Subquadratic Weighted Matroid Intersection Under Rank Oracles

ISAAC 2022

Ta-Wei Tu

National Taiwan University

Summary

Results. The first subquadratic, i.e., $\tilde{O}(n^{7/4})$, rank-query algorithm for weighted matroid intersection.

Techniques. Efficient implementation of previous work.

1. Use binary searches in weight adjustments.
2. A subquadratic shortest-path algorithm in weighted exchange graphs.

Matroids

Definition

A matroid is a tuple $\mathcal{M} = (V, \mathcal{I})$ over a *ground set* V and a non-empty family of *independent sets* $\mathcal{I}$ such that

1. if $R \subseteq S$ and $S \in \mathcal{I}$, then $R \in \mathcal{I}$, and
2. if $R, S \in \mathcal{I}$ and $|R| < |S|$, then there exists $x \in S \setminus R$ such that $R \cup \{x\} \in \mathcal{I}$.

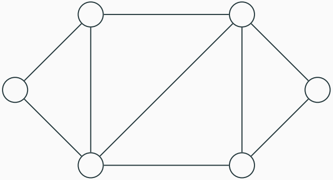
Matroids

Definition

A matroid is a tuple $\mathcal{M} = (V, \mathcal{I})$ over a *ground set* V and a non-empty family of *independent sets* $\mathcal{I}$ such that

1. if $R \subseteq S$ and $S \in \mathcal{I}$, then $R \in \mathcal{I}$, and
2. if $R, S \in \mathcal{I}$ and $|R| < |S|$, then there exists $x \in S \setminus R$ such that $R \cup \{x\} \in \mathcal{I}$.

$V =$ edges
$\mathcal{I} =$ acyclic subgraphs



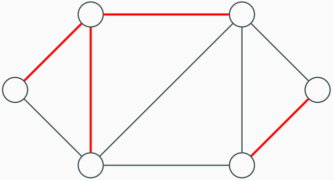
Matroids

Definition

A matroid is a tuple $\mathcal{M} = (V, \mathcal{I})$ over a *ground set* V and a non-empty family of *independent sets* $\mathcal{I}$ such that

1. if $R \subseteq S$ and $S \in \mathcal{I}$, then $R \in \mathcal{I}$, and
2. if $R, S \in \mathcal{I}$ and $|R| < |S|$, then there exists $x \in S \setminus R$ such that $R \cup \{x\} \in \mathcal{I}$.

$V =$ edges
$\mathcal{I} =$ acyclic subgraphs



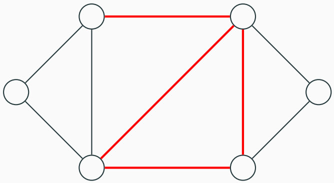
Matroids

Definition

A matroid is a tuple $\mathcal{M} = (V, \mathcal{I})$ over a *ground set* V and a non-empty family of *independent sets* $\mathcal{I}$ such that

1. if $R \subseteq S$ and $S \in \mathcal{I}$, then $R \in \mathcal{I}$, and
2. if $R, S \in \mathcal{I}$ and $|R| < |S|$, then there exists $x \in S \setminus R$ such that $R \cup \{x\} \in \mathcal{I}$.

$V =$ edges
$\mathcal{I} =$ acyclic subgraphs



Matroids

Definition

A matroid is a tuple $\mathcal{M} = (V, \mathcal{I})$ over a *ground set* V and a non-empty family of *independent sets* $\mathcal{I}$ such that

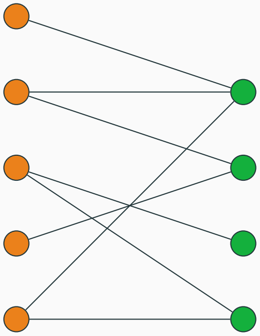
1. if $R \subseteq S$ and $S \in \mathcal{I}$, then $R \in \mathcal{I}$, and
2. if $R, S \in \mathcal{I}$ and $|R| < |S|$, then there exists $x \in S \setminus R$ such that $R \cup \{x\} \in \mathcal{I}$.

- Independence query: Is $S \in \mathcal{I}$?
- Rank query: What is rank(S)? ($\text{rank}(S) = \max_{R \in \mathcal{I}; R \subseteq S} |R|$)

Matroid Intersection

Given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$, $\mathcal{M}_2 = (V, \mathcal{I}_2)$, find the largest $S \in \mathcal{I}_1 \cap \mathcal{I}_2$.

- Bipartite matching
- Colorful spanning tree
- Disjoint spanning trees
- ...

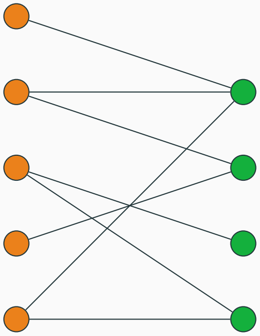


$\mathcal{M}_1 = \ell \in L$ has at most one edge

Matroid Intersection

Given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$, $\mathcal{M}_2 = (V, \mathcal{I}_2)$, find the largest $S \in \mathcal{I}_1 \cap \mathcal{I}_2$.

- Bipartite matching
- Colorful spanning tree
- Disjoint spanning trees
- ...



$\mathcal{M}_2 = r \in R$ has at most one edge

Weighted Matroid Intersection

Given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$, $\mathcal{M}_2 = (V, \mathcal{I}_2)$ with weights $w : V \to \mathbb{Z}$, find the $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ maximizing $w(S)$.

- (weighted) Bipartite matching
- (weighted) Colorful spanning tree
- (weighted) Disjoint spanning trees

Prior Work

’70s	Edmonds, Lawler, Aigner-Dowling	$O(n^3)$ indep
’86	Cunningham	$O(n^{5/2})$ indep
’15	Lee-Sidford-Wong	$\tilde{O}(n^2)$ rank
’15	Lee-Sidford-Wong	$\tilde{O}(n^2)$ indep
’19	Chakrabarty-Lee-Sidford-Singla-Wong	$\tilde{O}(n^2)$ indep
’19	Nguyễn	$\tilde{O}(n^2)$ indep

Prior Work

’70s	Edmonds, Lawler, Aigner-Dowling	$O(n^3)$ indep
’86	Cunningham	$O(n^{5/2})$ indep
’15	Lee-Sidford-Wong	$\tilde{O}(n^2)$ rank
’15	Lee-Sidford-Wong	$\tilde{O}(n^2)$ indep
’19	Chakrabarty-Lee-Sidford-Singla-Wong	$\tilde{O}(n^2)$ indep
’19	Nguyễn	$\tilde{O}(n^2)$ indep
’19	Chakrabarty-Lee-Sidford-Singla-Wong	$\tilde{O}(n^{3/2})$ rank

Prior Work

’70s	Edmonds, Lawler, Aigner-Dowling	$O(n^3)$ indep
’86	Cunningham	$O(n^{5/2})$ indep
’15	Lee-Sidford-Wong	$\tilde{O}(n^2)$ rank
’15	Lee-Sidford-Wong	$\tilde{O}(n^2)$ indep
’19	Chakrabarty-Lee-Sidford-Singla-Wong	$\tilde{O}(n^2)$ indep
’19	Nguyễn	$\tilde{O}(n^2)$ indep
’19	Chakrabarty-Lee-Sidford-Singla-Wong	$\tilde{O}(n^{3/2})$ rank
’19	Chakrabarty-Lee-Sidford-Singla-Wong	$\tilde{O}_\epsilon(n^{3/2})$ indep

Prior Work

’70s	Edmonds, Lawler, Aigner-Dowling	$O(n^3)$ indep
’86	Cunningham	$O(n^{5/2})$ indep
’15	Lee-Sidford-Wong	$\tilde{O}(n^2)$ rank
’15	Lee-Sidford-Wong	$\tilde{O}(n^2)$ indep
’19	Chakrabarty-Lee-Sidford-Singla-Wong	$\tilde{O}(n^2)$ indep
’19	Nguyễn	$\tilde{O}(n^2)$ indep
’19	Chakrabarty-Lee-Sidford-Singla-Wong	$\tilde{O}(n^{3/2})$ rank
’19	Chakrabarty-Lee-Sidford-Singla-Wong	$\tilde{O}_\epsilon(n^{3/2})$ indep
’21	Blikstad-v.d.Brand-Mukhopadhyay-Nanongkai	$\tilde{O}(n^{9/5})$ indep

Prior Work

’81	Frank	$O(n^3)$ indep
’95	Fujishige-Zhang, Shigeno-Iwata, Gabow-Xu	$\tilde{O}(n^{5/2})$ indep
’15	Lee-Sidford-Wong	$\tilde{O}(n^2)$ rank
’15	Lee-Sidford-Wong	$\tilde{O}(n^2)$ indep

Prior Work

’81	Frank	$O(n^3)$ indep
’95	Fujishige-Zhang, Shigeno-Iwata, Gabow-Xu	$\tilde{O}(n^{5/2})$ indep
’15	Lee-Sidford-Wong	$\tilde{O}(n^2)$ rank
’15	Lee-Sidford-Wong	$\tilde{O}(n^2)$ indep
’16	Chekuri-Quanrud	$\tilde{O}_\epsilon(n^2)$ indep

Prior Work

’81	Frank	$O(n^3)$ indep
’95	Fujishige-Zhang, Shigeno-Iwata, Gabow-Xu	$\tilde{O}(n^{5/2})$ indep
’15	Lee-Sidford-Wong	$\tilde{O}(n^2)$ rank
’15	Lee-Sidford-Wong	$\tilde{O}(n^2)$ indep
’16	Chekuri-Quanrud	$\tilde{O}_\epsilon(n^2)$ indep
’22	Tu	$\tilde{O}(n^{7/4})$ rank

Overview

1. Get a $(1-\epsilon)$-approximate solution in $\tilde{O}(\frac{n}{\epsilon})$ queries using weight adjustments.
2. Find the remaining $O(\epsilon n)$ augmenting paths by solving the shortest-path problem.

$\tilde{O}(n^{2-2\delta})$ shortest-path algorithm $\implies$ $\tilde{O}(n^{2-\delta})$ weighted matroid intersection algorithm

Shortest-Path Problem

We have an unknown directed bipartite graph with labels $w_1(v), w_2(v)$ on vertices $L \cup R$.

Shortest-Path Problem

We have an unknown directed bipartite graph with labels $w_1(v), w_2(v)$ on vertices $L \cup R$.

Non-negative edge weights:

- From $\ell \in L$ to $r \in R$: $w_1(\ell) - w_1(r)$.
- From $r \in R$ to $\ell \in L$: $w_2(\ell) - w_2(r)$.

Shortest-Path Problem

We have an unknown directed bipartite graph with labels $w_1(v), w_2(v)$ on vertices $L \cup R$.

Non-negative edge weights:

- From $\ell \in L$ to $r \in R$: $w_1(\ell) - w_1(r)$.
- From $r \in R$ to $\ell \in L$: $w_2(\ell) - w_2(r)$.

Query. Is there an edge from vertex y to vertex set X (or vice versa)?

Shortest-Path Problem

We have an unknown directed bipartite graph with labels $w_1(v), w_2(v)$ on vertices $L \cup R$.

Non-negative edge weights:

- From $\ell \in L$ to $r \in R$: $w_1(\ell) - w_1(r)$.
- From $r \in R$ to $\ell \in L$: $w_2(\ell) - w_2(r)$.

Query. Is there an edge from vertex y to vertex set X (or vice versa)?

Binary search

Can find an edge from y to $x \in X$ with minimum/maximum $w_1(x)$, $w_2(x)$, ... in $O(\log n)$ queries.

Shortest-Path Problem

We have an unknown directed bipartite graph with labels $w_1(v), w_2(v)$ on vertices $L \cup R$.

Non-negative edge weights:

- From $\ell \in L$ to $r \in R$: $w_1(\ell) - w_1(r)$.
- From $r \in R$ to $\ell \in L$: $w_2(\ell) - w_2(r)$.

Query. Is there an edge from vertex y to vertex set X (or vice versa)?

Binary search

Can find an edge from y to $x \in X$ with minimum/maximum $w_1(x)$, $w_2(x)$, … in $O(\log n)$ queries.

Goal. Find the shortest-path tree rooted at s.

Shortest-Path Algorithm

Dijkstra's Algorithm. Find the nearest unvisited vertex in each iteration by relaxing edges from visited vertices.

Shortest-Path Algorithm

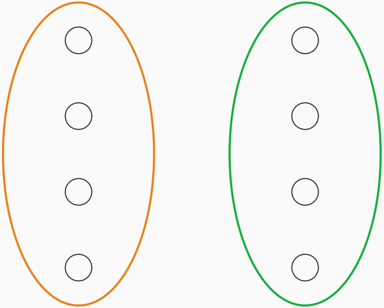
Dijkstra's Algorithm. Find the nearest unvisited vertex in each iteration by relaxing edges from visited vertices.

1. Recompute all distances

Shortest-Path Algorithm

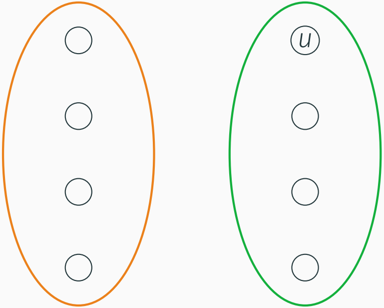
Dijkstra's Algorithm. Find the nearest unvisited vertex in each iteration by relaxing edges from visited vertices.

1. Recompute all distances



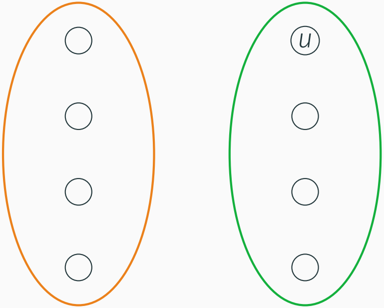
Dijkstra's Algorithm. Find the nearest unvisited vertex in each iteration by relaxing edges from visited vertices.

1. Recompute all distances



Dijkstra's Algorithm. Find the nearest unvisited vertex in each iteration by relaxing edges from visited vertices.

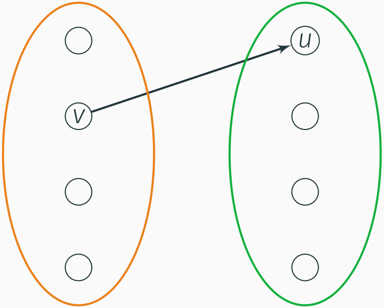
1. Recompute all distances



$$d(u) \leftarrow \min_{v:\text{visited}} d(v) + w_1(v) - w_1(u) = \left(\min_{v:\text{visited}} d(v) + w_1(v)\right) - w_1(u)$$

Dijkstra's Algorithm. Find the nearest unvisited vertex in each iteration by relaxing edges from visited vertices.

1. Recompute all distances

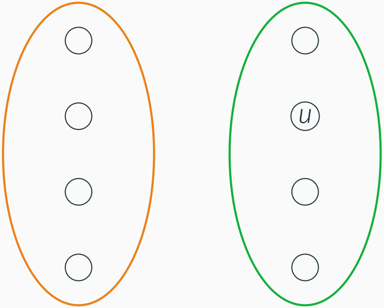


$$d(u) \leftarrow \min_{v:\text{visited}} d(v) + w_1(v) - w_1(u) = \left(\min_{v:\text{visited}} d(v) + w_1(v)\right) - w_1(u)$$

Shortest-Path Algorithm

Dijkstra's Algorithm. Find the nearest unvisited vertex in each iteration by relaxing edges from visited vertices.

1. Recompute all distances

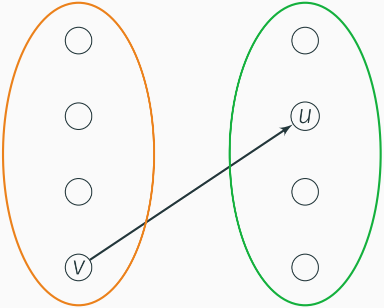


$$d(u) \leftarrow \min_{v:\text{visited}} d(v) + w_1(v) - w_1(u) = \left(\min_{v:\text{visited}} d(v) + w_1(v)\right) - w_1(u)$$

Shortest-Path Algorithm

Dijkstra's Algorithm. Find the nearest unvisited vertex in each iteration by relaxing edges from visited vertices.

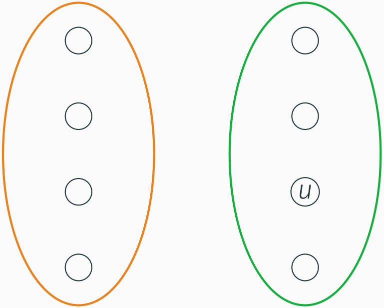
1. Recompute all distances



$$d(u) \leftarrow \min_{v:\text{visited}} d(v) + w_1(v) - w_1(u) = \left(\min_{v:\text{visited}} d(v) + w_1(v) \right) - w_1(u)$$

Dijkstra's Algorithm. Find the nearest unvisited vertex in each iteration by relaxing edges from visited vertices.

1. Recompute all distances

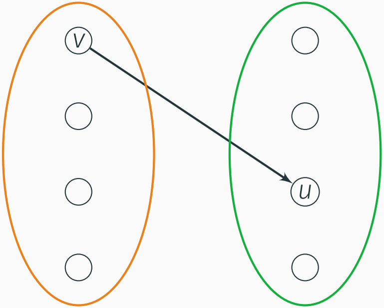


$$d(u) \leftarrow \min_{v:\text{visited}} d(v) + w_1(v) - w_1(u) = \left(\min_{v:\text{visited}} d(v) + w_1(v) \right) - w_1(u)$$

Shortest-Path Algorithm

Dijkstra's Algorithm. Find the nearest unvisited vertex in each iteration by relaxing edges from visited vertices.

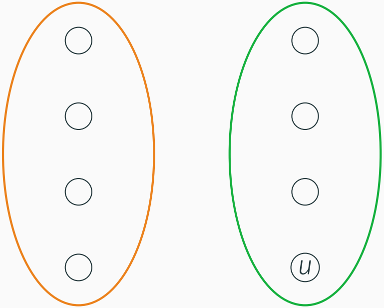
1. Recompute all distances



$$d(u) \leftarrow \min_{v:\text{visited}} d(v) + w_1(v) - w_1(u) = \left(\min_{v:\text{visited}} d(v) + w_1(v) \right) - w_1(u)$$

Dijkstra's Algorithm. Find the nearest unvisited vertex in each iteration by relaxing edges from visited vertices.

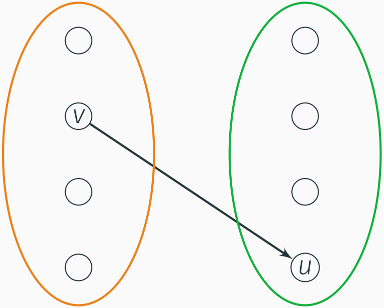
1. Recompute all distances



$$d(u) \leftarrow \min_{v:\text{visited}} d(v) + w_1(v) - w_1(u) = \left(\min_{v:\text{visited}} d(v) + w_1(v) \right) - w_1(u)$$

Dijkstra's Algorithm. Find the nearest unvisited vertex in each iteration by relaxing edges from visited vertices.

1. Recompute all distances



$$d(u) \leftarrow \min_{v\text{:visited}} d(v) + w_1(v) - w_1(u) = \left(\min_{v\text{:visited}} d(v) + w_1(v)\right) - w_1(u)$$

Shortest-Path Algorithm

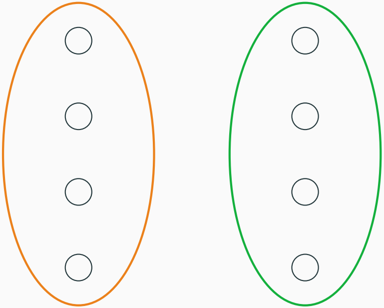
Dijkstra's Algorithm. Find the nearest unvisited vertex in each iteration by relaxing edges from visited vertices.

1. Recompute all distances
2. Relax the shortest edge from every visited vertex

Shortest-Path Algorithm

Dijkstra's Algorithm. Find the nearest unvisited vertex in each iteration by relaxing edges from visited vertices.

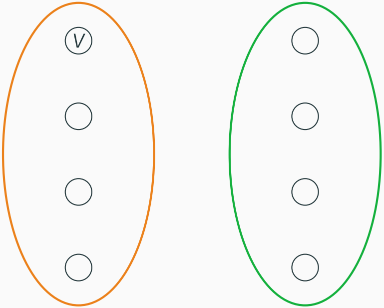
1. Recompute all distances
2. Relax the shortest edge from every visited vertex



Shortest-Path Algorithm

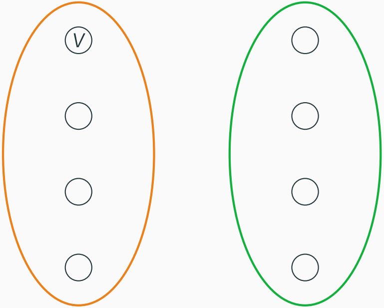
Dijkstra's Algorithm. Find the nearest unvisited vertex in each iteration by relaxing edges from visited vertices.

1. Recompute all distances
2. Relax the shortest edge from every visited vertex



Dijkstra's Algorithm. Find the nearest unvisited vertex in each iteration by relaxing edges from visited vertices.

1. Recompute all distances
2. Relax the shortest edge from every visited vertex

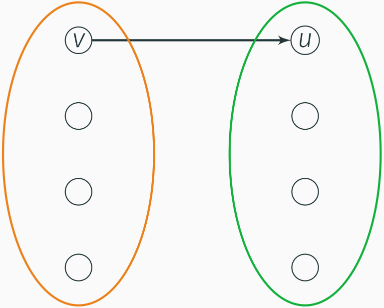


$$\min_{u:\text{unvisited}} d(v) + w_1(v) - w_1(u) = d(v) + w_1(v) - \max_{u:\text{unvisited}} w_1(u)$$

Shortest-Path Algorithm

Dijkstra's Algorithm. Find the nearest unvisited vertex in each iteration by relaxing edges from visited vertices.

1. Recompute all distances
2. Relax the shortest edge from every visited vertex

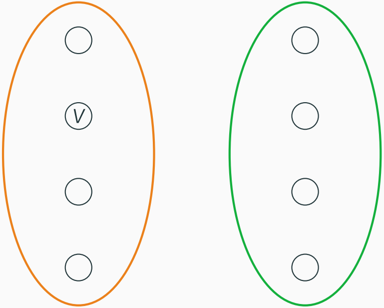


$$\min_{u:\text{unvisited}} d(v) + w_1(v) - w_1(u) = d(v) + w_1(v) - \max_{u:\text{unvisited}} w_1(u)$$

Shortest-Path Algorithm

Dijkstra's Algorithm. Find the nearest unvisited vertex in each iteration by relaxing edges from visited vertices.

1. Recompute all distances
2. Relax the shortest edge from every visited vertex

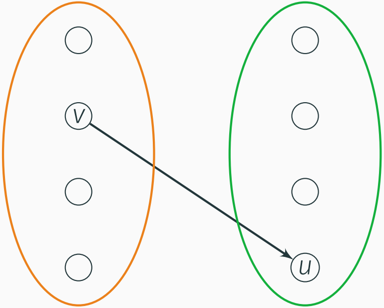


$$\min_{u:\text{unvisited}} d(v) + w_1(v) - w_1(u) = d(v) + w_1(v) - \max_{u:\text{unvisited}} w_1(u)$$

Shortest-Path Algorithm

Dijkstra's Algorithm. Find the nearest unvisited vertex in each iteration by relaxing edges from visited vertices.

1. Recompute all distances
2. Relax the shortest edge from every visited vertex

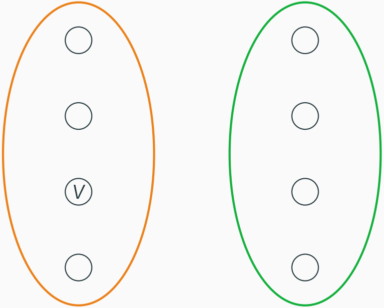


$$\min_{u:\text{unvisited}} d(v) + w_1(v) - w_1(u) = d(v) + w_1(v) - \max_{u:\text{unvisited}} w_1(u)$$

Shortest-Path Algorithm

Dijkstra's Algorithm. Find the nearest unvisited vertex in each iteration by relaxing edges from visited vertices.

1. Recompute all distances
2. Relax the shortest edge from every visited vertex

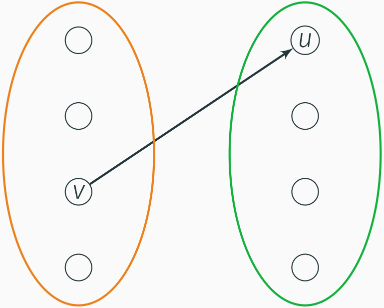


$$\min_{u:\text{unvisited}} d(v) + w_1(v) - w_1(u) = d(v) + w_1(v) - \max_{u:\text{unvisited}} w_1(u)$$

Shortest-Path Algorithm

Dijkstra's Algorithm. Find the nearest unvisited vertex in each iteration by relaxing edges from visited vertices.

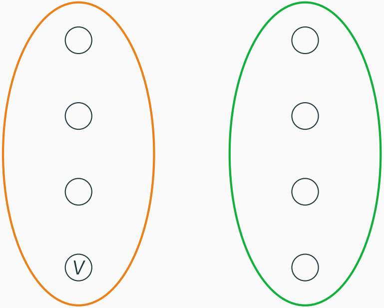
1. Recompute all distances
2. Relax the shortest edge from every visited vertex



$$\min_{u:\text{unvisited}} d(v) + w_1(v) - w_1(u) = d(v) + w_1(v) - \max_{u:\text{unvisited}} w_1(u)$$

Dijkstra's Algorithm. Find the nearest unvisited vertex in each iteration by relaxing edges from visited vertices.

1. Recompute all distances
2. Relax the shortest edge from every visited vertex

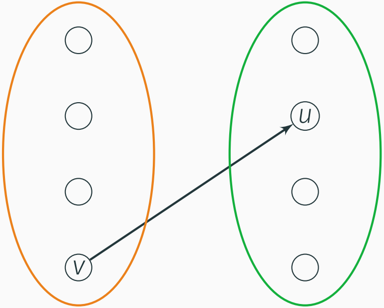


$$\min_{u:\text{unvisited}} d(v) + w_1(v) - w_1(u) = d(v) + w_1(v) - \max_{u:\text{unvisited}} w_1(u)$$

Shortest-Path Algorithm

Dijkstra's Algorithm. Find the nearest unvisited vertex in each iteration by relaxing edges from visited vertices.

1. Recompute all distances
2. Relax the shortest edge from every visited vertex



$$\min_{u:\text{unvisited}} d(v) + w_1(v) - w_1(u) = d(v) + w_1(v) - \max_{u:\text{unvisited}} w_1(u)$$

A Hybrid Approach

$F =$ visited vertices, $B \subseteq F =$ recently visited vertices

A Hybrid Approach

$F =$ visited vertices, $B \subseteq F =$ recently visited vertices

1. Make sure distances relaxed from $F \setminus B$ are correct
2. Relax the shortest edge from B

A Hybrid Approach

F = visited vertices, $B \subseteq F$ = recently visited vertices

1. Make sure distances relaxed from $F \setminus B$ are correct
2. Relax the shortest edge from B

Reset B and recompute all distances every $\sqrt{n}$ iterations

A Hybrid Approach

$F =$ visited vertices, $B \subseteq F =$ recently visited vertices

1. Make sure distances relaxed from $F \setminus B$ are correct
2. Relax the shortest edge from B

Reset B and recompute all distances every $\sqrt{n}$ iterations

Total number of queries: $\tilde{O}\left(\sqrt{n} \cdot n\right) + \tilde{O}\left(n \cdot \frac{n}{\sqrt{n}}\right) = \tilde{O}\left(n\sqrt{n}\right)$

Overview

1. Get a $(1-\epsilon)$-approximate solution in $\tilde{O}(\frac{n}{\epsilon})$ queries using weight adjustments.
2. Find the remaining $O(\epsilon n)$ augmenting paths by solving the shortest-path problem.

$\tilde{O}(n^{2-2\delta})$ shortest-path algorithm $\implies$ $\tilde{O}(n^{2-\delta})$ weighted matroid intersection algorithm

Overview

1. Get a $(1-\epsilon)$-approximate solution in $\tilde{O}(\frac{n}{\epsilon})$ queries using weight adjustments.
2. Find the remaining $O(\epsilon n)$ augmenting paths by solving the shortest-path problem.

$\tilde{O}(n^{3/2})$ shortest-path algorithm $\implies$ $\tilde{O}(n^{7/4})$ weighted matroid intersection algorithm

Open Problems

- Match the unweighted algorithm ($\tilde{O}(n^{7/4})$ versus $\tilde{O}(n^{3/2})$)? Perhaps via a $\tilde{O}(n)$ shortest-path algorithm.
- Improved strongly polynomial time algorithm?
- What about independence-query algorithms? No $\tilde{O}(n^2)$ "combinatorial" algorithm yet.
- Lower bounds?

Thanks for listening! Any question?