Fast Matroid Intersection Algorithms

Sunghyeon Jo

Seoul National University

Project-Hardness
Sep 27, 2022

Overview

1. Introduction
   - Matroid
   - Matroid Intersection
2. Faster Algorithms
   - Oracle Models
   - Core ideas and algorithms

1. Introduction
   - Matroid
   - Matroid Intersection

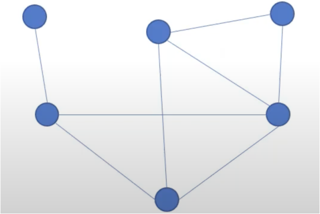
2. Faster Algorithms

Matroid

Matroid

A tuple $M = (V, \mathcal{I})$ for finite set V and $\mathcal{I} \subset 2^V$ is called a matroid if
1. $\phi \in \mathcal{I}$
2. $Y \subset X, X \in \mathcal{I} \Rightarrow Y \in \mathcal{I}$
3. $X, Y \in \mathcal{I}, |X| < |Y| \Rightarrow y \in Y \setminus X$ exists such that $X + y \in \mathcal{I}$

- $rank(S) = \max\{|A| : A \subset S, A \in \mathcal{I}\}$
- Basis: maximal/maximum independent set
- Ex) Graphic Matroid. V : edges, $\mathcal{I}$: acyclic subgraphs



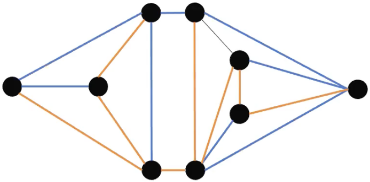
Matroid Problems

- Matroid Intersection
  - $\mathcal{M}_1 = (V, \mathcal{I}_1), \mathcal{M}_2 = (V, \mathcal{I}_2)$
  - Find maximum size $S \in \mathcal{I}_1 \cap \mathcal{I}_2$
- Matroid Union
  - $\mathcal{M}_1 = (V_1, \mathcal{I}_1), \cdots \mathcal{M}_k = (V_k, \mathcal{I}_k)$
  - Find maximum size $S = S_1 \cup \cdots \cup S_k$ where $S_i \in \mathcal{I}_i$
- k-fold Matroid Union
  - $\mathcal{M} = (V, \mathcal{I})$
  - Find maximum size $S = S_1 \cup \cdots \cup S_k$ where $S_i \in \mathcal{I}$

Many problems can be solved.
Ex) Bipartite Matching, Colorful Spanning Tree, k-disjoint spanning tree, Arboricity

Examples

- Bipartite Matching for $V = L \cup R$
  - Matroid Intersection of two counting matroid $\mathcal{M}_L$, $\mathcal{M}_R$
  - indepedent set in M_L: subgraph s.t. each vertex in L have ≤ 1 edges
- Colorful Spanning tree
  - Matroid Intersection of graphic matroid and color matroid
  - color matroid: subgraph s.t. each color is used at most once
- k-disjoint spanning tree
  - k-fold Matroid Union for graphic matroid



k-disjoint spanning tree for k=2

How to solve matroid problems?

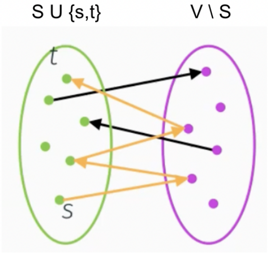
Matroid Union, k-folded matroid union can be reduced to Matroid intersection. =¿ Algorithm for matroid intersection is important!

How to solve Matroid intersection? By idea of *exchange graph* $G(S)$ for $S \in \mathcal{I}_1 \cap \mathcal{I}_2$.

Exchange graph

Exchange graph

For two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$, $\mathcal{M}_2 = (V, \mathcal{I}_2)$ over the same ground set and an $S \in \mathcal{I}_1 \cap \mathcal{I}_2$, the *exchange graph* with respect to S is a directed bipartite graph $G(S) = (V \cup \{s, t\}, E)$ where:
- $E = E_1 \cup E_2 \cup E_s \cup E_t$
- $E_1 = \{(u, v) | u \in S, v \in V \setminus S, S - u + v \in \mathcal{I}_1\}$
- $E_2 = \{(v, u) | u \in S, v \in V \setminus S, S - u + v \in \mathcal{I}_2\}$
- $E_s = \{(s, v) | v \in V \setminus S, S + v \in \mathcal{I}_1\}$
- $E_t = \{(v, t) | v \in V \setminus S, S + v \in \mathcal{I}_2\}$



Matroid Intersection Algorithm

Augmenting Path Lemma

Let P be a shortest (s, t)-path of $G(S)$. Then, the set $S' = S \oplus (V(P) \setminus \{s, t\})$ is a common independent set with $|S'| = |S| + 1$. On the other hand, if t is unreachable from s in $G(S)$, then S is *a largest common independent set*.

From the above, we can obtain the following algorithm for matroid intersection.

1. Initialize $S = \emptyset$.
2. Find a shortest (s, t)-path P in $G(S)$.
3. While P exists:
    1. Update S with $S' = S \oplus (V(P) \setminus \{s, t\})$ (called *augmenting S along path P*).
    2. Find a new shortest (s, t)-path P in $G(S)$.

Oracle - How to define fast?

Independence-oracle model

For a matroid $M = (V, \mathcal{I})$, the algorithm can access the oracle via the following operation for an arbitrary subset S of V.

- $Query(S)$: Return true if $S \in \mathcal{I}$, false otherwise.

Rank-oracle model

For a matroid $M = (V, \mathcal{I})$, the algorithm can access the oracle via the following operation for an arbitrary subset S of V.

- $Query(S)$: Return the rank of S, i.e., the size of the largest independent subset of S.

rank oracle is stronger than independence oracle. Need fewer oracle queries to solve the problem $\Rightarrow$ Fast algorithm!

Progress of Matroid Intersection Algorithms

Let r be the maximum size of the common independent set.

- Exact algorithm
  - Independent-oracle model
    - $\tilde{O}(nr)$ oracle queries ([Ngu19] , [CLS+19])
    - $\tilde{O}(nr^{3/4})$ oracle queries ([Bli21])
  - Dynamic-oracle model
    - $\tilde{O}(n\sqrt{r})$ oracle queries ([CLS+19])
- Approximate algorithm
  - Independent-oracle model
    - $\tilde{O}(n^{1.5}/\epsilon^{1.5})$ oracle queries ([CLS+19])
  - Rank-oracle model
    - $\tilde{O}(n/\epsilon)$ oracle queries ([CLS+19])

Matroid Intersection Algorithm Recall

Algorithm

1. Initialize $S = \emptyset$.
2. Find a shortest (s, t)-path P in $G(S)$.
3. While P exists:
    1. Update S with $S' = S \oplus (V(P) \setminus \{s, t\})$.
    2. Find a new shortest (s, t)-path P in $G(S)$.

Question: How to find the shortest (s, t)-path in $G(S)$ for given S?

Matroid Intersection Algorithm Recall

Algorithm

1. Initialize $S = \emptyset$.
2. Find a shortest (s, t)-path P in $G(S)$.
3. While P exists:
    1. Update S with $S' = S \oplus (V(P) \setminus \{s, t\})$.
    2. Find a new shortest (s, t)-path P in $G(S)$.

Question: How to find the shortest (s, t)-path in $G(S)$ for given S?

Simple answer: Do BFS from s in $G(S)$!

Matroid Intersection Algorithm Recall

Algorithm

1. Initialize $S = \emptyset$.
2. Find a shortest (s, t)-path P in $G(S)$.
3. While P exists:
    1. Update S with $S' = S \oplus (V(P) \setminus \{s, t\})$.
    2. Find a new shortest (s, t)-path P in $G(S)$.

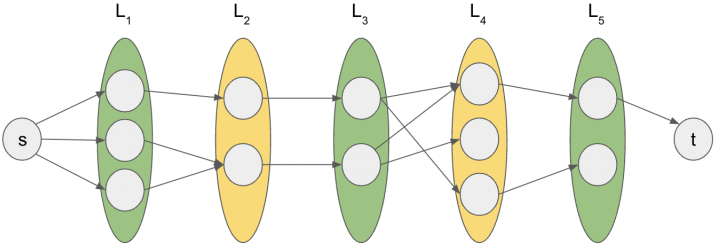
Question: How to find the shortest (s, t)-path in $G(S)$ for given S?

Simple answer: Do BFS from s in $G(S)$!

Question: how to find outgoing edges from a vertex?

BFS in $G(S)$

want to do: BFS from s and find all layers $L_1, L_2, \cdots L_d$ where
$L_i : \{v|dist(s,v) = i\}$



BFS in $G(S)$

to do BFS in $G(S)$, we have to find edges from the current vertex u to v that have not visited yet.
Let's consider the following problem.

Problem *Find*E_2

Let $S \in \mathcal{I}_1 \cap \mathcal{I}_2$. Given $u \in V \setminus S$ and $B \subset S$, find $v \in B$ s.t. $(u, v) \in G(S)$. i.e, $S + u - v \in \mathcal{I}_2$, or decide such v does not exist.

If the current vertex u is in $V \setminus S$, we can do BFS in $G(S)$ by solving the above problem simply by giving B as unvisited vertices in S.

FindE_2 can be done with binary search: list B as $b_1, \cdots b_k$ and then find first i such that $S + u - \{b_1, \cdots b_i\} \in \mathcal{I}_2$. Then $S + u + b_i \in \mathcal{I}_2$ holds. This requires $O(\log n)$ independence/rank queries.

BFS in $G(S)$

What if the current vertex u is in S? we have to find $(u, v) \in E_1$.

Problem *Find*E_1

Let $S \in \mathcal{I}_1 \cap \mathcal{I}_2$. Given $u \in S$ and $A \subset V \setminus S$, find $v \in A$ s.t. $(u, v) \in G(S)$. i.e, $S - u + v \in \mathcal{I}_1$, or decide such v does not exist.

If we can use rank queries, then it is the same to *Find*E_2: list A as $a_1, \cdots a_k$ and find first i such that $S - u + \{a_1, \cdots a_i\} >= rank(S)$. By property of matroids, $S - u + a_i \in \mathcal{I}_1$ holds.

However, solving problem *Find*E_1 with $O(\log n)$ independence queries is not easy.

Instead, for $U \subset S$, we can find all $v \in A$ such that edge (u, v) exists for some $u \in U$. This can be done in $O(n)$ independence quires.

BFS in $G(S)$

using algorithms for *Find*E_1 and *Find*E_2, BFS from s to t can be done in

- $O(n \log n)$ rank queries
- $O(n \log n + nl)$ rank queries where $l = dist(s, t)$

Therefore, simply repeating BFS, matroid intersection can be solved in $O(nr \log n)$ rank queries or $O(r(n \log n + n^2))$ independence queries.

BFS in $G(S)$

By far, we have $\tilde{O}(nr)$ rank queries algorithm.
In the Hopcroft-Karp algorithm, we use "Blocking Flow". We can do the same way for this and it will reduce the query complexity.

Blocking Flow

Algorithm *BlockFlow(S)*

1. Perform BFS from vertex v. Let d_v be the distance from s to v in $G(S)$.
2. Define L_i as $L_i = \{v \mid d_v = i\}$. Initialize *visited* as an empty set.
3. Repeat execute *DFS*(v) until S remains unchanged.
4. Return S.

Algorithm *DFS(v)*

1. If $v = t$:
   - Augment S using the current path from s to t.
   - Return.
2. Add v to the set *visited*: *visited* $=$ *visited* $\cup \{v\}$.
3. Identify a vertex x such that (v, x) is an edge in $G(S)$ and x belongs to L_{d_v+1} but not in *visited*.
4. If such an x exists, run *DFS*(x). Otherwise, return.

Blocking Flow

Let $S' \leftarrow BlockingFlow(S)$.

Want To Show:

- At most $\tilde{O}(n)$ queries are used in one execution of $BlockingFlow(S)$.
- $dist_{G(S')}(s,t) > dist_{G(S)}(s,t)$

First, no vertices will visited twice in a $BlockingFlow$ execution. Therefore, it requires $O(n \log n)$ rank queries.

Monotonicity Lemma

Why $dist_{G(S')}(s,t) > dist_{G(S)}(s,t)$?

First of all, Following lemma holds. It is just like monotonicity lemma of Hopcroft-Karp algorithm.

Why distance is increasing strictly?

Assume that we obtained S' by augmenting S along P, an augmenting path of $G(S)$. Then, the following are satisfied:

1. $d(s,a) < d(s,t) \Rightarrow d(s,a) \leq d'(s,a)$. Likewise, $d(a,t) < d(s,t) \Rightarrow d(a,t) \leq d'(a,t)$.
2. $d(s,a) \geq d(s,t) \Rightarrow d'(s,a) \geq d(s,t)$. Likewise, $d(a,t) \geq d(s,t) \Rightarrow d'(a,t) \geq d(s,t)$.

where d and d' are distance function of $G(S)$ and $G(S')$, respectively.

Why distance is increasing strictly?

Want to show: $dist_{G(S')}(s,t) > dist_{G(S)}(s,t)$

Proof

Let d be the distance function of $G(S)$ and d' be that of $G(S')$.
Define l as $d(s,t)$.

Assume there exists an (s,t)-path P in $G(S')$ with length $\leq l$.
By the Monotonicity Lemma, the length of P is $\geq l$. Thus, its length is exactly l.

Let the path P be represented as $P = (s, p_1, \cdots, p_{l-1}, t)$.

Why distance is increasing strictly?

Want to show: $dist_{G(S')}(s,t) > dist_{G(S)}(s,t)$

Proof

By Monotonicity Lemma, $d'(s,p_i) \geq d(s,p_i)$ and $d'(p_i,t) \geq d(p_i,t)$.

Given that p_i lies on P of length l, we can conclude that $d'(s,p_i) = d(s,p_i) = i$ and $d'(p_i,t) = d(p_i,t) = l - i$.

Since *BlockFlow* process is terminated, at least one p_i must have been visited during the process. As p_i can be visited only once in the *BlockFlow* process, it follows that $p_i \in S$ if and only if $p_i \notin S'$.

However, the condition $d(s,p_i) = d'(s,p_i) = i$ is untenable, given that paths in the exchange graph $G(S)$ alternate between members of S and $V \setminus S$. ■

Algorithm

We can solve matroid intersection by repeating $S \leftarrow \mathit{BlockFlow}(S)$, and each *BlockFlow* execution uses $O(n \log n)$ rank queries.
Therefore, the number of different $d(s,t)$ for this process decides query complexity.
Meanwhile, the following lemma holds:

Lemma

For a common independent set S with $|S| < r$, there is an augmenting path in $G(S)$ where its length $\leq 2|S|/(r - |S|) + 2$.

This is because for two common independent set $S_1, S_2 (|S_1| < |S_2|)$, there are $|S_2| - |S_1|$ disjoint (s,t)-path in $G(S_1)$ which consists vertices only from $S_1 \cap S_2 \cap \{s,t\}$ ([Cunningham86]).

Algorithm

Lemma

For a common independent set S with $|S| < r$, there is an augmenting path in $G(S)$ where its length $\leq 2|S|/(r - |S|) + 2$.

By the lemma, there are only $O(\sqrt{r})$ different length of augmenting paths. Therefore our algorithm uses at most $O(n \log n \sqrt{r})$ rank quries.

On the other hand, for independence queries, one BFS uses at most $O(nl + n \log n)$ queries. And by the lemma, total summation of l is bounded to $O(r \log r)$. Therefore, we obtained an $O(nr \log n)$ independence queries algorithm by just repeating BFS.

$\therefore$ Matroid Intersection is can be done by

- $\tilde{O}(n\sqrt{r})$ rank queries
- $\tilde{O}(nr)$ independence queries

Faster Algorithms

- $1 - \epsilon$ approximation algorithm that uses $O(n^{1.5}/\epsilon^{1.5})$ indepence queries (2019)
  - use brilliant "Augmenting set" and "Partially augmenting set" idea.
- exact algorithm that uses $O(nr^{3/4})$ independence queries (2021)
  - uses above approximation algorithm
- Dynamic Rank Oracle Model - uses $\tilde{O}(n + r^{1.5})$ oracle queries for matorid intersection. (2023)
  - more practical, since each query runs $\tilde{O}(1)$ in amortized manner.

Dynamic-rank-oracle Model

- Limit of rank query / independence query: $rank(S)$ takes at least $O(\min(|S|, r))$ time
- For most matroid we interests in: one element update is not expensive. i.e. $rank(S) \to rank(S \pm \{e\})$ is fast
  - graphic matroid: fully-dynamic counting components: $O(\text{polylog } n)$
  - counting matroid: $O(1)$
- Fully persistent update costs only additional $\log N$ factor [DSST'86]. i.e, $S_i = S_j \pm \{e\}$ for $j < i$.

Dynamic-rank-oracle Model

Dynamic-rank-oracle model

For a matroid $M = (V, \mathcal{I})$, starting from $S_0 = \phi$ and $k = 0$, the algorithm can access the oracle via the following three operations.

- $Insert(v, i)$: Create a new set $S_{k+1} := S_i \cup \{v\}$ and increment k by one.
- $Delete(v, i)$: Create a new set $S_{k+1} := S_i \setminus \{v\}$ and increment k by one.
- $Query(i)$: Return the rank of S_i, i.e., the size of the largest independent subset of S_i

$O(T)$ Dynamic rank queries for graphic, counting, color matroids guarantees $\tilde{O}(T)$ time complexity!

Dynamic-rank-oracle model

Theorem

Matroid intersection can be solved in $\tilde{O}(n + r^{1.5})$ dyanmic rank queries.

Basic idea is the same: do BFS and find all distance from s in $G(S)$.

Since Binary search for BFS requires many dynamic queries (difference between adjacent queries may big), data structure for precomputing is required for this algorithm.

Idea: Construct binary search tree and lazily update when S changes

Results

problems	our bounds	state-of-the-art results
(Via k-fold matroid union)		
k-forest[8]	$\tilde{O}(\lvert E\rvert + (k\lvert V\rvert)^{3/2})$ ✓	$\tilde{O}(k^{3/2}\lvert V\rvert\sqrt{\lvert E\rvert})$ [GW88]
k-pseudoforest	$\tilde{O}(\lvert E\rvert + (k\lvert V\rvert)^{3/2})$ ✗	$\lvert E\rvert^{1+o(1)}$ [CKL+22]
k-disjoint spanning trees	$\tilde{O}(\lvert E\rvert + (k\lvert V\rvert)^{3/2})$ ✓	$\tilde{O}(k^{3/2}\lvert V\rvert\sqrt{\lvert E\rvert})$ [GW88]
arboricity[9]	$\tilde{O}(\lvert E\rvert\lvert V\rvert)$ ✗	$\tilde{O}(\lvert E\rvert^{3/2})$ [Gab95]
tree packing	$\tilde{O}(\lvert E\rvert^{3/2})$	$\tilde{O}(\lvert E\rvert^{3/2})$ [GW88]
Shannon Switching Game	$\tilde{O}(\lvert E\rvert + \lvert V\rvert^{3/2})$ ✓	$\tilde{O}(\lvert V\rvert\sqrt{\lvert E\rvert})$ [GW88]
graph k-irreducibility	$\tilde{O}(\lvert E\rvert + (k\lvert V\rvert)^{3/2} + k^2\lvert V\rvert)$ ✓	$\tilde{O}(k^{3/2}\lvert V\rvert\sqrt{\lvert E\rvert})$ [GW88]
(Via matroid union)		
(f,p)-mixed forest-pseudoforest	$\tilde{O}_{f,p}(\lvert E\rvert + \lvert V\rvert\sqrt{\lvert V\rvert})$ ✓	$\tilde{O}((f+p)\lvert V\rvert\sqrt{f\lvert E\rvert})$ [GW88]
(Via matroid intersection)		
bipartite matching (combinatorial[12])	$\tilde{O}(\lvert E\rvert\sqrt{\lvert V\rvert})$	$O(\lvert E\rvert\sqrt{\lvert V\rvert})$ [HK73]
bipartite matching (continuous)	$\tilde{O}(\lvert E\rvert\sqrt{\lvert V\rvert})$ ✗	$\lvert E\rvert^{1+o(1)}$ [CKL+22]
graphic matroid intersection	$\tilde{O}(\lvert E\rvert\sqrt{\lvert V\rvert})$	$\tilde{O}(\lvert E\rvert\sqrt{\lvert V\rvert})$ [GX89]
simple job scheduling matroid intersection	$\tilde{O}(n\sqrt{r})$	$\tilde{O}(n\sqrt{r})$ [XG94]
convex transversal matroid [EF65] intersection	$\tilde{O}(\lvert V\rvert\sqrt{\mu})$	$\tilde{O}(\lvert V\rvert\sqrt{\mu})$ [XG94]
linear matroid intersection[10]	$\tilde{O}(n^{2.529}\sqrt{r})$ ✗	$\tilde{O}(nr^{\omega-1})$ [Har09]
colorful spanning tree	$\tilde{O}(\lvert E\rvert\sqrt{\lvert V\rvert})$	$\tilde{O}(\lvert E\rvert\sqrt{\lvert V\rvert})$ [GS85]
maximum forest with deadlines	$\tilde{O}(\lvert E\rvert\sqrt{\lvert V\rvert})$ ✓	(no prior work)