Sublinear-Round Parallel Matroid Intersection

Joakim Blikstad ✉
KTH Royal Institute of Technology, Sweden

Abstract

Despite a lot of recent progress in obtaining faster sequential matroid intersection algorithms, the fastest *parallel* poly(n)-query algorithm was still the straightforward $O(n)$-round parallel implementation of Edmonds' augmenting paths algorithm from the 1960s.

Very recently, Chakrabarty-Chen-Khanna [FOCS'21] showed the lower bound that any, possibly randomized, parallel matroid intersection algorithm making poly(n) rank-queries requires $\tilde{\Omega}(n^{1/3})$ rounds of adaptivity. They ask, as an open question, if the lower bound can be improved to $\tilde{\Omega}(n)$, or if there can be sublinear-round, poly(n)-query algorithms for matroid intersection.

We resolve this open problem by presenting the first sublinear-round parallel matroid intersection algorithms. Perhaps surprisingly, we do not only break the $\tilde{O}(n)$-barrier in the rank-oracle model, but also in the weaker independence-oracle model. Our rank-query algorithm guarantees $O(n^{3/4})$ rounds of adaptivity, while the independence-query algorithm uses $O(n^{7/8})$ rounds of adaptivity, both making a total of poly(n) queries.

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

Keywords and phrases Matroid Intersection, Combinatorial Optimization, Parallel Algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.25

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. The author is also supported by the Swedish Research Council (Reg. No. 2019-05622).

Acknowledgements I thank Danupon Nanongkai and Sagnik Mukhopadhyay for insightful discussions and their valuable comments throughout the development of this work. Part of this work was done while the author visited BARC and the University of Copenhagen.

1 Introduction

Matroid intersection. Given two matroids[1] $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ over the same n-element ground set V (but with different notions of independence $\mathcal{I}_1, \mathcal{I}_2$), the *matroid intersection problem* is to find the largest common independent set $S^* \subseteq \mathcal{I}_1 \cap \mathcal{I}_2$. This is a fundamental discrete optimization problem that has been studied for over half a century. Matroid intersection can be used to model many important combinatorial optimization problems, such as bipartite matching, finding arborescences, spanning tree packing, etc. As such, matroid intersection is a natural avenue to study all these problems simultaneously.

Oracle access. There are two standard ways to access the matroids – *independence oracles* and *rank oracles* – and we study both in this work. In an *independence-query* we may ask if $S \subseteq V$ is independent in one of the matroids, i.e. a query of the form "Is $S \in \mathcal{I}_1$?" or "Is $S \in \mathcal{I}_2$?" In a *rank-query* we instead ask for the *rank* of $S \subseteq V$ in one of the matroids. The

[1] Matroids are a well-studied combinatorial structure which can be though of as a generalization of the notion of linear independence in vector spaces. For a formally definition, see Definition 4.

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 25; pp. 25:1–25:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

rank $\mathrm{rk}_1(S)$ with respect to the matroid $\mathcal{M}_1$ (similarly rk_2 for $\mathcal{M}_2$) is the size of the largest (or, equivalently, any maximal) independent set, w.r.t. $\mathcal{I}_1$, contained in S. Note that the rank oracle is strictly more powerful than the independence oracle, since $S \in \mathcal{I}_1$ if and only if $\mathrm{rk}_1(S) = |S|$.

Parallel matroid intersection. A parallel matroid intersection algorithm accesses the oracle in rounds. In each round, a number of queries – that may only depend on the answers to queries made in previous rounds – can be issued in parallel. There is certainly a trade-off between (1) *adaptivity*, usually measured by the number of rounds, and (2) the total number of queries. When constructing parallel algorithms the goal is often to have as few rounds of adaptivity as possible while making only polynomially many queries in total.

Previous work. Edmonds [10] showed the first polynomial algorithm for matroid intersection in the 1960s, using $O(n^3)$ independence-queries, and there has been a long line of research since then e.g. [1, 2, 3, 5, 6, 8, 9, 10, 18, 19, 21]. Many of these are based on Edmonds' framework of finding *augmenting paths* in the *exchange graph*. In the sequential setting, only recently was the quadratic $O(n^2)$-query-barrier broken, first for rank-queries by Chakrabarty-Lee-Sidford-Singla-Wong in FOCS 2019 [5] and subsequently also for independence-queries by Blikstad-v.d.Brand-Mukhopadhyay-Nanongkai in STOC 2021 [3]. The current state-of-the-art in the sequential setting are the[2] $\tilde{O}(n\sqrt{n})$ rank-query algorithm by [5] and the $\tilde{O}(n^{7/4})$ independence-query algorithm by [2].

When it comes to the *parallel* setting, there is a straightforward $O(n)$-round, $\mathrm{poly}(n)$ independence-query implementation of Edmonds' algorithm: find the (up to $O(n)$ many) augmenting paths one-by-one. Each augmenting path can be found in a single round by querying all the potential edges in the exchange graph. In some special cases of matroid intersection we can do much better: a sequence of work has shown that both bipartite matching [11, 16, 20] and subsequently linear matroid intersection [13, 20] are in RNC[3] and quasi-NC.[4]

Another line of relevant work is showing that, in the parallel setting, the *search*-problem (finding a largest common independent set S) and the *decision*-problem (just finding the size of the answer) are "equivalent" (with only $O(\mathrm{polylog}(n))$ overhead). This is not at all obvious in the parallel setting, however, a recent work from SODA 2022 by Ghosh-Gurjar-Ray [12] shows that this is indeed the case for *weighted* matroid intersection, with rank-oracle access.

In FOCS 1985, Karp-Upfal-Wigderson [15] showed that any *independence-query* algorithm, possibly randomized, that finds a maximum independent set (basis) in a *single* matroid must use $\tilde{\Omega}(n^{1/3})$ rounds of adaptivity if it makes $\mathrm{poly}(n)$ queries. They also show algorithms to find a basis of a (single) matroid in $O(\sqrt{n})$ rounds of independence-queries or a single round of the more powerful rank-queries. Arguably, this polynomial gap between the independence-query ($\tilde{\Omega}(n^{1/3})$ rounds) and rank-query ($O(1)$ rounds) for the seemingly easy problem to find a basis of a matroid illustrates that the independence-query is much weaker than the rank-query when used in parallel algorithms.

Nevertheless, a recent result from FOCS 2021 by Chakrabarty-Chen-Khanna [4] shows that even *rank-query* algorithms require a polynomial number of rounds to solve matroid intersection. In particular, they show a lower bound of $\tilde{\Omega}(n^{1/3})$ rounds of adaptivity for any, possibly randomized, $\mathrm{poly}(n)$ rank-query matroid intersection algorithm.

---

[2] We use the usual convention of hiding $\mathrm{polylog}(n)$-factors with $\tilde{O}$ and $\tilde{\Omega}$ throughout the paper.
[3] Randomized $\mathrm{polylog}(n)$ rounds of adaptivity with $\mathrm{poly}(n)$ total work.
[4] Deterministic $\mathrm{polylog}(n)$ rounds of adaptivity with $n^{O(\log n)}$ total work.

Despite efficient algorithms for some special cases of matroid intersection, the trivial $O(n)$-round algorithm has remained unbeaten in the general case. The major open question (asked, for example, by [4]) is then whether it is possible to beat the $O(n)$-round barrier, or if matroid intersection is inherently very sequential and requires $\tilde{\Omega}(n)$ rounds of adaptivity.

Our results. We answer the above question by showing the first sublinear-round parallel matroid intersection algorithms, both in the rank-oracle and independence-oracle models. In particular, we obtain the following theorem.

▶ **Theorem 1** (Sublinear-round Matroid Intersection). *There is a deterministic parallel 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 largest common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ using either*

- *$O(n^{3/4})$ rounds of polynomially many rank-queries, or*
- *$O(n^{7/8})$ rounds of polynomially many independence-queries.*

Our results, together with the lower bounds of [4, 15], imply that the true adaptivity of matroid intersection is somewhere between $n^{1/3}$ and $n^{3/4}$ (or $n^{7/8}$ for independence queries).

▶ Remark 2. Although we focus on the query-complexity in this paper, we note that the rounds and work in our algorithms are dominated, up to log-factors, by the oracle queries.

1.1 Technical Overview

The exchange graph and augmenting paths. Like many matroid intersection algorithms, we work in Edmonds' framework of finding *augmenting paths* in the *exchange graph*. The *exchange graph* $G(S)$ with respect to a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ is a directed bipartite graph, where finding a shortest (s, t)-path – called an *augmenting path* – means that we can increase the size of S by one. In a single round of $O(n^2)$ independence (or rank) queries, we can learn the entire exchange graph, and can thus find an augmenting path if one exists. This immediately gives a straightforward $O(n)$-round algorithm: *find the (up to $O(n)$ many) augmenting paths one-by-one.*

The exchange graph depends on the current common independent set S, and changes after each augmentation. In fact, if we have two disjoint augmenting paths p_1 and p_2 in $G(S)$, it is **not** necessarily the case that we can augment along both of these: augmenting along p_1 might destroy the path p_2 even if they were disjoint.[5] This forms the main difficulty in trying to beat the $O(n)$-round barrier, and illustrates the need in finding several "compatible" augmenting paths which can all be augmented along simultaneously.

Blocking flow. Cunningham [6] was the first to introduce *blocking flow* algorithms to matroid intersection, similar to Hopcroft-Karp's [14] bipartite matching or Dinitz's [7] max-flow algorithms. The idea is to run in phases, where after each phase the length of a shortest augmenting path in the exchange graph has increased. This is done by finding a maximal collection of compatible shortest augmenting paths. Both of the current state-of-the-art sequential $O(n\sqrt{n})$-rank-query [5] and $O(n^{7/4})$-independence-query [2] algorithms are based on versions of these blocking flow ideas. The $O(n\sqrt{n})$-rank-query algorithm still finds the augmenting paths in a sequential way, so it does unfortunately not seem to parallelize well.

[5] This is unlike the case of augmenting path algorithms for bipartite matching or maximum flow, where one can indeed augment along disjoint paths simultaneously.

The $O(n^{7/4})$-independence-query algorithm, on the other hand, is based on a recent notion of *augmenting sets* introduced by Chakrabarty-Lee-Sidford-Singla-Wong [5]. This notion of *augmenting sets* precisely captures what a collection of "compatible" shortest augmenting paths looks like. The authors of [5] also present an algorithm to find such augmenting sets, using independence-queries.

Our contribution is to show that a modified version of the augmenting sets algorithm of [5, Section 6] (which was later improved by [2]) can be implemented in parallel when combined with the parallel matroid-basis finding algorithms of Karp-Upfal-Wigderson [15]. Previous to this work, augmenting sets algorithms have before only been used in the sequential setting, and only in the independence-oracle model. Nevertheless, augmenting sets are what allows us to break the $O(n)$-round barrier also with rank-queries.

2 Preliminaries

We use the standard definitions of *matroid* $\mathcal{M} = (V, \mathcal{I})$; *rank* $\mathrm{rk}(X)$ 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 [5], which we also define in later this section.

▶ **Definition 3** (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 $A + B := A \cup B$, and $A - B := A \setminus B$.

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 $S' + x \in \mathcal{I}$.

▶ **Definition 5** (Matroid rank)**.** The *rank* of $A \subseteq V$, denoted by $\mathrm{rk}(A)$, 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. $\mathrm{rk}(A + x) - \mathrm{rk}(A) \geq \mathrm{rk}(B + x) - \mathrm{rk}(B)$ whenever $A \subseteq B \subseteq V$ and $x \in V \setminus B$. Note that $\mathrm{rk}(A) = |A|$ if and only if $A \subseteq \mathcal{I}$.

▶ **Definition 6** (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. We use rk_1 and rk_2 to denote the rank functions of the corresponding matroids, and $n = |V|$ to be the size of the ground set.

The Exchange Graph

▶ **Definition 7** (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 $b \in V \setminus S$ when $S + b \in \mathcal{I}_1$.
2. (b, t) for $b \in V \setminus S$ when $S + b \in \mathcal{I}_2$.
3. (a, b) for $b \in V \setminus S, a \in S$ when $S - a + b \in \mathcal{I}_1$.
4. (b, a) for $b \in V \setminus S, a \in S$ when $S - a + b \in \mathcal{I}_2$.

We will denote the set of elements at distance k from s by the distance-layer D_k. Note that $D_k \subseteq V \setminus S$ when k is odd and $D_k \subseteq S$ when k is even.

▶ **Definition 8** (Shortest augmenting path). A shortest (s,t)-path $p = (s, b_1, a_2, b_3, a_4, \ldots, a_{\ell-1}, b_\ell, t)$ (with $b_i \in V \setminus S$ and $a_i \in S$) in $G(S)$ is called a *shortest augmenting path*. We can *augment* S along the path p to obtain $S' = S + b_1 - a_2 + b_3 - a_4 \ldots + b_\ell$, which is well-known to also be a common independent set (with $|S'| = |S| + 1$). Conversely, there must exist a shortest augmenting path whenever S is not a largest common independent set.

Augmenting Sets

Augmenting Sets is a notion capturing a "blocking flow" in the exchange graph, and was introduced by [5], and also subsequently used in the algorithms of [2,3]. In order to efficiently find "good" augmenting sets, the algorithm works with a relaxed form of them instead, called *partial* augmenting sets. The following definitions and key properties of (partial) augmenting sets are copied from [5] where one can find the corresponding proofs.

▶ **Definition 9** (Augmenting Sets, from [5, 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 $\ell + 1$ (ℓ must be odd) and distance layers $D_1, D_2, \ldots, D_\ell$. A collection of sets $\Pi_\ell := (B_1, A_2, B_3, A_4, \ldots, A_{\ell-1}, B_\ell)$[6] form an *augmenting set* in $G(S)$ if the following conditions are satisfied:

(a) $A_k \subseteq D_k$ for even k, and $B_k \subseteq D_k$ for odd k.
(b) $|B_1| = |A_2| = |B_3| = \cdots = |B_\ell|$
(c) $S + B_1 \in \mathcal{I}_1$
(d) $S + B_\ell \in \mathcal{I}_2$
(e) For all even $1 \leq k \leq \ell$, we have $S - A_k + B_{k+1} \in \mathcal{I}_1$
(f) For all odd $1 \leq k \leq \ell$, we have $S - A_{k+1} + B_k \in \mathcal{I}_2$

▶ **Definition 10** (Partial Augmenting Sets, from [5, Definition 37]). We say that $\Phi_\ell := (B_1, A_2, B_3, A_4, \ldots, A_{\ell-1}, B_\ell)$ forms a *partial augmenting set* if it satisfies the conditions (a), (c),[7] and (e) of an *augmenting set*, plus the following two relaxed conditions :

(b) $|B_1| \geq |A_2| \geq |B_3| \geq \cdots \geq |B_\ell|$.
(f) For all odd $1 \leq k \leq \ell$, we have $\mathrm{rk}_2(S - A_{k+1} + B_k) = \mathrm{rk}_2(S)$.

▶ **Theorem 11** (from [5, Theorem 25]). *Let $\Pi_\ell := (B_1, A_2, B_3, A_4, \cdots, A_{\ell-1}, B_\ell)$ be the an augmenting set in the exchange graph $G(S)$. Then the set $S' := S \oplus \Pi_\ell := S + B_1 - A_2 + B_3 - \cdots - A_{\ell-1} + B_\ell$ 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. Together with a lemma from [6] (Lemma 14), we can see, on a high-level, how to obtain $(1-\varepsilon)$-approximation algorithms: *find "blocking flows" (i.e. maximal augmenting sets) until the (s,t)-distance is $\Omega(1/\varepsilon)$.*

▶ **Definition 12** (Maximal Augmenting Sets, from [5, Definition 35]). Let $\Pi_\ell = (B_1, A_2, B_3, \cdots, A_{\ell-1}, B_\ell)$ and $\tilde{\Pi}_\ell = (\tilde{B}_1, \tilde{A}_2, \tilde{B}_3, \cdots, \tilde{A}_{\ell-1}, \tilde{B}_\ell)$ 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.

[6] Our indexing of the sets differ a bit from [2,5].
[7] Note that we intentionally skip item (d), unlike [5] which includes it in the definition, however they do not always maintain this property in their algorithms.
[8] Note that $|S'| = |S| + |B_1|$. In particular, an augmenting set with $|B_1| = 1$ is exactly an augmenting path. [5] shows that augmenting sets correspond exactly to a sequence of consecutive shortest augmenting paths.

▶ **Lemma 13** (from [5, Theorem 36]). *An augmenting set Π_ℓ is maximal if and only if there is no augmenting path of length at most $\ell+1$ in $G(S \oplus \Pi_\ell)$.*

▶ **Lemma 14** (Cunningham [6]). *If the length of the shortest (s,t)-path in $G(S)$ is at least $2\ell+1$, then $|S| \geq (1-O(1/\ell))r$, where r is the size of the largest common independent set.*

3 Warm-up: Finding a *Maximal* Common Independent Set

Consider first the easier problem of finding a *maximal* (instead of *maximum*) common independent set: that is we want to find a set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ such that there is no $x \in V \setminus S$ for which $S + x \in \mathcal{I}_1 \cap \mathcal{I}_2$. It is well-known that a maximal common independent set is also a $\frac{1}{2}$-approximation for the matroid intersection problem, and indeed our algorithm for a general $(1-\varepsilon)$-approximation (Section 4) will use similar ideas as our algorithm to find a *maximal* common independent set in this section.

In the sequential setting there is a very easy $O(n)$-query greedy algorithm: *Start with $S = \emptyset$ and go through all elements $x \in V$ and add them to S if $S + x$ is independent in both matroids.* However, this greedy algorithm is inherently very sequential and does not seem to adapt well to the parallel setting. Instead, we must somehow try to find several xs "in parallel" which we can all add to S simultaneously without breaking independence.

3.1 One Matroid

Let us start even simpler, and consider how to find a *maximal* independent set[9] S in a single matroid $\mathcal{M} = (V, \mathcal{I})$. It turns out that in our final matroid intersection algorithm we will many times, as a subroutinue, need to do exactly this.

Karp-Upfal-Wigderson [15] provides some simple parallel algorithms (both for rank- and independence-oracle access), whose results we present in Lemma 15. We briefly sketch their algorithms below, more details and full proofs can be found in [15, 17].

Rank Oracle. The rank-query algorithm only needs a single round. Let $\{v_1, v_2, \ldots, v_n\} = V$ be the elements of the ground set, and let $V_i = \{v_1, v_2, \ldots, v_{i-1}, v_i\}$ (so that $V_0 = \emptyset$ and $V_n = V$). Now query $\mathrm{rk}(V_i)$ for all i, and return $S = \{v_i : \mathrm{rk}(V_i) > \mathrm{rk}(V_{i-1})\}$. Intuitively, we can imagine that we go through all elements v_i one-by-one and add them to S if and only if the rank goes up.

Independence Oracle. The independence-query algorithm will need $O(\sqrt{n})$ rounds of $O(n)$ queries per round. Partition the elements of V into $\sqrt{n}$ different groups of (almost) equal size $F_1, F_2, \ldots F_{\sqrt{n}}$. If any group is independent (say F_i), then we select it, and consider the contracted matroid $\mathcal{M}/F_i$. Note that this can only happen $\sqrt{n}$ times. On the other hand, if all F_i are dependent, then we will find one element per group (that is $\sqrt{n}$ in total) which we can safely discard: If $\{v_1, v_2, \ldots, v_k\} = F_i$ are the elements of F_i, we query all prefixes, i.e. "Is $\{v_1, v_2, \ldots, v_j\} \in \mathcal{I}$?" for all j, and discard the first element v_j for which the answer is "No".

[9] Such a set S is usually called a *basis* of the matroid, and due to the exchange-property all the maximal independent sets must have the same size.

▶ **Lemma 15** (Parallel basis algorithm, [15]). *There is a deterministic parallel algorithm which given a matroids $\mathcal{M} = (V, \mathcal{I})$ finds a **maximal** independent set $S \in \mathcal{I}$ using either*

- *$O(1)$ round of $O(n)$ many rank-queries, or*
- *$O(\sqrt{n})$ rounds of $O(n)$ many independence-queries.*[10]

▶ Remark 16. Note that if we have a set $X \in \mathcal{I}$ and $Y \subseteq V \setminus X$, we can with the same algorithms as above find a maximal $Y' \subseteq Y$ such that $X + Y' \in \mathcal{I}$, even though the above algorithms are only stated as if $X = \emptyset$ and $Y = V$. This is since we can consider the contracted and restricted matroid $\mathcal{M}' = (\mathcal{M} \,/\, X) \setminus (V \setminus Y)$; and an independence/rank-query on $\mathcal{M}'$ can be simulated with the corresponding query on $\mathcal{M}$.

3.2 Two Matroids

Now we return to our problem of finding a maximal common independent set S of two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$. Suppose we already have some common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$. We will try to add more elements to S until it becomes maximal.

Firstly, let us concentrate on the first matroid and pick a maximal set $B \subseteq V$ such that $S + B \in \mathcal{I}_1$ using Lemma 15. However, $S + B$ is not necessarily independent in the second matroid, so we would need to fix this: let $B' \subseteq B$ be a maximal subset such that $S + B' \in \mathcal{I}_2$, which we again can find using Lemma 15. Now we know $S + B' \in \mathcal{I}_1 \cap \mathcal{I}_2$ is a common independent set, so we set $S \leftarrow S + B'$, and we have made some progress (unless $B' = \emptyset$ of course).

At this point we can make a crucial observation: *we can safely discard the elements $x \in B \setminus B'$, since now $S + x \notin \mathcal{I}_2$.* Hence, for each element in B we have either (i) added it to our common independent set or (ii) discarded it. As long as $|B|$ is relatively large (say $\approx \sqrt{n}$), we have made significant progress.

On the other hand, if $|B|$ is small, we may resort to a different strategy. By the exchange property of matroids, we know that any $A \subseteq V$ such that $S + A \in \mathcal{I}_1$ has size $|A| \leq |B|$. So we can add at most $|B|$ more elements to our common independent set S before it becomes maximal. We can thus simply find these remaining (up to $|B|$ many) elements one-by-one, using one round each.

We present this two-stage strategy below in Algorithm 1, which is parametrized by the cut-off threshold p for when to consider $|B|$ small. The optimal choice of p differs depending on the oracle access (independence or rank) we have.

Adaptivity. The first stage of Algorithm 1 runs in $O(n/p \cdot \mathcal{T}_{\text{basis}})$ rounds if $\mathcal{T}_{\text{basis}}$ is the number of rounds needed to find a maximal independent set for a single matroid (Lemma 15 gives $\mathcal{T}^{\text{rank}}_{\text{basis}} = O(1)$ and $\mathcal{T}^{\text{indep}}_{\text{basis}} = O(\sqrt{n})$). This is since the size of F will decrease by $|B| \geq p$ each time the while-loop is run, which can happen at most n/p times. The second stage of Algorithm 1 runs in $O(p)$ rounds, both for independence and rank-oracle. Picking p optimally gives: $O(\sqrt{n})$ rounds of rank-queries (with $p = \sqrt{n}$); or $O(n^{3/4})$ rounds of independence-queries (with $p = n^{3/4}$). This proves Theorem 17, stated below.

▶ **Theorem 17.** *There is a deterministic parallel 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 **maximal** common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ using either*

- *$O(\sqrt{n})$ rounds of polynomially many rank-queries, or*
- *$O(n^{3/4})$ rounds of polynomially many independence-queries.*

[10] KUW [15] also provides a lower bound of $\tilde{\Omega}(n^{1/3})$ rounds for any independence-query algorithm which uses only polynomial number of queries per round, even if randomization is allowed. It remains an open problem to close this gap between $\tilde{\Omega}(n^{1/3})$ and $O(\sqrt{n})$.

Algorithm 1 Maximal Common Independent Set.

Let $S = \emptyset$ and $F = V$.
while true **do** ▷ Stage 1
  Find a maximal $B \subseteq F$ such that $S + B \in \mathcal{I}_1$ (using Lemma 15).
  if $|B| \leq p$ **then break**
  Find a maximal $B' \subseteq B$ such that $S + B' \in \mathcal{I}_2$ (using Lemma 15).
  Update $S \leftarrow S + B'$ and $F \leftarrow F - B$.
while true **do** ▷ Stage 2
  Query "Is $S + x \in \mathcal{I}_1$" and "Is $S + x \in \mathcal{I}_2$?" for all $x \in F$ in parallel.
  Pick an arbitrary $x \in F$ such that $S + x \in \mathcal{I}_1$ and $S + x \in \mathcal{I}_2$.
  if no such x exists **then break**
  Update $S \leftarrow S + x$ and $F \leftarrow F - x$.

4 Finding a *Maximum* Common Independent Set

In this section we present our sublinear-round matroid intersection algorithm.

The algorithm consists of two steps: first it finds an $(1-\varepsilon)$-approximation, and then it finds the remaining εn (which is sublinear if $1/\varepsilon$ is polynomially large in n) augmenting paths one-by-one. Each such remaining augmenting path can be found in a single round of n^2 independence (or rank) queries: in parallel query each possible edge of the exchange graph, and then see if there was an augmenting path. Indeed, when we know all the edges of the exchange graph, we do not need any more (rounds of) queries to figure out if there was an path.[11] If we skipped the $(1-\varepsilon)$-approximation step and just found all the augmenting paths one-by-one we obtain the straightforward $O(n)$-round algorithm.

The difficult part of the algorithm is how to find the $(1-\varepsilon)$-approximation in sublinear number of rounds (even when $1/\varepsilon$ is polynomially large). To do this, we would need to find many augmenting paths simultaneously, and indeed this is our strategy. Our main result of this section is this approximation algorithm which we summarize in Theorem 18 below.

▶ **Theorem 18** (Sublinear-round $(1-\varepsilon)$-approximation). *There is a deterministic parallel 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 size $|S| \geq (1-\varepsilon)r$, where r is the size of the largest common independent set, using either*

- *$O(\sqrt{n}/\varepsilon)$ rounds of polynomially many rank-queries, or*
- *$O(n^{3/4}/\varepsilon)$ rounds of polynomially many independence-queries.*

Exact algorithm. By an appropriate choice of ε ($\varepsilon = n^{-1/4}$ for rank-oracle and $\varepsilon = n^{-1/8}$ for independence-oracle), together with our discussion above, the main result (Theorem 1, restated below) of the paper – the sublinear-round exact algorithm – follows immediately from Theorem 18. The remainder of this paper will go towards proving Theorem 18, i.e. the $(1-\varepsilon)$-approximation algorithm.

▶ **Theorem 1** (Sublinear-round Matroid Intersection). *There is a deterministic parallel 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 largest common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ using either*

- *$O(n^{3/4})$ rounds of polynomially many rank-queries, or*
- *$O(n^{7/8})$ rounds of polynomially many independence-queries.*

[11] Note that if we also care about the number of rounds of work of the algorithm (and not just the rounds of *queries*), we can find the augmenting path in the exchange graph in just poly(n) rounds, as s,t-reachability is well-known to be in NC.

4.1 Blocking Flow

The approximation algorithm maintains a common independent set S and runs in $O(1/\varepsilon)$ phases, where in the i'th phase it eliminates all augmenting paths of length $2i$ by finding a blocking flow, similar to the Hopcroft-Karp's [14] bipartite matching algorithm and Dinitz's [7] max-flow algorithm. By *blocking flow* we mean a set of compatible shortest augmenting paths after which augmenting along them the (s,t)-distance in the exchange graph has increased. At the end, by Lemma 14, we will have found a common independent set S which is a $(1-\varepsilon)$-approximation, since the shortest augmenting path will have length $O(1/\varepsilon)$.

This idea of applying blocking flow algorithms to matroid intersection originates from the algorithm of Cunningham [6], but has since been improved by Chakrabarty-Lee-Sidford-Singla-Wong [5] and subsequently Blikstad [2], and it is the framework of these two later algorithms which we will follow.

▶ Remark 19. In the first phase we will eliminate all augmenting paths of length 2. This corresponds exactly to finding a *maximal* common independent set, like we did in Section 3. In general, we will show that we can implement any phase in the same round-complexity as the first phase, using similar ideas.

Beginning of a phase. In each phase, we consider a layered graph, where we let the *distance-layer* D_i denotes all the elements of distance i from the source node s in the exchange graph $G(S)$. At the beginning of a phase, the algorithm will use a single round (of $O(n^2)$ queries) to find these distance layers: simply query all potential edges of the exchange graph.

Unfortunately, knowing all the edges in the exchange graph is not sufficient to find a blocking flow, since a set of disjoint augmenting paths might not be compatible with each other. The exchange graph $G(S)$ does not capture the full structure of the matroid intersection problem, and this is where the difficulty in obtaining sublinear-round matroid intersection algorithms comes from. There is a need to be able to find many compatible augmenting paths "in parallel".

Augmenting sets. The notion of a collection of *compatible*[12] augmenting paths is captured by *augmenting sets*, as defined in Definition 9. So our goal in a phase is to find a *maximal* augmenting set (see Definition 12), which is what we formally mean by "blocking flow". After augmenting along a maximal augmenting set, Lemma 13 implies that the (s,t)-distance has increased, and we can move on to the next phase.

Our algorithm will follow the framework of [5] and [2], which are the state-of-the-art sequential independence-query approximation algorithms. The overall idea can be seen as a generalization of the warm-up *maximal* common independent set algorithm from Section 3. Instead of working with just a single distance layer in the exchange graph we now have up to $O(1/\varepsilon)$ many layers. Fortunately, layers far apart from each other can be handled relatively well in parallel, and we will see that the final adaptivity of our algorithm to find a blocking flow in a phase will not depend on the number of layers.

We start by, on a high level, summarizing how the algorithm of [2, 5] implements a phase in two stages. Our main result is how we can implement this algorithm efficiently in a parallel.

[12] That is they can all be augmented along simultaneously without breaking independence in either matroid.

1. The first stage keeps track of a *partial* augmenting set (Definition 10) 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 maximal augmenting set, the progress we make in the first stage slows down. Then we fall back to the second stage in which we find the relatively few remaining augmenting paths individually one at a time.

4.2 First Stage: Refining

The basic refining ideas and procedures in this section are the same as in [2,5]; our contribution is to show how they can be implemented in a parallel fashion.

Say we are in the phase where the (s,t)-distance is $\ell+1$, that is we have ℓ layers in our exchange graph. The algorithm keeps track of a partial augmenting set $\Phi_\ell = (B_1, A_2, B_3, \ldots, A_{\ell-1}, B_\ell)$ (see Definition 10), which it makes local improvements to, called *refining*. Essentially Φ_ℓ looks like a stair-case: B_{k+1} is a set which can be "matched" to some subset of the previous layer A_k; and similarly A_{i+1} can be "matched" to a subset of B_i. As long as Φ_ℓ is "far" from being a maximal augmenting set, the refinement procedures make significant progress. When Φ_ℓ becomes "close" to being a maximal augmenting set we move on to the second stage.

We maintain three types of elements in each layer D_k in the exchange graph.
Selected. Sets A_k and B_k form the partial augmenting set Φ_ℓ.
Removed. Sets R_k contain the discarded elements which we have deemed useless.
Fresh. Sets F_k contain the elements which are neither selected nor removed.

All elements are initially fresh, and for convenience we also define "imaginary" empty boundary layers $D_0 = D_{\ell+1} = \emptyset$, with corresponding sets $A_0, R_0, F_0, A_{\ell+1}, R_{\ell+1}, F_{\ell+1}$. Note that (A_k, R_k, F_k) forms a partition of $D_k \subseteq S$ when k is even, and that (B_k, R_k, F_k) partitions $D_k \subseteq V \setminus S$ when k is odd.

The idea of the refinement procedures is to make some local improvements to adjacent distance-layers. While doing this, we make sure that elements only change their types from *fresh* $\to$ *selected* $\to$ *removed*, but never in the other direction. In order to formalize that the removed elements are actually useless, we maintain the following *phase invariants*.

▶ **Definition 20** (Phase Invariants, from [5, Section 6.3.2])**.** The *phase invariants* are:
(a-b) $\Phi_\ell = (B_1, A_2, B_3, \ldots, A_{\ell-1}, B_\ell)$ forms a partial augmenting set.[13]
(c) For all even k, $\mathrm{rk}_1(W - R_k) = \mathrm{rk}_1(W) - |R_k|$ where $W = S - A_k + (D_{k+1} - R_{k+1})$.[14]
(d) For all odd k, $\mathrm{rk}_2(W + R_k) = \mathrm{rk}_2(W)$ where $W = S - (D_{k+1} - R_{k+1}) + B_k$.

▶ Remark 21. Invariant (c) and (d) essentially says that if R_{k+1} is useless, then so is R_k, for both even and odd layers, and thus, by induction, all removed elements are indeed useless. For example, (d) says that any element $x \in R_k$ does not increase the rank, even if we take away all non-useless elements $(D_{k+1} - R_{k+1})$ in the next layer. Hence such an x cannot be "matched" to any non-useless element in the next layer, so it is safe to discard it, since we will never be able to add it to B_k while maintaining that $(\ldots, B_k, A_{k+1}, \ldots)$ form an partial augmenting set.

[13] The naming of this invariant as (a-b) is to be consistent with [5] where this condition is split up into two separate items (a) and (b).
[14] This invariant differs from [5], where it was written in the following equivalent form: For $1 \le k \le \ell/2$, for any $X \subseteq B_{2k+1} + F_{2k+1} = D_{2k+1} - R_{2k+1}$, if $S - (A_{2k} + R_{2k}) + X \in \mathcal{I}_1$ then $S - A_{2k} + X \in \mathcal{I}_1$.

4.2.1 Refining Locally

We now present the basic refinement procedures from [5], which are operations on two neighboring layers. We note that [2] improves upon the algorithm of [5] (in the sequential setting) by considering refinement operations on **three** consecutive layers instead. Unfortunately, the three-layer refinement procedures of [2] does not seem to work efficiently in the parallel setting.

Intuitively, for an even k, `RefineAB`(k) tries to extend B_{k+1} as much as possible while it still can be "matched" from A_k in the previous layer (i.e. while $S - A_k + B_{k+1} \in \mathcal{I}_1$). After this, if $|A_k| > |B_{k+1}|$, we can remove elements from A_k and argue that they are useless (if they were useful, then it should have been possible to "match" them to something more in the next layer, but this is not the case since B_{k+1} could not be extended more). So `RefineAB`(k) extends B_{k+1} and shrinks A_k so that they are the same size. Doing so, $|A_k^{old}| - |B_{k+1}^{old}|$ elements have changed types, and this crucial observation is what allows us to measure progress. For an odd k, `RefineBA`(k) works very similarly, but now between the consecutive layers (B_k, A_{k+1}).

Algorithm 2 `RefineAB`(k) for even k. (called `Refine1` in [5, Algorithm 9])

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

Algorithm 3 `RefineBA`(k) for odd k. (called `Refine2` in [5, Algorithm 10])

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

▶ Remark 22. When we are in the first phase, that is when there is only a single layer between s and t in the graph, running `RefineAB`(0) and `RefineBA`(1) corresponds to our warm-up algorithm to find a *maximal* common independent set from Section 3. In particular `RefineAB`(0) finds a maximal B_1 such that $S + B_1 \in \mathcal{I}_1$, and `RefineAB`$(1)$ shrinks B_1 such that $S + B_1 \in \mathcal{I}_2$ too.

▶ **Lemma 23.** *`RefineAB` and `RefineBA` can each be implemented in either:*

- $O(1)$ *rounds of polynomially many rank-queries, or*
- $O(\sqrt{n})$ *rounds of polynomially many independence-queries.*

Proof. The refine procedures only need to find a maximal independent set (for a single matroid) twice, so we can apply Lemma 15. ◀

The following properties are proven in [5].

▶ **Lemma 24** (from [5, Lemmas 40-42])**.** *Both `RefineAB` and `RefineBA` preserve the phase invariants. Also: after `RefineAB`(k) is run, we have $|A_k| = |B_{k+1}|$ (unless $k = 0$). After `RefineBA`(k) is run, we have $|B_k| = |A_{k+1}|$ (unless $k = \ell$).*

▶ **Observation 25.** *Lemma 24 can be used to messure progress. In particular, after running* `RefineAB`(k)*,* $|A_k| = |B_{k+1}|$*, so a total of* $|A_k^{old}| - |B_{k+1}^{old}|$ *elements must have changed types (*$x \in A_k^{old}$ *might have been* removed*, while a* $x \in F_{k+1}^{old}$ *might have been* selected*). Similarly* `RefineBA`(k) *will change types of* $|B_k^{old}| - |A_{k+1}^{old}|$ *elements. Note that each element can only change type at most twice (from* fresh *to* selected *to* removed*), so this observation can be used to measure progress.*

4.2.2 Refining Globally

In the sequential algorithms of [2, 5], a refinement pass consists of running `RefineAB`(k) and `RefineAB`(k) for all k in sequence. However, in the parallel setting we can do better. Since `RefineAB` and `RefineBA` only change things locally in two adjacent layers, we observe that we can perform several of these refinement operations in parallel.

Algorithm 4 `Refine`().

1: In parallel, run `RefineAB`(k) for all even $0 \leq k \leq \ell$.
2: In parallel, run `RefineBA`(k) for all odd $0 \leq k \leq \ell$.

▶ **Lemma 26.** *`Refine` can be implemented in either:*
- $O(1)$ *rounds of polynomially many rank-queries, or*
- $O(\sqrt{n})$ *rounds of polynomially many independence-queries.*

The following Lemma 27 will be useful to bound the number of `Refine` calls needed in our final algorithm, and is similar to [5, Corollary 43].

▶ **Lemma 27.** *Suppose that* $|B_k| = |A_{k+1}|$ *for all odd* $1 \leq k \leq \ell - 1$ *and that* $S + B_\ell \in \mathcal{I}_2$*, before `Refine` is run. After `Refine` is run we have:*

(i) $|B_k| = |A_{k+1}|$ *for all odd* $1 \leq k \leq \ell - 1$*, still.*
(ii) $S + B_\ell \in \mathcal{I}_2$*, still.*
(iii) $|B_1'| - |B_\ell|$ *elements have changed their type, where* B_1' *is any maximal subset of* $(D_1 - R_1)$ *such that* $S + B_1' \in \mathcal{I}_1$*.*

Proof. Property (i) is true, since it is true just after we run `RefineBA`(k), by Lemma 27, and this is what is done in the last step of `Refine`. Similarly property (ii) is true, since `RefineBA`(ℓ) ensures this when "shrinking" B_ℓ (see how B is picked in Algorithm 3 line 1).

What remains is to prove property (iii). Let $(B_1^{old}, A_2^{old}, \ldots)$ be the sets before `Refine` is run. In the first line of Algorithm 4, we run `RefineAB`(2), `RefineAB`(4), `RefineAB`(6), $\ldots$, which according to Lemma 24 and Observation 25, has incurred a total of $\sum |A_k^{old}| - |B_{k+1}^{old}| = |B_1^{old}| - |B_\ell^{old}|$ type-changes (the sum telescopes since we assume $|B_{k-1}^{old}| = |A_k^{old}|$).

Also note that we run `RefineAB`(0), which extends B_1 until it is a maximal subset of $D_1 \setminus R_1$ such that $S + B_1 \in \mathcal{I}_1$ (line 1 of Algorithm 2). This means that an additional $|B_1'| - |B_1^{old}|$ elements have changed their type – from *fresh* to *selected* – in the first layer, where B_1' is the value of B_1 we get after running `RefineAB`(0). Note that this B_1' is a maximal subset of $(D_1 - R_1)$ such that $S + B_1' \in \mathcal{I}_1$ (see line 1 of Algorithm 2).

Hence, in the first line of Algorithm 4, $|B_1'| - |B_\ell|$ types have changed. We might additionally change types of more elements when running the second line of Algorithm 4. ◀

▶ Remark 28. We measure progress in terms of Lemma 27. Since each element can only change types twice (from *fresh* $\mapsto$ *selected* $\mapsto$ *removed*), there will be in total $O(n)$ type-changes. If we just run `Refine`, we might need to do so $O(n)$ times (in the case when $|B_1'| - |B_\ell|$ is

constant), so this is not good enough to obtain a sublinear round parallel algorithm. Like we did in the easier case of *maximal* common independent set, we must swap to a different strategy when the progress of refining stagnates, i.e. when $|B'_1| - |B_\ell|$ is relatively small.

4.3 Second Stage: Finding the Remaining Augmenting Paths

When $|B'_1| - |B_{\ell+1}|$ is relatively small, we fall back to finding a special kind of augmenting paths one-by-one. We will show that we only need to find $|B'_1| - |B_{\ell+1}|$ many such paths before we get stuck and have found our *maximal* augmenting set (i.e. the desired blocking flow). These special kind of augmenting paths we consider are essentially augmenting paths in the exchange graph *with respect to our partial augmenting set*. They were first introduced by [2], and are called *valid paths*.

▶ **Definition 29** (Valid path, from [2, Definition 31]). $(b_i, a_{i+1}, b_{i+2}, \ldots, a_{\ell-1}, b_\ell, t)$ is a *valid path* (w.r.t. our partial augmenting set) *starting at* b_i if for all $k \geq i$:

(a) $a_k \in F_k$ for even k and $b_k \in F_k$ for odd k.
(b) $S - A_{i-1} + B_i + b_i \in \mathcal{I}_1$.
(c) $S + B_\ell + b_\ell \in \mathcal{I}_2$.
(d) $S - A_{k+1} + B_k - a_{k+1} + b_k \in \mathcal{I}_2$ for odd k.
(e) $S - A_k + B_{k+1} - a_k + b_{k+1} \in \mathcal{I}_1$ for even k.

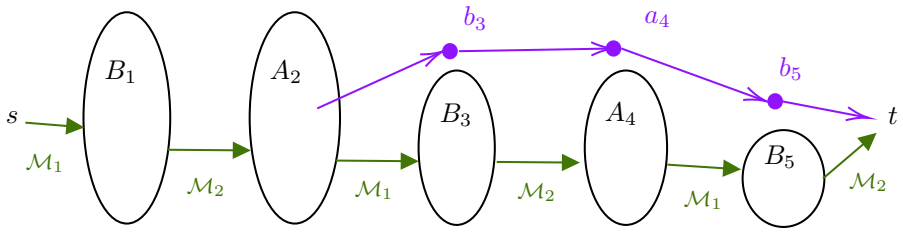

Figure 1 An example of a valid path (b_3, a_4, b_5, t) starting at b_3.

▶ Remark 30. Compare the definition of valid paths to the edges in the exchange graph from Definition 7. Essentially, items (b-e) corresponds to edges of the exchange graph $G(S + B_1 - A_2 + B_3 - A_4 + \cdots + B_\ell)$ of S *after* augmenting along our partial augmenting set. Note also that item (b) can only hold when $|A_{i-1}| > |B_i|$ (or, when $i = 1$ and $A_0 = \emptyset$ is an "imaginary" boundary set).

▶ **Lemma 31** (Augmenting along a valid path [2, Lemma 33]). *Suppose that* $|B_k| = |A_{k+1}|$ *for all odd* $1 \leq k \leq \ell - 1$ *and that* $S + B_\ell \in \mathcal{I}_2$.[15] *If* $(b_i, a_{i+1}, b_{i+2}, \ldots, b_\ell, t)$ *is a valid path starting at* b_i*, then* $(B_1, A_2, \ldots, B_{i-2}, A_{i-1}, B_i + b_i, A_{i+1} + a_{i+1}, \ldots, B_\ell + b_\ell)$ *is a partial augmenting set satisfying the phase invariants and with* $S + B_\ell + b_\ell \in \mathcal{I}_2$.

Proof sketch. It is easy to verify that all the properties in the definition of a partial augmenting set are still satisfied after the augmentation. Moreover, the phase invariants are also true, since the sets B_k and A_k are only extended by the augmentation (so an element deemed useless before remains useless). ◀

[15] These conditions are actually redundant, since they are covered by items (d) and (c) in the definition of the valid paths. However, they make the intuition slightly easier, and our algorithm maintains them.

▶ **Lemma 32.** *We can find a valid path, if one exists, in a single round of $O(n^2)$ queries.*

Proof. Finding a valid path in a single round of queries is not very different from finding a normal augmenting path. In a single round we query all potential "directed edges", that is all potential (a_k, b_{k+1}) or (b_k, a_{k+1}) pairs satisfying the items of Definition 29 (valid paths). Then we can combine these edges to form a valid path, or else determine that no valid path exist. ◀

▶ Remark 33. After augmenting along a valid path, $|B_\ell|$ increases by one. Let B_1' be any any maximal subset of $(D_1 - R_1)$ such that $S + B_1' \in \mathcal{I}_1$. Note that we always know that $|B_1'| \geq |B_1| \geq |B_\ell|$. Hence, we need to only find at most $|B_1'| - |B_\ell|$ many valid paths to augment along after we finished the first stage.

We also need the following lemma saying that if, for an element $x \in F_k$, there is no "partial" valid path $(x, \ldots, a_{\ell-1}, b_\ell, t)$ (satisfying all items of a valid path, except maybe item (b) of Definition 29), then it is safe to delete x. We prove this by showing that if x has no "out-edges" (of the form of items (c-e) in Definition 29), then it can be removed.

▶ **Lemma 34.** *Suppose that $|B_k| = |A_{k+1}|$ for all odd $1 \leq k \leq \ell - 1$ and that $S + B_\ell \in \mathcal{I}_2$. Then:*

- *If $b_\ell \in F_k$ is such that $S + B_\ell + b_\ell \notin \mathcal{I}_2$, we can safely remove it.*
- *For a given $b_k \in F_k$, if there exist no $a_{k+1} \in F_{k+1}$ such that $S - A_{k+1} + B_k - a_{k+1} + b_k \in \mathcal{I}_2$, then it is safe to remove b_k.*
- *For a given $a_k \in F_k$, if there exist no $b_{k+1} \in F_{k+1}$ such that $S - A_k + B_{k+1} - a_k + b_{k+1} \in \mathcal{I}_1$, then it is safe to remove a_k.*

Proof sketch. We must argue that the phase invariants are preserved when the elements are removed. It is straightforward to verify that phase invariants (c) and (d) hold in all these three cases. As a black-box intuition, we can imagine temporarily selecting the element x we want to remove, and then running either `RefineAB` or `RefineBA` and note that this procedure can immediately remove x again (and the refine-procedures preserves the invariants). ◀

4.4 Combining the Stages

Now we present the full algorithm of a phase, whose goal is to find a maximal augmenting set, that is a "blocking flow". Pseudo-code can be found in Algorithm 5, which is parametrized by a cut-off threshold p (which will be different for rank- and independence-query) for when to move from the first to second stage.

▶ Remark 35. After the two stages, we will have some partial augmenting set $(B_1, A_2, \ldots, B_\ell)$ such that there are no more valid paths. However, it is not yet an actual augmenting set, for instance it can be the case that $|A_k| > |B_{k+1}|$ for some k. Still, we can argue that $(B_1, A_2, \ldots, B_\ell)$ contains some *maximal augmenting set* $(\tilde{B}_1, \tilde{A}_2, \ldots, \tilde{B}_\ell)$ with $\tilde{B}_\ell = B_\ell$. So we will need a short extra clean-up step to reduce our partial augmenting set to such a maximal augmenting set.

Note that it is possible to show that we actually can directly augment along our partial augmenting set $(B_1, A_2, \ldots, B_\ell)$ which the algorithm finds (this relies on the extra properties that $|B_k| = |A_{k+1}|$ and $S + B_\ell \in \mathcal{I}_2$). That is $S' = S + B_1 - A_2 + B_2 - \cdots + B_\ell \in \mathcal{I}_1 \cap \mathcal{I}_2$ is a common independent set. Additionally $|S'| = |S| + |B_\ell|$, so we have increased the size of S as much as we would have if we found the the maximal augmenting set $(\tilde{B}_1, \tilde{A}_2, \ldots, \tilde{B}_\ell)$ instead. However, there is a critical problem with this approach: *there can be short augmenting paths in $G(S')$*. This means that such an approach will have failed to eliminate all (s,t)-paths of length $\leq \ell$. Hence the clean-up step is actually necessary.

Algorithm 5 Implementation of phase $(\ell+1)/2$.

1: In parallel query all potential edges of the exchange graph $G(S)$ (see Definition 7).
2: Find the distance layers $D_1, D_2, \dots D_\ell$.
3: Initialize $B_k = \emptyset$, $A_k = \emptyset$, $R_k = \emptyset$ and $F_k = D_k$ for all k.

4: **while** true **do** ▷ Stage 1
5: Find a maximal $B' \subseteq D_1 - R_1$ such that $S + B' \in \mathcal{I}_1$ (using Lemma 15).
6: **if** $|B'| - |B_\ell| \leq p$ **then break**
7: Call `Refine`() (Algorithm 4).

8: **while** true **do** ▷ Stage 2
9: In a single round, find a valid path if one exists (Lemma 32).
10: **if** no valid path exists **then break**
11: Augment the partial augmenting set along the found valid path (Lemma 31).

▷ Clean-up
12: Remove all elements which do not have any partial valid path from them (see Lemma 34).
13: Sequentially, call $\texttt{RefineBA}(\ell), \texttt{RefineAB}(\ell-1), \texttt{RefineBA}(\ell-2), \dots \texttt{RefineAB}(0)$
14: Augment along the *maximal augmenting set* $\Phi = (B_1, A_2, \dots, B_\ell)$:
15: that is, update $S \leftarrow S + B_1 - A_2 + B_3 + \dots + B_\ell$.

Correctness. In the beginning of Algorithm 5 the following hold: (i) the phase invariants (Definition 20); (ii) $|B_k| = |A_{k+1}|$ for all odd $1 \leq k \leq \ell - 1$; and (iii) $S + B_\ell \in \mathcal{I}_2$.

In the first stage, whenever we call `Refine`, the above properties (i-iii) are all preserved according to Lemma 27. Similarly, in the second stage, whenever we augment along a valid path, the above properties (i-iii) are also preserved, by Lemma 31.

What remains to be shown is that after the clean-up phase, $\Phi = (B_1, A_2, \dots, B_\ell)$ is a maximal augmenting set. We prove this by showing that these refine calls cannot *select* any new elements, that is they do not add any elements to the sets B_k or A_k. If we show this, then we know that $|B_\ell| = |A_{\ell-1}| = \dots = |B_1|$ after all these refine calls, as $\texttt{RefineAB}(\ell-1)$ reduced $|A_{\ell-1}|$ to match $|B_\ell|$; $\texttt{RefineBA}(\ell-2)$ reduces $|B_{\ell-2}|$ to match $|A_{\ell-1}|$; etc.

To argue that $\texttt{RefineAB}(k)$ does not select any new elements, we note that if it added b_k to B_k, it meant that $S - A_{k+1} + B_k + b_k \in \mathcal{I}_1$. However, since b_k was not removed in line 12 of the algorithm, there must have been a valid path starting from b_k, which is a contradiction. The argument for $\texttt{RefineBA}(k)$ is the same. Note that since we only remove elements, no new valid paths can occur.

Now, after $|B_1| = |A_2| = \dots = |B_\ell|$, $(B_1, A_2, \dots, B_\ell)$ forms a *maximal* augmenting set. If it did not, there must have been some path $(b_1, a_2, \dots, b_\ell)$ which we can add to it, but this is impossible, since this path would have been a valid path (starting at b_1).

Rounds of adaptivity. The first stage of Algorithm 5 runs in $O(n/p \cdot \mathcal{T}_{\texttt{Refine}})$ rounds if $\mathcal{T}_{\texttt{Refine}}$ is the number of rounds needed to run `Refine`() once (Lemma 26 gives $\mathcal{T}^{\text{rank}}_{\texttt{Refine}} = O(1)$ and $\mathcal{T}^{\text{indep}}_{\texttt{Refine}} = O(\sqrt{n})$). This is since each time we run refine we will have at least p type-changes (Lemma 27), and in total each of the n elements can change types at most twice.

The second stage of Algorithm 5 runs in $O(p)$ rounds, both for independence and rank-oracle, by Lemma 32. Picking p optimally gives: $O(\sqrt{n})$ rounds of rank-queries (with $p = \sqrt{n}$) or $O(n^{3/4})$ rounds of independence-queries (with $p = n^{3/4}$) for the first and second stages combined.

Finally the clean-up stage runs, sequentially, with $O(\ell)$ refinement operations. This takes $O(\ell)$ rounds of rank-queries or $O(\ell\sqrt{n})$ rounds of independence queries, so this depends on the number of layers. However, we argue that we can ignore this term for the interesting range of ℓ. This is because when ℓ is too large ($> \sqrt{n}$ for rank-queries and $> n^{1/4}$ for independence-queries), we know by Lemma 14 that there are only $O(1/\ell)$ many augmenting paths left in total, and we can instead find them one-by-one in at most $O(\sqrt{n})$ rounds for rank-queries or $O(n^{3/4})$ rounds for independence queries.

Concluding, we have argued that we can implement a blocking-flow phase in $O(\sqrt{n})$ rounds of rank-queries or $O(n^{3/4})$ rounds of independence-queries.

Approximation algorithm. Running Algorithm 5 for $O(1/\varepsilon)$ phases eliminates all paths in the exchange graph of length $O(1/\varepsilon)$ (Lemma 13), so by Lemma 14 we know that the common independent set S we end up with is a $(1-\varepsilon)$-approximation. The adaptivity is thus $O(\sqrt{n}/\varepsilon)$ rounds of rank-queries or $O(n^{3/4}/\varepsilon)$ rounds of independence-queries. Hence we have shown a $(1-\varepsilon)$-approximation algorithm using $O(\sqrt{n}/\varepsilon)$ rounds of (polynomially many) rank-queries or $O(n^{3/4}/\varepsilon)$ rounds of (polynomially many) independence-queries, which proves Theorem 18.

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 *ICALP*, volume 198 of *LIPIcs*, pages 31:1–31:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.31.

3 Joakim Blikstad, Jan van den Brand, Sagnik Mukhopadhyay, and Danupon Nanongkai. Breaking the quadratic barrier for matroid intersection. In *STOC*, pages 421–432. ACM, 2021. doi:10.1145/3406325.3451092.

4 Deeparnab Chakrabarty, Yu Chen, and Sanjeev Khanna. A polynomial lower bound on the number of rounds for parallel submodular function minimization and matroid intersection. In *FOCS*, pages 37–48. IEEE Computer Society, 2021. doi:10.1109/FOCS52979.2021.00013.

5 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. doi:10.1109/FOCS.2019.00072.

6 William H. Cunningham. Improved bounds for matroid partition and intersection algorithms. *SIAM J. Comput.*, 15(4):948–957, 1986. doi:10.1137/0215066.

7 Efim A Dinic. Algorithm for solution of a problem of maximum flow in networks with power estimation. In *Soviet Math. Doklady*, volume 11, pages 1277–1280, 1970.

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

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

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

11 Stephen A. Fenner, Rohit Gurjar, and Thomas Thierauf. Bipartite perfect matching is in quasi-nc. *SIAM J. Comput.*, 50(3), 2021. doi:10.1137/16M1097870.

12 Sumanta Ghosh, Rohit Gurjar, and Roshan Raj. A deterministic parallel reduction from weighted matroid intersection search to decision. In *SODA*, pages 1013–1035. SIAM, 2022. doi:10.1137/1.9781611977073.44.

13 Rohit Gurjar and Thomas Thierauf. Linear matroid intersection is in quasi-nc. *Comput. Complex.*, 29(2):9, 2020. doi:10.1007/s00037-020-00200-z.

14 John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973. doi:10.1137/0202019.

15 Richard M. Karp, Eli Upfal, and Avi Wigderson. The complexity of parallel computation on matroids. In *FOCS*, pages 541–550. IEEE Computer Society, 1985. doi:10.1109/SFCS.1985.57.

16 Richard M. Karp, Eli Upfal, and Avi Wigderson. Constructing a perfect matching is in random NC. *Comb.*, 6(1):35–48, 1986. doi:10.1007/BF02579407.

17 Richard M. Karp, Eli Upfal, and Avi Wigderson. The complexity of parallel search. *J. Comput. Syst. Sci.*, 36(2):225–253, 1988. doi:10.1016/0022-0000(88)90027-X.

18 Eugene L. Lawler. Matroid intersection algorithms. *Math. Program.*, 9(1):31–56, 1975. doi:10.1007/BF01681329.

19 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. doi:10.1109/FOCS.2015.68.

20 László Lovász. On determinants, matchings, and random algorithms. In *FCT*, pages 565–574. Akademie-Verlag, Berlin, 1979.

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