removed to preserve the phase invariants. Finally, the algorithm decreases the size of A_k, respectively increases the size of A_{k+1}, to both match $|B_{k+1}|$.

[12] This algorithm on its own is a well-known $\frac{1}{2}$-approximation algorithm for matroid intersection.

Algorithm 3 `RefineABA`(k).

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

We now state some properties of `RefineABA`. These properties are relatively straightforward, although technical and notation-heavy, to prove.

▶ **Lemma 25.** *`RefineABA`(k) preserves the phase invariants.*

▶ **Lemma 26.** *After `RefineABA`(k) is run, we have $|A_k| = |B_{k+1}| = |A_{k+1}|$ (unless $k = 0$ or $k = \ell$, where the sets $A_0 = A_{\ell+1} = \emptyset$ are "imaginiary").*

▶ **Lemma 27.** *`RefineABA`(k) uses $O(|D_{2k}| + |D_{2k+1}| + |D_{2k+2}|)$ independence queries.*

Proof of Lemma 25. Intuitively, the only tricky part is showing that invariant (c) is preserved when some x is removed in line 7. We can pretend that we add x to B_{k+1} temporarily, and then run `RefineBA`$(k+1)$ in a way which would remove this x immediately (and thus removing x did indeed preserve the invariants). We present a formal proof below.

We already know that `RefineAB` and `RefineBA` preserve the invariants by Lemma 20, so it suffices to check that the for-loop starting in line 2 preserves the invariants. We verify that this is the case after processing each $x \in F_{2k+1}$ in the for-loop:

Invariant (a-b) holds by design: when x is added to B_{k+1} we know both that $S - A_k + B_{k+1} + x \in \mathcal{I}_1$ and $\mathrm{rk}_2(S - A_{k+1} + B_{k+1})$ cannot decrease. Note also that $\mathrm{rk}_2(S - A_{k+1} + B_{k+1}) \leq \mathrm{rk}_2(S)$ when $k + 1 \leq \ell$ too (so it cannot increase either), since otherwise there must exist some $b \in B_{k+1}$ so that $S + b \in \mathcal{I}_2$ (by the matroid exchange property) which is impossible since we are not in the last layer (the layer preceding t in $G(S)$).

Invariant (c) trivially holds, since the set $B_{k+1} + F_{2k+1}$ will only decrease, which only restricts the choice of $X \subseteq B_{k+1} + F_{2k+1}$.

Invariant (d) will also be preserved. We need to argue that this is the case when x is removed in line 7. Let $W := S - A_{k+1} - F_{2k+2} + B_{k+1} = S - (D_{2k+2} - R_{2k+2}) + B_{k+1}$, and R^{old}_{2k+1} be the set R_{2k+1} before x was added to it. First note that $W \in \mathcal{I}_2$, since this holds after the `RefineBA` call in line 1, (since $|A_{k+1}| = |B_{k+1}|$ after this call) and B_{k+1} is only extended with elements which preserve this property. This means that $\mathrm{rk}_2(W + x) = \mathrm{rk}_2(W) = |W|$, since $W + x = S - A_{k+1} - F_{2k+2} + B_{k+1} + x \notin \mathcal{I}_2$. Since the invariant held before, we also know that $\mathrm{rk}_2(W + R^{old}_{2k+1}) = \mathrm{rk}_2(W) = |W|$. Hence W is a maximal independent (in $\mathcal{M}_2$) subset of $W + R^{old}_{2k+1} + x$, as neither x nor elements from R^{old}_{2k+1} can be used to extend it. Hence $\mathrm{rk}_2(W + R^{old}_{2k+1} + x) = |W| = \mathrm{rk}_2(W)$; that is invariant (d) is preserved. ◀

Proof of Lemma 26. We focus our attention on the `RefineBA` and `RefineAB` calls in lines 8-9, and argue that they do not change B_{k+1}. This would prove the lemma, since by Lemma 20 we would then have $|A_k| = |B_{k+1}|$ and $|B_{k+1}| = |A_{k+1}|$.

Indeed, `RefineBA`$(k+1)$ finds a maximal $B \subseteq B_{k+1}$ such that $S-(D_{2k+2}-R_{2k+2})+B \subseteq \mathcal{I}_2$, and remove all elements not in B from B_{k+1}. Here, $B = B_{k+1}$ will be found, since $S-(D_{2k+2}-R_{2k+2})+B_{k+1} \in \mathcal{I}_2$ after the for-loop in line 2 of `RefineABA`.

Similarly, we see that `RefineAB`(k) finds a maximal $B \subseteq F_{2k+1}$ such that $S-A_k+B_{k+1}+B \in \mathcal{I}_1$, and extend B_{k+1} with this B. However, only $B=\emptyset$ works, since each $x \in F_{2k+1}$ for which $S-A_k+B_{k+1}+x \in \mathcal{I}_1$ was either selected or removed in lines 5 or 7. ◀

Proof of Lemma 27. `RefineAB`(k) uses $O(|D_{2k}|+|D_{2k+1}|)$ queries, and `RefineBA`$(k+1)$ uses $O(|D_{2k+1}|+|D_{2k+2}|)$ queries. The for-loop in line 2 will use $O(|D_{2k+1}|)$ queries. ◀

3.1.3 Refinement Pass

We can now present the full `Refine` method (Algorithm 4), which simply scans over the layers and calls `RefineABA` on them. Our `Refine` is a modified version of `Refine` from [3, Algorithm 11] using our new `RefineABA` method instead of just `RefineAB` and `RefineBA`. Just replacing the `Refine` method in the final algorithm of [3] with our modified `Refine` below leads to an $\tilde{O}(n\sqrt{r}/\varepsilon^{1.5})$-query algorithm (compared to their $\tilde{O}(n^{1.5}/\varepsilon^{1.5})$), and concludes our first improvement (as discussed in Item 1 in Section 1.1).

Algorithm 4 `Refine`(k).

```
1: for $k = \ell,\ \ell-1,\ \ell-2,\ \dots,\ 1,\ 0$ do
2:     RefineABA(k)
```

The following Lemma 28 will be useful to bound the number of `Refine` calls needed in our final algorithm, and closely corresponds to [3, Corollary 43]. Our `Refine` implementation has the advantage that it only counts the elements in the **even** layers, of which there are at most r.

▶ **Lemma 28.** *Let $(B_1^{old}, A_1^{old}, \dots)$ and $(B_1^{new}, A_1^{new}, \dots)$ be the sets before and after `Refine` is run. Then at least $|B_1^{new}|-|B_{\ell+1}^{new}|$ elements in even layers have changed types.*

Proof. Note that whenever A_k changes, it is because some elements changed it types in D_{2k}. In particular, if the size of A_k increases (respectively decreases) by z, at least z elements will change types from fresh to selected (respectively from selected to removed) in D_{2k}.

After the first iteration $|A_\ell| = |B_{\ell+1}^{new}|$, so at least $|A_\ell^{old}|-|B_{\ell+1}^{new}|$ elements in $D_{2\ell}$ changed types. Similarly, after the iteration when $k=i$ (for $1 \le i \le \ell-1$), $|A_i| = |A_{i+1}|$, and hence at least $|A_i^{old}|-|A_i|$ elements in D_{2i} changed types plus at least $|A_{i+1}|-|A_{i+1}^{old}|$ elements in D_{2i+2} changed types.[13] Finally, after the last iteration $|A_1| = |B_1^{new}|$, and hence at least $|B_1^{new}|-|A_1^{old}|$ elements in D_2 changed types.

The above terms telescope, and we conclude that at least $|B_1^{new}|-|B_{\ell+1}^{new}|$ elements in the even layers changed its types when `Refine` was run. ◀

▶ **Lemma 29.** *`Refine` uses $O(n)$ independence queries.*

Proof. This follows directly by Lemma 27. ◀

[13] $|A_{i+1}| \le |A_{i+1}^{old}|$ just before the `RefineABA`(i) call, since earlier iterations can only have decreased the size of $|A_{i+1}|$.

3.2 Refining Along a Path

If we just run `Refine` until we get a maximal augment set (i.e. until $|B_1| = |B_{\ell+1}|$) we need to potentially run `Refine` as many as $\Theta(r)$ times, which needs too many independence queries. Lemma 28 tells us that `Refine` makes the most "progress" while $|B_1| - |B_{\ell+1}|$ is large: in fact, only $O(r/p)$ calls to `Refine` is needed until $|B_1| - |B_{\ell+1}| \le p$. The idea in [3] is thus to stop refining when $|B_1| - |B_{\ell+1}|$ is small enough and fall back to finding augmenting paths one at a time (they prove that one needs to find at most $O((|B_1| - |B_{\ell+1}|)\ell)$ many). We use a similar idea in that we swap to a different procedure when $|B_1| - |B_{\ell+1}|$ is small enough, the difference being that we still work with the partial augmenting set. This will let us show that only $O(|B_1| - |B_{\ell+1}|)$ many "paths" need to be found, saving a factor $\ell \approx \frac{1}{\varepsilon}$ compared to [3].

This section thus describes the second improvement (as discussed in Item 2 in Section 1.1). Note that this improvement is independent of the first improvement (i.e. the three-layer refine). We aim to prove the following lemma.

▶ **Lemma 30.** *There exists a procedure (`RefinePath`, Algorithm 5), which uses $O(n \log r)$ independence queries, preserves the invariants, and either:*
(i) *Increases the size of $B_{\ell+1}$ by at least 1.*
(ii) *Terminates with $(B_1, A_1, \dots, B_{\ell+1})$ being a maximal augmenting set.*

`RefinePath` attempts to find what we call a *valid path*. What we want is a sequence of elements which we can add to the partial augmenting set without violating the invariants and the properties of the partial augmenting set. It turns out (not very surprisingly) that such sequences of elements can be characterized by a notion of *paths* in something which resembles the *exchange graph with respect to our partial augmenting set.* This is what motivates the definition of *valid paths* below.

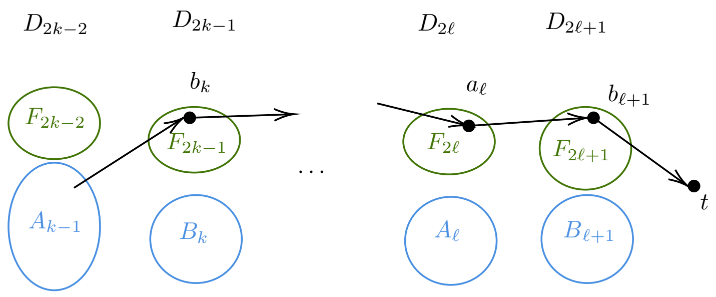

Figure 3 A valid path $(b_k, \dots, a_\ell, b_{\ell+1}, t)$ "starting" from the partial augmenting set at A_{k-1}, so that we can use Lemma 33 and augment along it.

▶ **Definition 31** (Valid path). A sequence $(b_i, a_i, b_{i+1}, \dots, b_{\ell+1}, t)$ (or $(a_i, b_{i+1}, \dots, b_{\ell+1}, t)$) is called a *valid path* (with respect to the partial augmenting set) if for all $k \ge i$:
(a) $a_k \in F_{2k}$ and $b_k \in F_{2k-1}$.
(b) $S + B_{\ell+1} + b_{\ell+1} \in \mathcal{I}_2$.
(c) $S - A_k + B_k - a_k + b_k \in \mathcal{I}_2$.
(d) $S - A_k + B_{k+1} - a_k + b_{k+1} \in \mathcal{I}_1$.

▶ Remark 32. Compare the properties of valid paths with the edges in the exchange graph from Definition 8. A valid path is essentially a path in the exchange graph *after* we have already augmented S by our partial augmenting set (even though this exchange graph is not exactly defined, since it is not guaranteed that S remains a common independent set when augmented by a *partial* augmenting set).

▶ **Lemma 33.** *If $p = (b_i, a_i, b_{i+1}, \dots, b_{\ell+1}, t)$ is a valid path starting at b_i, such that $S - A_{i-1} + B_i + b_i \in \mathcal{I}_1$, then $(B_1, A_1, \dots, B_{i-1}, A_{i-1},\ B_i + b_i, A_i + a_i, \dots, B_{\ell+1} + b_\ell)$ is a partial augmenting set satisfying the invariants.*

Proof. That it forms a partial augmenting set is true by the definition of valid paths, and the fact that $S - A_{i-1} + B_i + b_i \in \mathcal{I}_1$. Indeed, it cannot be the case that $|A_{i-1}| < |B_i + b_i|$ when $i > 1$, since then $\mathrm{rk}_1(S - A_{i-1} + B_i + b_i) > |S| = \mathrm{rk}_1(S)$ implies that some element $x \in (B_i + b_i)$ satisfies $S + x \in \mathcal{I}_1$ (i.e. it is in the first layer D_1) by the exchange property of matroids. Invariants (c) and (d) are trivially true since the sets A_k and B_k are only extended. ◀

The goal of `RefinePath` (Algorithm 5) is thus to find a valid path satisfying the conditions in Lemma 33. Towards this goal, `RefinePath` will start from the last layer $D_{2\ell+1}$ and "scan left" in a breadth-first-search manner while keeping track of valid paths starting at each fresh vertex x (the next element on such a path will be stored as `next`$[x]$). If at some point one valid path can "enter" the partial augmenting set in a layer, we are done and can use Lemma 33. We also show that it is safe (i.e. preserves the invariants) to remove all the fresh elements x for which we cannot find a valid path starting at x.

To efficiently find the "edges" during our breadth-first-search using only independence-queries, we use the binary-search trick from Lemma 11. However, this relies on the partial augmenting set being locally "flat" in the layers we are currently exploring, i.e. $|B_k| = |A_k|$ respectively $|B_k| = |A_{k+1}|$. We can ensure this by running `RefineAB` respectively `RefineBA` while performing the scan.

Now we are ready to present the pseudo-code of the `RefinePath` method (Algorithm 5). Due to the asymmetry between even/odd layers and independence queries, we need to handle moving from layer B to A and from A to B a bit differently, but the ideas are similar.

▶ **Lemma 34.** *`RefinePath` preserves the invariants.*

Proof. The proof is relatively straightforward, but technical. The only non-trivial part is showing that invariants (c) and (d) are preserved after we remove something in line 8 or line 20. Intuitively, if we remove b in line 8, we can instead think of temporarily adding b to B_k and running `RefineBA`(k) in such a way so that b is immediately removed. A similar intuitive argument works for line 20. We next present a formal proof.

We know that `RefineAB` and `RefineBA` preserve the invariants, by Lemma 20. We also know by Lemma 33 that adding a valid path to the partial augmenting set also preserves the invariants. So what remains is to show that the invariants are preserved after:

Line 8. We only need to check invariant (d), the other ones trivially hold. Let $W = S - A_k - F_{2k} + B_k = S - (D_{2k} - R_{2k}) + B_k$ and R^{old}_{2k-1} be R_{2k-1} before b was added to it. Note that b is such that $W + b \notin \mathcal{I}_2$, and we know that $W \subseteq S - A_k + B_k \in \mathcal{I}_2$ and hence $\mathrm{rk}_2(W + R^{old}_{2k-1}) = \mathrm{rk}_2(W) = |W|$ and $\mathrm{rk}_2(W + b) = \mathrm{rk}_2(W) = |W|$. We thus need to show that $\mathrm{rk}_2(W + R^{old}_{2k-1} + b) = |W|$ too, which is clear since W is a maximal independent subset of $W + R^{old}_{2k-1} + b$ (it can neither be extended with elements from R^{old}_{2k-1} nor with b).

Line 20. We only need to check invariant (c), the other ones trivially hold. We imagine we add the $a \in Q$ to R_{2k-2} one-by-one, and show that the invariant (c) is preserved after each such addition. So consider some $a \in Q$ which will be removed, and let R^{old}_{2k-2} be the set R_{2k-2} just before we added a to it. First note that $\mathrm{rk}_1(S - A_{k-1} + B_k + F_{2k-1} - a) = \mathrm{rk}_1(S - A_{k-1} + B_k + F_{2k-1}) - 1 = |S - A_{k-1} + B_k| - 1$, as otherwise there must exist some $b \in F_{2k-1}$ such that $S - A_{k-1} + B_k + b - a \in \mathcal{I}_1$ (by the matroid exchange property),

Algorithm 5 RefinePath.

```
1: for $k = \ell+1,\ \ell,\ \ldots,\ 2,\ 1$ do
                                                        ▷ Process $(B_k, A_k)$
2:     RefineBA($k$)
3:     if some element $a$ was added to $A_k$ in the above refine-call then
4:         Add the valid path starting at next[$a$] to the partial augmenting set
5:         return
6:     for each element $b \in F_{2k-1}$ do
7:         if $S - A_k - F_{2k} + B_k + b \notin \mathcal{I}_2$ then
8:             Remove $b$, that is: $F_{2k-1} \leftarrow F_{2k-1} - b$,   $R_{2k-1} \leftarrow R_{2k-1} + b$
9:         else
10:             Find an $a \in F_{2k}$ such that $S - A_k + B_k + b - a \in \mathcal{I}_2$. Let next[$b$] $= a$.
11:             (Or, if $k = \ell + 1$, just let next[b] $= t$)
                                                        ▷ Process $(A_{k-1}, B_k)$
12:     if some element $b \in F_{2k-1}$ satisfies $S - A_{k-1} + B_k + b \in \mathcal{I}_1$ then
13:         Add the valid path starting at $b$ to the partial augmenting set.
14:         return
15:     RefineAB($k - 1$)
16:     $Q \leftarrow F_{2k-2}$.
17:     for each element $b \in F_{2k-1}$ do
18:         while can find $a \in Q$ such that $S - A_{k-1} + B_k + b - a \in \mathcal{I}_1$ do
19:             $Q \leftarrow Q - a$. Let next[$a$] $= b$.
20:     Remove all elements in $Q$, that is: $F_{2k-2} \leftarrow F_{2k-2} - Q$,   $R_{2k-2} \leftarrow R_{2k-2} + Q$.

21: If we reached here, $(B_1, A_1, \ldots, B_{\ell+1})$ is a maximal augmenting set.
```

and a would have been discovered in line 18 and therefore been removed from Q. So the "return" of adding a to $S - A_{k-1} + B_k + F_{2k-1} - a$ is increasing the rank by 1. Now consider some arbitrary $X \subseteq B_k + F_{2k-1}$ such that $S - A_{k-1} + X - R^{old}_{2k-2} - a \in \mathcal{I}_1$. We need to show that $S - A_{k-1} + X \in \mathcal{I}_1$. Note that $S - A_{k-1} + X - R^{old}_{2k-2} - a \subseteq S - A_{k-1} + B_k + F_{2k-1} - a$. Hence, by the diminishing returns (of adding a) we know $\mathrm{rk}_1(S - A_{k-1} + X - R^{old}_{2k-2}) \geq \mathrm{rk}_1(S - A_{k-1} + X - R^{old}_{2k-2} - a) + 1 = |S - A_{k-1} + X - R^{old}_{2k-2}|$, or equivalently that $S - A_{k-1} + X - R^{old}_{2k-2} \in \mathcal{I}_1$. Since the invariant held before, we conclude that $S - A_{k-1} + X \in \mathcal{I}_1$ too, which finishes the proof. ◀

Valid paths. The algorithm keeps track of a valid path starting at each fresh vertex it has processed. That is, after processing layer D_k, all elements in F_k must be the beginning of a valid path, else they were removed. In particular, the algorithm remembers the valid path starting at x as $(x, \texttt{next}[x], \texttt{next}[\texttt{next}[x]], \ldots)$. It is easy to verify that this sequence does indeed satisfy the conditions of *valid paths* by inspecting lines 10 and 18.

We also discuss what happens when the algorithm chooses to add a valid path to the partial augmenting set (i.e. in line 4 or 13). If we are in Line 13, we can directly apply Lemma 33. Say we instead are in Line 4, and some a which was previously fresh has been added to A_k. The RefineBA call can only have increased A_k (that is $A_k \supseteq A^{old}_k + a$), so $S - A_k + B_{k+1} + b \in \mathcal{I}_1$ will holds for $b = \texttt{next}[a]$ and we can apply Lemma 33 here too.

When no path is found. In the case when no valid path to add to the partial augmenting set is found, `RefinePath` must terminate with $|B_1| = |A_1| = \cdots = |B_{\ell+1}|$. This is because the `RefineAB` and `RefineBA` will never select any new elements. That is `RefineBA` will not change A_k (as otherwise we enter the if-statement at line 4), and `RefineAB` will not change B_k (since if $b \in F_{2k-1}$ with $S - A_{k-1} + B_k + b \in \mathcal{I}_1$ existed we would have entered the if-statement at line 13). We also remark that `RefinePath` ends with B_1 being a maximal subset of $D_1 \setminus R_1$, as otherwise some b would have been found in line 12. Hence Lemma 19 implies that $(B_1, A_1, \ldots, B_{\ell+1})$ now forms a *maximal* augmenting set.

Query complexity. The `RefineAB` and `RefineBA` calls will in total use $O(n)$ queries. The independence checks at Lines 7 and 12 happens at most once for each element, and thus use $O(n)$ queries in total. Lines 10 and 18 can be implemented using the binary-search-exchange-discovery Lemma 11. Hence Line 10 will use, in total, $O(n \log r)$ queries and Line 18 will use, in total, $O(n \log r)$ queries (since each $a \in Q$ will be discovered at most once). So we conclude that Algorithm 5 uses $O(n \log r)$ independence queries.

3.3 Hybrid Algorithm

Now we are finally ready to present the full algorithm of a phase, which is parameterized by a variable p. The following algorithm is similar to that of [3, Algorithm 12] but uses our improved `Refine` method and finds individual paths using the `RefinePath` method.

Algorithm 6 Phase ℓ.

```
1: Calculate the distance layers by a BFS.
2: Run Refine (Algorithm 4) until $|B_1| - |B_{\ell+1}| \le p$, but at least once.
3: Run RefinePath (Algorithm 5) until $(B_1, A_1, \ldots B_{\ell+1})$ is maximal. Augment along it.
```

▶ **Lemma 35.** *Except for line 1, Algorithm 6 uses $O(nr/p + np \log r)$ queries.*[14]

Proof. Lemma 28 tells us that `Refine` changes types of at least p elements in even layers (i.e. elements in S) every time it is run, except maybe the last time. Thus we only run `Refine` $O(|S|/p + 1)$ times. Each call takes $O(n)$ queries (Lemma 29), for a total of $O(nr/p)$ queries in line 2 of the algorithm.

Now we argue that B_1 can never become larger than what it was just after line 2 was run. This is because `Refine` will run at least once, and ends with a `RefineABA`(0) call which in turn ends with a `RefineAB(0)` call – which extends B_1 to be a maximal set in $D_1 \setminus R_1$ for which $S + B_1 \subseteq \mathcal{I}_1$ holds.[15]

Lemma 30 tells us that each (except the last) time `RefinePath` is run, $B_{\ell+1}$ increases by 1. This can happen at most p times, so line 3 uses a total of $O(np \log r)$ queries. ◀

Now it is easy to prove Theorem 1, which we restate below.

[14] Compare this to $O(n^2/p + np\ell \log r)$ in [3]. The improvement from n^2/p to nr/p comes from the use of the new three-layer `RefineABA` method, and the (independent) improvement from $np\ell \log r$ to $np \log r$ comes from the use of the new `RefinePath` method.

[15] Indeed, since $\mathcal{M}_1$ is a matroid, all such maximal sets have the same size, so we can never obtain something larger later.

▶ **Theorem 1** (Approximation algorithm). *There is a deterministic algorithm which given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ on the same ground set V, finds a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ with $|S| \geq (1-\varepsilon)r$, using $O\left(\frac{n\sqrt{r \log r}}{\varepsilon}\right)$ independence queries.*

Proof. Pick $p = \sqrt{r/\log r}$.[16] Then each phase will use $O(n\sqrt{r \log r})$ independence queries (by Lemma 35), plus a total of $O(\frac{1}{\varepsilon} n \log r)$ to run the BFS's across all phases (see [3] for details on the BFS implementation). Since we need only run $O(\frac{1}{\varepsilon})$ phases (by Lemma 10 and Theorem 16), in total the algorithm will use $O(\frac{1}{\varepsilon} n\sqrt{r \log r})$ queries. ◀

4 Exact Matroid Intersection

In this section, we prove Theorem 2 (restated below) by showing how our improved approximation algorithm leads to an improved exact algorithm when combined with the algorithms of [2].

▶ **Theorem 2** (Exact algorithm). *There is a randomized algorithm which given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ on the same ground set V, finds a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ of maximum cardinality r, and w.h.p.*[17] *uses $O(nr^{3/4} \log n)$ independence queries. There is also a deterministic exact algorithm using $O(nr^{5/6} \log n)$ queries.*

Approximation algorithms are great at finding the *many, very short* augmenting paths efficiently. Blikstad-v.d.Brand-Mukhopadhyay-Nanongkai [2, Algorithm 2] very recently showed how to efficiently find the remaining *few, very long* augmenting paths, with a randomized algorithm using $\tilde{O}(n\sqrt{r})$ queries per augmentation (or, with a slightly less efficient deterministic algorithm using $\tilde{O}(nr^{2/3})$ queries). In the randomized $\tilde{O}(n^{6/5}r^{3/5})$-query exact algorithm of [2, Algorithm 3], the current bottleneck is the approximation algorithm used. Replacing the use of the $\tilde{O}(n^{1.5}/\varepsilon^{1.5})$-query approximation algorithm from [3] with our improved version we obtain the more efficient randomized[18] $\tilde{O}(nr^{3/4})$-query Algorithm 7.

Algorithm 7 Exact Matroid Intersection. (Modified version of [2, Algorithm 3])

```
1: Run the approximation algorithm (Theorem 1) with $\varepsilon = r^{-1/4}$ to obtain a common
   independent set $S$ of size at least $(1-\varepsilon)r = r - r^{3/4}$.
2: Starting with $S$, run Cunningham's algorithm (as implemented by [3]), until the distance
   between $s$ and $t$ becomes larger than $r^{3/4}$.
3: Keep finding augmenting paths – one at a time – to augment along, using the randomized
   $O(n\sqrt{r} \log n)$-query algorithm of [2, Algorithm 2]. When no $(s,t)$-path can be found in
   the exchange graph, $S$ is a largest common independent set.
```

Query complexity. We analyse the individual lines of Algorithm 7.

Line 1. We see that the approximation algorithm uses $O(nr^{3/4} \log n)$ queries in line 1.

Line 2. One need to (i) compute distances up to $d = r^{3/4}$, and (ii) perform at most $O(r^{3/4})$ augmentations. [2, 3, 11] show how to do (i) in $O(nd \log n) = O(nr^{3/4} \log n)$ queries in total over all phases of Cunningham's algorithm, and how to do (ii) using $O(n \log n)$ queries per augmentation (for a total of $O(nr^{3/4} \log n)$ queries).

[16] Compare this to $p = \sqrt{n\varepsilon/\log r}$ in [3].
[17] *w.h.p. = with high probability* meaning with probability $1 - n^{-c}$ for some arbitrarily large constant c.
[18] The deterministic algorithm of Theorem 2 is obtained in the same fashion but by using the deterministic version of the augmenting path finding algorithm [2, Algorithm 2].

Line 3. By Lemma 10, line 3 runs $O(r^{1/4})$ times – each using $O(n\sqrt{r}\log n)$ queries – for a total of $O(nr^{3/4}\log n)$ queries.

▶ Remark 36. In Algorithm 7, the bottleneck between line 1-2 and line 2-3 now matches (which was not the case in [2]). This means that if one wants to improve the algorithm by replacing the subroutines in line 1 and 3, one need to **both** improve the approximation algorithm (line 1) and the method to find a single augmenting-path (line 3).

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, Jan van den Brand, Sagnik Mukhopadhyay, and Danupon Nanongkai. Breaking the quadratic barrier for matroid intersection. In *STOC*. ACM, 2021.

3 Deeparnab Chakrabarty, Yin Tat Lee, Aaron Sidford, Sahil Singla, and Sam Chiu-wai Wong. Faster matroid intersection. In *FOCS*, pages 1146–1168. IEEE Computer Society, 2019.

4 Chandra Chekuri and Kent Quanrud. A fast approximation for maximum weight matroid intersection. In *SODA*, pages 445–457. SIAM, 2016.

5 William H. Cunningham. Improved bounds for matroid partition and intersection algorithms. *SIAM J. Comput.*, 15(4):948–957, 1986.

6 Jack Edmonds. Submodular functions, matroids, and certain polyhedra. In *Combinatorial structures and their applications*, pages 69–87. Gordon and Breach, 1970.

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

8 Jack Edmonds, GB Dantzig, AF Veinott, and M Jünger. Matroid partition. *50 Years of Integer Programming 1958–2008*, page 199, 1968.

9 Eugene L. Lawler. Matroid intersection algorithms. *Math. Program.*, 9(1):31–56, 1975.

10 Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. In *FOCS*, pages 1049–1065. IEEE Computer Society, 2015.

11 Huy L. Nguyen. A note on cunningham's algorithm for matroid intersection. *CoRR*, abs/1904.04129, 2019. arXiv:1904.04129.