Greedy Algorithms, Matroids, and Parallel Complextiy

Joint work with Sumanta Ghosh and Roshan Raj

NISER Bhubaneswar

July 28, 2023

Outline

- Introduction to matroids and connection with greedy algorithms
- Search vs. decision: parallel complexity
- Matroid intersection: deterministic parallel search to decision reduction

Outline

- Introduction to matroids and connection with greedy algorithms
- Search vs. decision: parallel complexity
- Matroid intersection: deterministic parallel search to decision reduction

Takeaways:

- Isolation Lemma
- Succinct representation of all MSTs
- Succinct representation of all maximum weight perfect matchings

Maximum weight spanning tree

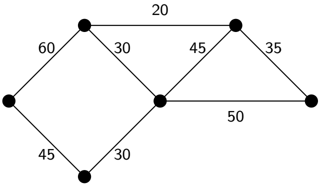
Kruskal's Algorithm

- Sort the edges in decreasing order of weights.
- Keep selecting edges which do not create a cycle (maintain a forest).

Maximum weight spanning tree

Kruskal's Algorithm

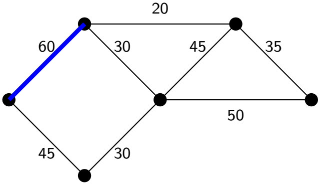
- Sort the edges in decreasing order of weights.
- Keep selecting edges which do not create a cycle (maintain a forest).



Maximum weight spanning tree

Kruskal's Algorithm

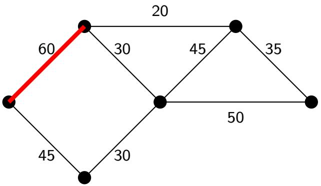
- Sort the edges in decreasing order of weights.
- Keep selecting edges which do not create a cycle (maintain a forest).



Maximum weight spanning tree

Kruskal's Algorithm

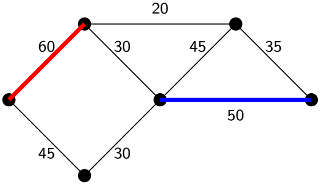
- Sort the edges in decreasing order of weights.
- Keep selecting edges which do not create a cycle (maintain a forest).



Maximum weight spanning tree

Kruskal's Algorithm

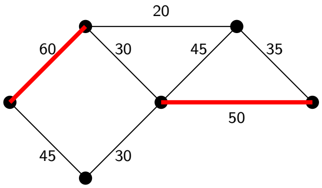
- Sort the edges in decreasing order of weights.
- Keep selecting edges which do not create a cycle (maintain a forest).



Maximum weight spanning tree

Kruskal's Algorithm

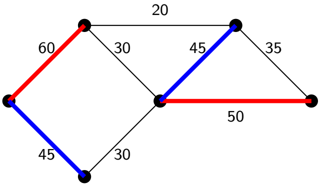
- Sort the edges in decreasing order of weights.
- Keep selecting edges which do not create a cycle (maintain a forest).



Maximum weight spanning tree

Kruskal's Algorithm

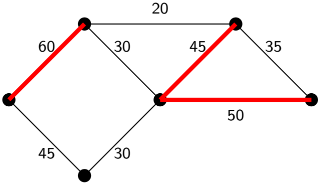
- Sort the edges in decreasing order of weights.
- Keep selecting edges which do not create a cycle (maintain a forest).



Maximum weight spanning tree

Kruskal's Algorithm

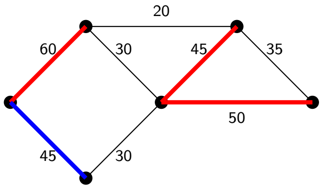
- Sort the edges in decreasing order of weights.
- Keep selecting edges which do not create a cycle (maintain a forest).



Maximum weight spanning tree

Kruskal's Algorithm

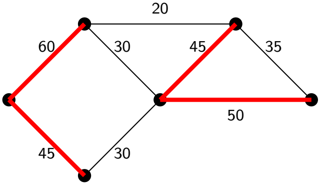
- Sort the edges in decreasing order of weights.
- Keep selecting edges which do not create a cycle (maintain a forest).



Maximum weight spanning tree

Kruskal's Algorithm

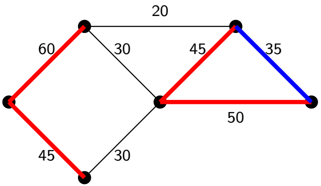
- Sort the edges in decreasing order of weights.
- Keep selecting edges which do not create a cycle (maintain a forest).



Maximum weight spanning tree

Kruskal's Algorithm

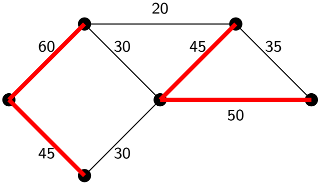
- Sort the edges in decreasing order of weights.
- Keep selecting edges which do not create a cycle (maintain a forest).



Maximum weight spanning tree

Kruskal's Algorithm

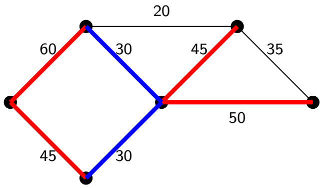
- Sort the edges in decreasing order of weights.
- Keep selecting edges which do not create a cycle (maintain a forest).



Maximum weight spanning tree

Kruskal's Algorithm

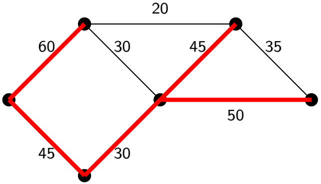
- Sort the edges in decreasing order of weights.
- Keep selecting edges which do not create a cycle (maintain a forest).



Maximum weight spanning tree

Kruskal's Algorithm

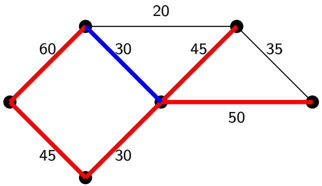
- Sort the edges in decreasing order of weights.
- Keep selecting edges which do not create a cycle (maintain a forest).



Maximum weight spanning tree

Kruskal's Algorithm

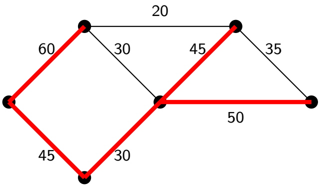
- Sort the edges in decreasing order of weights.
- Keep selecting edges which do not create a cycle (maintain a forest).



Maximum weight spanning tree

Kruskal's Algorithm

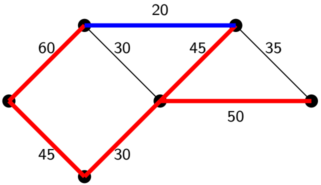
- Sort the edges in decreasing order of weights.
- Keep selecting edges which do not create a cycle (maintain a forest).



Maximum weight spanning tree

Kruskal's Algorithm

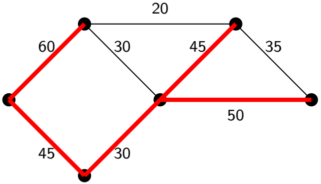
- Sort the edges in decreasing order of weights.
- Keep selecting edges which do not create a cycle (maintain a forest).



Maximum weight spanning tree

Kruskal's Algorithm

- Sort the edges in decreasing order of weights.
- Keep selecting edges which do not create a cycle (maintain a forest).



Job Scheduling

Max Profit Job Scheduling

- Unit time jobs with a profit, release time, and deadline.
- Find a schedulable set of jobs, maximizing profit.

Job Scheduling

Max Profit Job Scheduling

- Unit time jobs with a profit, release time, and deadline.
- Find a schedulable set of jobs, maximizing profit.

Job	P	Q	R	S	T	U	V
Release	2	2	3	4	2	3	1
Deadline	6	3	5	6	4	5	7
Profit	15	65	45	30	80	70	10

Job Scheduling

Max Profit Algorithm

- Sort the jobs in decreasing order of profit.

Job Scheduling

Max Profit Algorithm

- Sort the jobs in decreasing order of profit.
- Keep selecting jobs while maintaining schedulability.

Job Scheduling

Max Profit Algorithm

- Sort the jobs in decreasing order of profit.
- Keep selecting jobs while maintaining schedulability.

Job	P	Q	R	S	T	U	V
Release	2	2	3	4	2	3	1
Deadline	6	3	5	6	4	5	7
Profit	15	65	45	30	80	70	10

Job Scheduling

Max Profit Algorithm

- Sort the jobs in decreasing order of profit.
- Keep selecting jobs while maintaining schedulability.

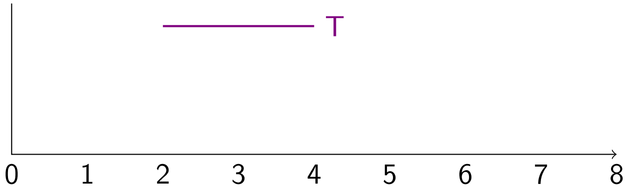
Job	P	Q	R	S	T	U	V
Release	2	2	3	4	2	3	1
Deadline	6	3	5	6	4	5	7
Profit	15	65	45	30	80	70	10

Job Scheduling

Max Profit Algorithm

- Sort the jobs in decreasing order of profit.
- Keep selecting jobs while maintaining schedulability.

Job	P	Q	R	S	T	U	V
Release	2	2	3	4	2	3	1
Deadline	6	3	5	6	4	5	7
Profit	15	65	45	30	**80**	70	10

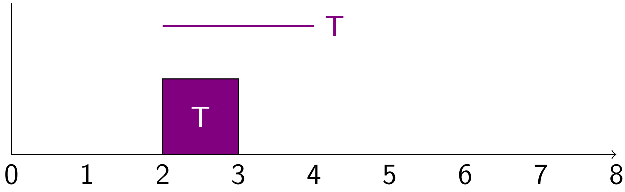


Job Scheduling

Max Profit Algorithm

- Sort the jobs in decreasing order of profit.
- Keep selecting jobs while maintaining schedulability.

Job	P	Q	R	S	T	U	V
Release	2	2	3	4	2	3	1
Deadline	6	3	5	6	4	5	7
Profit	15	65	45	30	**80**	70	10

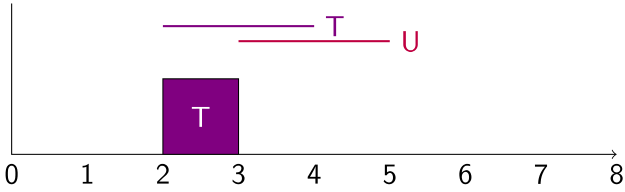


Job Scheduling

Max Profit Algorithm

- Sort the jobs in decreasing order of profit.
- Keep selecting jobs while maintaining schedulability.

Job	P	Q	R	S	T	U	V
Release	2	2	3	4	2	3	1
Deadline	6	3	5	6	4	5	7
Profit	15	65	45	30	**80**	**70**	10

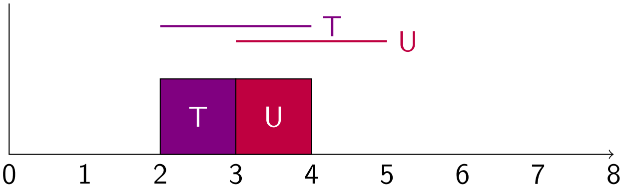


Job Scheduling

Max Profit Algorithm

- Sort the jobs in decreasing order of profit.
- Keep selecting jobs while maintaining schedulability.

Job	P	Q	R	S	T	U	V
Release	2	2	3	4	2	3	1
Deadline	6	3	5	6	4	5	7
Profit	15	65	45	30	**80**	**70**	10

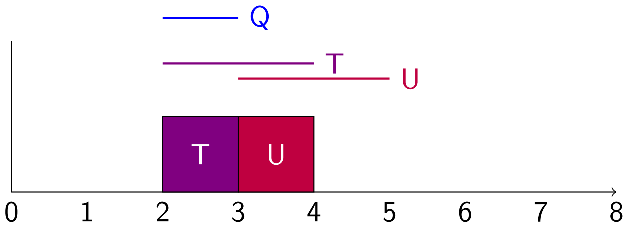


Job Scheduling

Max Profit Algorithm

- Sort the jobs in decreasing order of profit.
- Keep selecting jobs while maintaining schedulability.

Job	P	Q	R	S	T	U	V
Release	2	2	3	4	2	3	1
Deadline	6	3	5	6	4	5	7
Profit	15	**65**	45	30	**80**	**70**	10

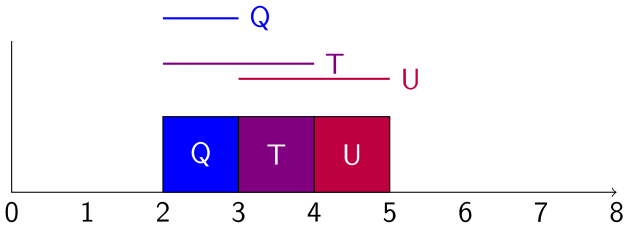


Job Scheduling

Max Profit Algorithm

- Sort the jobs in decreasing order of profit.
- Keep selecting jobs while maintaining schedulability.

Job	P	Q	R	S	T	U	V
Release	2	2	3	4	2	3	1
Deadline	6	3	5	6	4	5	7
Profit	15	**65**	45	30	**80**	**70**	10

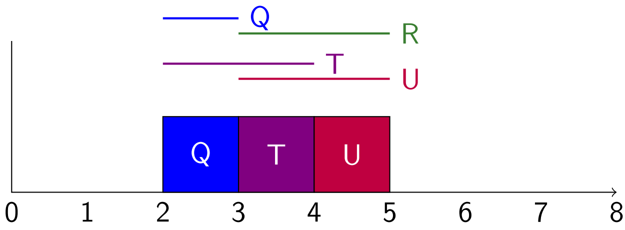


Job Scheduling

Max Profit Algorithm

- Sort the jobs in decreasing order of profit.
- Keep selecting jobs while maintaining schedulability.

Job	P	Q	R	S	T	U	V
Release	2	2	3	4	2	3	1
Deadline	6	3	5	6	4	5	7
Profit	15	**65**	**45**	30	**80**	**70**	10

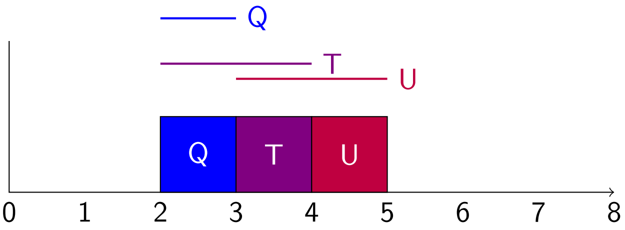


Job Scheduling

Max Profit Algorithm

- Sort the jobs in decreasing order of profit.
- Keep selecting jobs while maintaining schedulability.

Job	P	Q	R	S	T	U	V
Release	2	2	3	4	2	3	1
Deadline	6	3	5	6	4	5	7
Profit	15	**65**	45	30	**80**	**70**	10

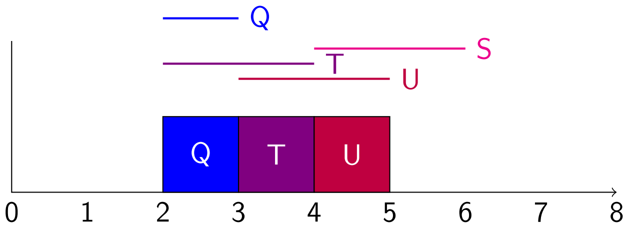


Job Scheduling

Max Profit Algorithm

- Sort the jobs in decreasing order of profit.
- Keep selecting jobs while maintaining schedulability.

Job	P	Q	R	S	T	U	V
Release	2	2	3	4	2	3	1
Deadline	6	3	5	6	4	5	7
Profit	15	**65**	45	**30**	**80**	**70**	10

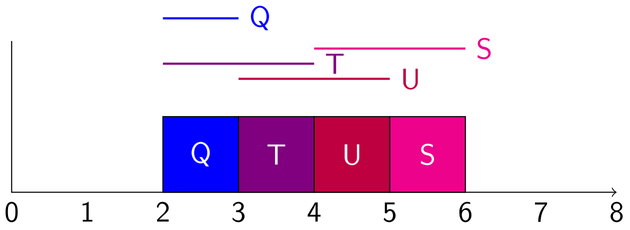


Job Scheduling

Max Profit Algorithm

- Sort the jobs in decreasing order of profit.
- Keep selecting jobs while maintaining schedulability.

Job	P	Q	R	S	T	U	V
Release	2	2	3	4	2	3	1
Deadline	6	3	5	6	4	5	7
Profit	15	**65**	45	**30**	**80**	**70**	10

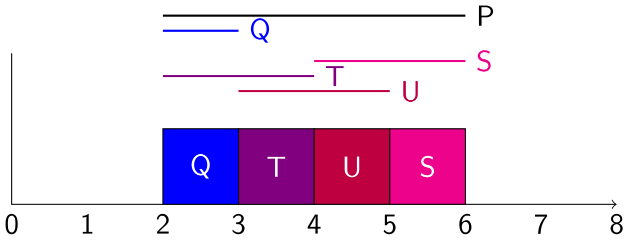


Job Scheduling

Max Profit Algorithm

- Sort the jobs in decreasing order of profit.
- Keep selecting jobs while maintaining schedulability.

Job	P	Q	R	S	T	U	V
Release	2	2	3	4	2	3	1
Deadline	6	3	5	6	4	5	7
Profit	**15**	**65**	45	**30**	**80**	**70**	10

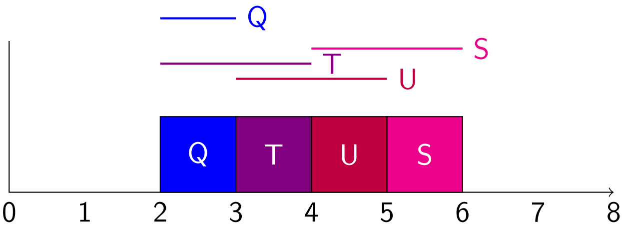


Job Scheduling

Max Profit Algorithm

- Sort the jobs in decreasing order of profit.
- Keep selecting jobs while maintaining schedulability.

Job	P	Q	R	S	T	U	V
Release	2	2	3	4	2	3	1
Deadline	6	3	5	6	4	5	7
Profit	15	**65**	45	**30**	**80**	**70**	10

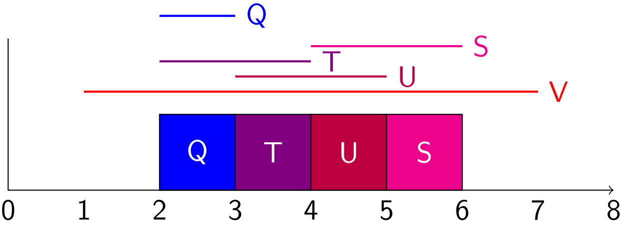


Job Scheduling

Max Profit Algorithm

- Sort the jobs in decreasing order of profit.
- Keep selecting jobs while maintaining schedulability.

Job	P	Q	R	S	T	U	V
Release	2	2	3	4	2	3	1
Deadline	6	3	5	6	4	5	7
Profit	15	**65**	45	**30**	**80**	**70**	**10**

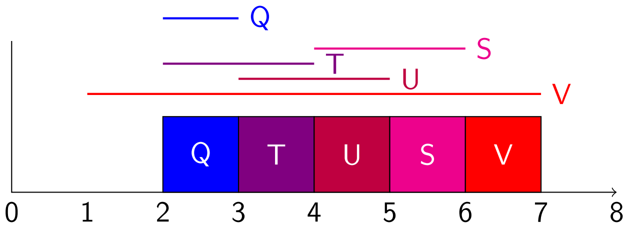


Job Scheduling

Max Profit Algorithm

- Sort the jobs in decreasing order of profit.
- Keep selecting jobs while maintaining schedulability.

Job	P	Q	R	S	T	U	V
Release	2	2	3	4	2	3	1
Deadline	6	3	5	6	4	5	7
Profit	15	**65**	45	**30**	**80**	**70**	**10**



Linear Indpendence

Max weight basis

- Set of vectors from $\mathbb{R}^n$, each with a weight.
- Find a subset of linearly independent vectors with maximum total weight.

Linear Indpendence

Max weight basis

- Set of vectors from $\mathbb{R}^n$, each with a weight.
- Find a subset of linearly independent vectors with maximum total weight.

Algorithm

- Sort the vectors in decreasing order of weights.
- Keep selecting vectors while maintaining linear independence.

Greedy Algorithms

- All three algorithms are the same at a high level.

Greedy Algorithms

- All three algorithms are the same at a high level.
- Correctness is not obvious.

Greedy Algorithms

- All three algorithms are the same at a high level.
- Correctness is not obvious.
- Is there a common reason why greedy works in these three settings?

Greedy Algorithms

- All three algorithms are the same at a high level.
- Correctness is not obvious.
- Is there a common reason why greedy works in these three settings?
- Is there something in common between
  - forests in a graph
  - schedulable subsets of jobs
  - linearly independent sets of vectors

Greedy Algorithms

- All three algorithms are the same at a high level.
- Correctness is not obvious.
- Is there a common reason why greedy works in these three settings?
- Is there something in common between
  - forests in a graph
  - schedulable subsets of jobs
  - linearly independent sets of vectors
- Extendibility: if the selected set is not the largest, then it can be *extended*.

Greedy Algorithms

- All three algorithms are the same at a high level.
- Correctness is not obvious.
- Is there a common reason why greedy works in these three settings?
- Is there something in common between
    - forests in a graph
    - schedulable subsets of jobs
    - linearly independent sets of vectors
- Extendibility: if the selected set is not the largest, then it can be *extended*.
  (without removing any elements)

Matroids

Definition (Matroid)

- E: Ground set

Matroids

Definition (Matroid)

- E: Ground set (edge set, set of jobs, set of vectors)

Matroids

Definition (Matroid)

- E: Ground set (edge set, set of jobs, set of vectors)
- $\mathcal{I}$: family of subsets of E (called independent sets)

Matroids

Definition (Matroid)

- E: Ground set (edge set, set of jobs, set of vectors)
- $\mathcal{I}$: family of subsets of E (called independent sets) (forests, schedulable sets of jobs, linearly independent sets of vectors)

Matroids

Definition (Matroid)

- E: Ground set (edge set, set of jobs, set of vectors)
- $\mathcal{I}$: family of subsets of E (called independent sets) (forests, schedulable sets of jobs, linearly independent sets of vectors)
- $(E, \mathcal{I})$ is a matroid if

Matroids

Definition (Matroid)

- E: Ground set (edge set, set of jobs, set of vectors)
- $\mathcal{I}$: family of subsets of E (called independent sets) (forests, schedulable sets of jobs, linearly independent sets of vectors)
- $(E, \mathcal{I})$ is a matroid if
  - $\emptyset \in \mathcal{I}$.

Matroids

Definition (Matroid)

- E: Ground set (edge set, set of jobs, set of vectors)
- $\mathcal{I}$: family of subsets of E (called independent sets) (forests, schedulable sets of jobs, linearly independent sets of vectors)
- $(E, \mathcal{I})$ is a matroid if
  - $\emptyset \in \mathcal{I}$.
  - $I \in \mathcal{I} \implies J \in \mathcal{I}$ for all $J \subseteq I$.

Matroids

Definition (Matroid)

- E: Ground set (edge set, set of jobs, set of vectors)
- $\mathcal{I}$: family of subsets of E (called independent sets) (forests, schedulable sets of jobs, linearly independent sets of vectors)
- $(E, \mathcal{I})$ is a matroid if
  - $\emptyset \in \mathcal{I}$.
  - $I \in \mathcal{I} \implies J \in \mathcal{I}$ for all $J \subseteq I$.
  - $A, B \in \mathcal{I}$ with $|A| < |B|$

Matroids

Definition (Matroid)

- E: Ground set (edge set, set of jobs, set of vectors)
- $\mathcal{I}$: family of subsets of E (called independent sets) (forests, schedulable sets of jobs, linearly independent sets of vectors)
- $(E, \mathcal{I})$ is a matroid if
  - $\emptyset \in \mathcal{I}$.
  - $I \in \mathcal{I} \implies J \in \mathcal{I}$ for all $J \subseteq I$.
  - $A, B \in \mathcal{I}$ with $|A| < |B|$ then $\exists a \in B \setminus A$ such that $A + a \in \mathcal{I}$.

Examples of Matroids

- Graphic matroids

Examples of Matroids

- Graphic matroids $E \leftarrow$ edge set, $\mathcal{I} \leftarrow$ family of all forests.

Examples of Matroids

- Graphic matroids $E \leftarrow$ edge set, $\mathcal{I} \leftarrow$ family of all forests.
- Transversal matroids

Examples of Matroids

- Graphic matroids $E \leftarrow$ edge set, $\mathcal{I} \leftarrow$ family of all forests.
- Transversal matroids
- Linear matroids

Examples of Matroids

- Graphic matroids $E \leftarrow$ edge set, $\mathcal{I} \leftarrow$ family of all forests.
- Transversal matroids
- Linear matroids
- Partition matroids

Examples of Matroids

- Graphic matroids $E \leftarrow$ edge set, $\mathcal{I} \leftarrow$ family of all forests.
- Transversal matroids
- Linear matroids
- Partition matroids
  $E \leftarrow$ set of students in a college,
  $\mathcal{I} \leftarrow$ teams that take
  at most 3 students from 4th year,
  at most 4 from 3rd year, ...

Examples of Matroids

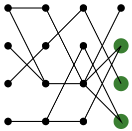
- Graphic matroids $E \leftarrow$ edge set, $\mathcal{I} \leftarrow$ family of all forests.
- Transversal matroids
- Linear matroids
- Partition matroids
  $E \leftarrow$ set of students in a college,
  $\mathcal{I} \leftarrow$ teams that take
  at most 3 students from 4th year,
  at most 4 from 3rd year, ...
- Gammoids

Examples of Matroids

- Graphic matroids $E \leftarrow$ edge set, $\mathcal{I} \leftarrow$ family of all forests.
- Transversal matroids
- Linear matroids
- Partition matroids
  $E \leftarrow$ set of students in a college,
  $\mathcal{I} \leftarrow$ teams that take
  at most 3 students from 4th year,
  at most 4 from 3rd year, ...
- Gammoids $E \leftarrow$ client nodes in a network

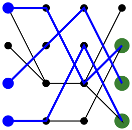
Examples of Matroids

- Graphic matroids $E \leftarrow$ edge set, $\mathcal{I} \leftarrow$ family of all forests.
- Transversal matroids
- Linear matroids
- Partition matroids
  $E \leftarrow$ set of students in a college,
  $\mathcal{I} \leftarrow$ teams that take
  at most 3 students from 4th year,
  at most 4 from 3rd year, ...
- Gammoids $E \leftarrow$ client nodes in a network
  $\mathcal{I} \leftarrow$ sets of clients with vertex-disjoint paths to servers



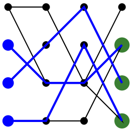
Examples of Matroids

- Graphic matroids $E \leftarrow$ edge set, $\mathcal{I} \leftarrow$ family of all forests.
- Transversal matroids
- Linear matroids
- Partition matroids
  $E \leftarrow$ set of students in a college,
  $\mathcal{I} \leftarrow$ teams that take
  at most 3 students from 4th year,
  at most 4 from 3rd year, ...
- Gammoids $E \leftarrow$ client nodes in a network
  $\mathcal{I} \leftarrow$ sets of clients with vertex-disjoint paths to servers



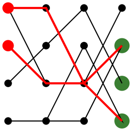
Examples of Matroids

- Graphic matroids $E \leftarrow$ edge set, $\mathcal{I} \leftarrow$ family of all forests.
- Transversal matroids
- Linear matroids
- Partition matroids
  $E \leftarrow$ set of students in a college,
  $\mathcal{I} \leftarrow$ teams that take
  at most 3 students from 4th year,
  at most 4 from 3rd year, ...
- Gammoids $E \leftarrow$ client nodes in a network
  $\mathcal{I} \leftarrow$ sets of clients with vertex-disjoint paths to servers



Examples of Matroids

- Graphic matroids $E \leftarrow$ edge set, $\mathcal{I} \leftarrow$ family of all forests.
- Transversal matroids
- Linear matroids
- Partition matroids
  $E \leftarrow$ set of students in a college,
  $\mathcal{I} \leftarrow$ teams that take
  at most 3 students from 4th year,
  at most 4 from 3rd year, ...
- Gammoids $E \leftarrow$ client nodes in a network
  $\mathcal{I} \leftarrow$ sets of clients with vertex-disjoint paths to servers



Matroids in Computer Science

- Combinatorial optimization
- Game theory
- Online algorithms
- Algebraic problems

Matroid Intersection

Problem

- Given two matroids on the same ground set,
- Find the largest size (weight) *common independent set*.

Matroid Intersection

Problem

- Given two matroids on the same ground set,
- Find the largest size (weight) *common independent set*.

Example I:

- Graph with Colored edges

Matroid Intersection

Problem

- Given two matroids on the same ground set,
- Find the largest size (weight) *common independent set*.

Example I:

- Graph with Colored edges
  largest forest with ≤ 2 red edges, ≤ 2 blue edges, ≤ 1 green edges ...

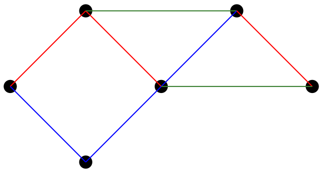
Matroid Intersection

Problem

- Given two matroids on the same ground set,
- Find the largest size (weight) *common independent set*.

Example I:

- Graph with Colored edges
  largest forest with ≤ 2 red edges, ≤ 2 blue edges, ≤ 1 green edges ...
  Graphic Matroid $\cap$ Partition Matroid



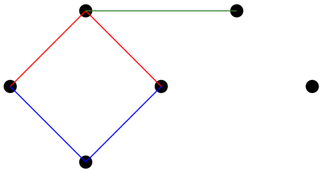
Matroid Intersection

Problem

- Given two matroids on the same ground set,
- Find the largest size (weight) *common independent set*.

Example I:

- Graph with Colored edges
  largest forest with ≤ 2 red edges, ≤ 2 blue edges, ≤ 1 green edges ...
  Graphic Matroid $\cap$ Partition Matroid



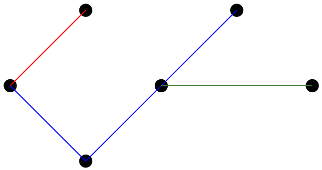
Matroid Intersection

Problem

- Given two matroids on the same ground set,
- Find the largest size (weight) *common independent set*.

Example I:

- Graph with Colored edges
  largest forest with ≤ 2 red edges, ≤ 2 blue edges, ≤ 1 green edges ...
  Graphic Matroid $\cap$ Partition Matroid



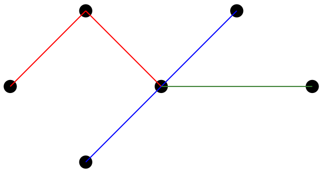
Matroid Intersection

Problem

- Given two matroids on the same ground set,
- Find the largest size (weight) *common independent set*.

Example I:

- Graph with Colored edges
  largest forest with ≤ 2 red edges, ≤ 2 blue edges, ≤ 1 green edges ...
  Graphic Matroid $\cap$ Partition Matroid



Matroid Intersection

Problem

- Given two matroids on the same ground set,
- Find the largest size (weight) *common independent set*.

Example II:

- Bipartite matching

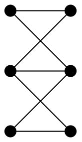
Matroid Intersection

Problem

- Given two matroids on the same ground set,
- Find the largest size (weight) *common independent set*.

Example II:

- Bipartite matching (any vertex having at most one edge)



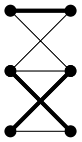
Matroid Intersection

Problem

- Given two matroids on the same ground set,
- Find the largest size (weight) *common independent set*.

Example II:

- Bipartite matching (any vertex having at most one edge)



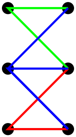
Matroid Intersection

Problem

- Given two matroids on the same ground set,
- Find the largest size (weight) *common independent set*.

Example II:

- Bipartite matching (any vertex having at most one edge)



Partition Matroid ∩ Partition Matroid.

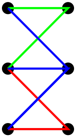
Matroid Intersection

Problem

- Given two matroids on the same ground set,
- Find the largest size (weight) *common independent set*.

Example II:

- Bipartite matching (any vertex having at most one edge)



Partition Matroid ∩ Partition Matroid.

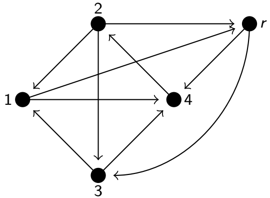
Matroid Intersection

Problem

- Given two matroids on the same ground set,
- Find the largest size (weight) *common independent set*.

Example III:

- r-Arborescences in a directed graph



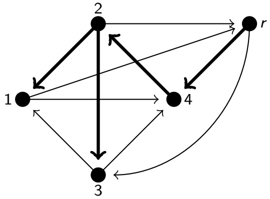
Matroid Intersection

Problem

- Given two matroids on the same ground set,
- Find the largest size (weight) *common independent set*.

Example III:

- r-Arborescences in a directed graph



Matroid Intersection

Problem

- Given two matroids on the same ground set,
- Find the largest size (weight) *common independent set*.

Example III:

- r-Arborescences in a directed graph

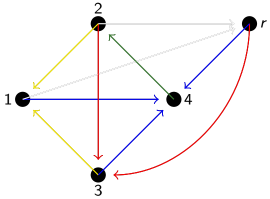


Figure: Graphic Matroid ∩ Partition Matroid

Matroid Intersection

Problem

- Given two matroids on the same ground set,
- Find the largest size (weight) *common independent set*.

Example III:

- r-Arborescences in a directed graph

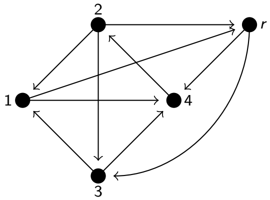


Figure: Graphic Matroid $\cap$ Partition Matroid

- At most 1 incoming edge at each vertex.

Matroid Intersection

Problem

- Given two matroids on the same ground set,
- Find the largest size (weight) *common independent set*.

Examples:

- Rainbow spanning tree
- Bipartite matching
- r-Arborescences in a directed graph

Matroid Intersection

Problem

- Given two matroids on the same ground set,
- Find the largest size (weight) *common independent set*.

Examples:

- Rainbow spanning tree
- Bipartite matching
- r-Arborescences in a directed graph
- Finding two disjoint spanning trees (Homework)

Search vs. Decision

Search vs. Decision

Satisfiability

- Decision: Given a Boolean formula, is there a satisfying assignment?
- Search: Find a satisfying assignment, if one exists.

Search vs. Decision

Satisfiability

- Decision: Given a Boolean formula, is there a satisfying assignment?
- Search: Find a satisfying assignment, if one exists.

Matching

- Decision: Is there a perfect matching in a given graph?
- Search: Find a perfect matching, if one exists.

Search vs. Decision

Satisfiability

- Decision: Given a Boolean formula, is there a satisfying assignment?
- Search: Find a satisfying assignment, if one exists.

Matching

- Decision: Is there a perfect matching in a given graph?
- Search: Find a perfect matching, if one exists.

- Decision is as easy as Search.

Search vs. Decision

Satisfiability

- Decision: Given a Boolean formula, is there a satisfying assignment?
- Search: Find a satisfying assignment, if one exists.

Matching

- Decision: Is there a perfect matching in a given graph?
- Search: Find a perfect matching, if one exists.

- Decision is as easy as Search.
- Is Search as easy as Decision?

Search to Decision Reduction

Satisfiability: finding a satisfying assignment for $\varphi(x_1, x_2, \ldots, x_n)$

Search to Decision Reduction

Satisfiability: finding a satisfying assignment for $\varphi(x_1, x_2, \ldots, x_n)$

- is $\varphi(x_1 = \mathit{true}, x_2, \ldots, x_n)$ Satisfiable?

Search to Decision Reduction

Satisfiability: finding a satisfying assignment for $\varphi(x_1, x_2, \ldots, x_n)$

- is $\varphi(x_1 = true, x_2, \ldots, x_n)$ Satisfiable?
  - If yes, set $x_1 = true$ and continue.

Search to Decision Reduction

Satisfiability: finding a satisfying assignment for $\varphi(x_1, x_2, \ldots, x_n)$

- is $\varphi(x_1 = true, x_2, \ldots, x_n)$ Satisfiable?
  - If yes, set $x_1 = true$ and continue.
  - If no, set $x_1 = false$ and continue.

Search to Decision Reduction

Satisfiability: finding a satisfying assignment for $\varphi(x_1, x_2, \ldots, x_n)$

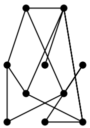
- is $\varphi(x_1 = \mathit{true}, x_2, \ldots, x_n)$ Satisfiable?
  - If yes, set $x_1 = \mathit{true}$ and continue.
  - If no, set $x_1 = \mathit{false}$ and continue.
- Repeat for each variable one by one to get a satisfying assignment.

Search to Decision Reduction

Satisfiability: finding a satisfying assignment for $\varphi(x_1, x_2, \ldots, x_n)$

- is $\varphi(x_1 = true, x_2, \ldots, x_n)$ Satisfiable?
  - If yes, set $x_1 = true$ and continue.
  - If no, set $x_1 = false$ and continue.
- Repeat for each variable one by one to get a satisfying assignment.

Matching: finding perfect matching in a graph G.



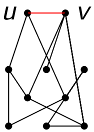
Search to Decision Reduction

Satisfiability: finding a satisfying assignment for $\varphi(x_1, x_2, \ldots, x_n)$

- is $\varphi(x_1 = true, x_2, \ldots, x_n)$ Satisfiable?
  - If yes, set $x_1 = true$ and continue.
  - If no, set $x_1 = false$ and continue.
- Repeat for each variable one by one to get a satisfying assignment.

Matching: finding perfect matching in a graph G.

- Pick an edge $e = (u, v)$.

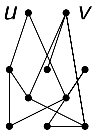


Search to Decision Reduction

Satisfiability: finding a satisfying assignment for $\varphi(x_1, x_2, \ldots, x_n)$

- is $\varphi(x_1 = \mathit{true}, x_2, \ldots, x_n)$ Satisfiable?
  - If yes, set $x_1 = \mathit{true}$ and continue.
  - If no, set $x_1 = \mathit{false}$ and continue.
- Repeat for each variable one by one to get a satisfying assignment.

Matching: finding perfect matching in a graph G.



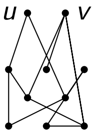
- Pick an edge $e = (u, v)$.
- Does $G - e$ have a perfect matching ?

Search to Decision Reduction

Satisfiability: finding a satisfying assignment for $\varphi(x_1, x_2, \ldots, x_n)$

- is $\varphi(x_1 = \mathit{true}, x_2, \ldots, x_n)$ Satisfiable?
  - If yes, set $x_1 = \mathit{true}$ and continue.
  - If no, set $x_1 = \mathit{false}$ and continue.
- Repeat for each variable one by one to get a satisfying assignment.

Matching: finding perfect matching in a graph G.



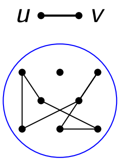
- Pick an edge $e = (u, v)$.
- Does $G - e$ have a perfect matching ?
  - If yes, delete e and continue.

Search to Decision Reduction

Satisfiability: finding a satisfying assignment for $\varphi(x_1, x_2, \ldots, x_n)$

- is $\varphi(x_1 = true, x_2, \ldots, x_n)$ Satisfiable?
  - If yes, set $x_1 = true$ and continue.
  - If no, set $x_1 = false$ and continue.
- Repeat for each variable one by one to get a satisfying assignment.

Matching: finding perfect matching in a graph G.



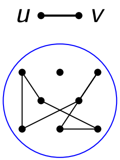
- Pick an edge $e = (u, v)$.
- Does $G - e$ have a perfect matching ?
  - If yes, delete e and continue.
  - If no, include e in the perfect matching and continue with $G - u - v$.

Search to Decision Reduction

Satisfiability: finding a satisfying assignment for $\varphi(x_1, x_2, \ldots, x_n)$

- is $\varphi(x_1 = \mathit{true}, x_2, \ldots, x_n)$ Satisfiable?
  - If yes, set $x_1 = \mathit{true}$ and continue.
  - If no, set $x_1 = \mathit{false}$ and continue.
- Repeat for each variable one by one to get a satisfying assignment.

Matching: finding perfect matching in a graph G.



- Pick an edge $e = (u, v)$.
- Does $G - e$ have a perfect matching ?
  - If yes, delete e and continue.
  - If no, include e in the perfect matching and continue with $G - u - v$.
- Keep repeating to get a perfect matching.

Search to Decision Reduction

- Search can be done using n decision queries.

Search to Decision Reduction

- Search can be done using n decision queries.
- These decision queries are adaptive.

Search to Decision Reduction

- Search can be done using n decision queries.
- These decision queries are adaptive.
- Is there a parallel search-to-decision reduction?

Search to Decision Reduction

- Search can be done using n decision queries.
- These decision queries are adaptive.
- Is there a parallel search-to-decision reduction?
  - If we are allowed poly(n) decision queries in parallel
  - can we do search, say, in $O(\sqrt{n})$ (or $O(\log^c n)$) rounds?

Search to Decision Reduction

- Search can be done using n decision queries.
- These decision queries are adaptive.
- Is there a parallel search-to-decision reduction?
  - If we are allowed poly(n) decision queries in parallel
  - can we do search, say, in $O(\sqrt{n})$ (or $O(\log^c n)$) rounds?
- [Karp, Upfal, Wigderson 1985] studied this question, motivated by the parallel complexity status of matching and matroid intersection.

Parallel Complexity of Matroid Intersection

- [Lovász 1979] gave randomized parallel algorithms for the Decision version of matching and linear matroid intersection.

Parallel Complexity of Matroid Intersection

- [Lovász 1979] gave randomized parallel algorithms for the Decision version of matching and linear matroid intersection.
- Based on determinant computation.

Parallel Complexity of Matroid Intersection

- [Lovász 1979] gave randomized parallel algorithms for the Decision version of matching and linear matroid intersection.
- Based on determinant computation.
- Efficient parallel algorithm (NC): $O(\log^c n)$ time on poly(n) parallel processors.

Parallel Complexity of Matroid Intersection

- [Lovász 1979] gave randomized parallel algorithms for the Decision version of matching and linear matroid intersection.
- Based on determinant computation.
- Efficient parallel algorithm (NC): $O(\log^c n)$ time on poly(n) parallel processors.
- Did not imply any parallel algorithm for Search version.

Parallel Complexity of Matroid Intersection

- [KUW86, MVV87]: efficient parallel randomized reduction (RNC) from search to weighted-decision.

Parallel Complexity of Matroid Intersection

- [KUW86, MVV87]: efficient parallel randomized reduction (RNC) from search to weighted-decision.
- Weighted-decision: given a graph with edge weights, is there a matching with weight at least W?

Parallel Complexity of Matroid Intersection

- [KUW86, MVV87]: efficient parallel randomized reduction (RNC) from search to weighted-decision.
- Weighted-decision: given a graph with edge weights, is there a matching with weight at least W?
- Implied a randomized parallel algorithm for the search version.

Parallel Complexity of Matroid Intersection

- [KUW86, MVV87]: efficient parallel randomized reduction (RNC) from search to weighted-decision.
- Weighted-decision: given a graph with edge weights, is there a matching with weight at least W?
- Implied a randomized parallel algorithm for the search version.

Open questions:

Parallel Complexity of Matroid Intersection

- [KUW86, MVV87]: efficient parallel randomized reduction (RNC) from search to weighted-decision.
- Weighted-decision: given a graph with edge weights, is there a matching with weight at least W?
- Implied a randomized parallel algorithm for the search version.

Open questions:

- Is there a deterministic parallel (NC) algorithm for any version?

Parallel Complexity of Matroid Intersection

- [KUW86, MVV87]: efficient parallel randomized reduction (RNC) from search to weighted-decision.
- Weighted-decision: given a graph with edge weights, is there a matching with weight at least W?
- Implied a randomized parallel algorithm for the search version.

Open questions:

- Is there a deterministic parallel (NC) algorithm for any version?
  still open.

Parallel Complexity of Matroid Intersection

- [KUW86, MVV87]: efficient parallel randomized reduction (RNC) from search to weighted-decision.
- Weighted-decision: given a graph with edge weights, is there a matching with weight at least W?
- Implied a randomized parallel algorithm for the search version.

Open questions:

- Is there a deterministic parallel (NC) algorithm for any version?
  still open.
- Is there a deterministic parallel (NC) reduction from search to decision (or weighted-decision)?

Parallel Complexity of Matroid Intersection

- [KUW86, MVV87]: efficient parallel randomized reduction (RNC) from search to weighted-decision.
- Weighted-decision: given a graph with edge weights, is there a matching with weight at least W?
- Implied a randomized parallel algorithm for the search version.

Open questions:

- Is there a deterministic parallel (NC) algorithm for any version?
  still open.
- Is there a deterministic parallel (NC) reduction from search to decision (or weighted-decision)?
  Some exciting progress recently.

Search to weighted-decision

Search to weighted-decision deterministic parallel reduction

- [FGT16, GG17] Bipartite Matching

Search to weighted-decision

Search to weighted-decision deterministic parallel reduction

- [FGT16, GG17] Bipartite Matching
- [AV20] General Matching

Search to weighted-decision

Search to weighted-decision deterministic parallel reduction

- [FGT16, GG17] Bipartite Matching
- [AV20] General Matching
- This work Matroid Intersection

Search to decision: unique solution

Search to decision: unique solution

- Suppose a Boolean formula φ has exactly one satisfying assignment.

Search to decision: unique solution

- Suppose a Boolean formula φ has exactly one satisfying assignment.
- Can we find it using non-adaptive decision queries?

Search to decision: unique solution

- Suppose a Boolean formula φ has exactly one satisfying assignment.
- Can we find it using non-adaptive decision queries?

Finding unique assignment

For each variable x_i, in parallel:
is $\varphi(\ldots, x_i = \mathit{true}, \ldots)$ satisfiable?

Search to decision: unique solution

- Suppose a Boolean formula φ has exactly one satisfying assignment.
- Can we find it using non-adaptive decision queries?

Finding unique assignment

For each variable x_i, in parallel:
is $\varphi(\dots, x_i = \mathit{true}, \dots)$ satisfiable?

- If yes, set $x_i = \mathit{true}$.

Search to decision: unique solution

- Suppose a Boolean formula φ has exactly one satisfying assignment.
- Can we find it using non-adaptive decision queries?

Finding unique assignment

For each variable x_i, in parallel:
is $\varphi(\dots, x_i = \mathit{true}, \dots)$ satisfiable?

- If yes, set $x_i = \mathit{true}$.
- If no, set $x_i = \mathit{false}$.

Search to decision: unique solution

- If a graph has a only one perfect matching, then can find it using non-adaptive decision queries.

Search to decision: unique solution

- If a graph has a only one perfect matching, then can find it using non-adaptive decision queries.

Finding unique perfect matching

For each edge e, in parallel:
  does $G - e$ have a perfect matching?

Search to decision: unique solution

- If a graph has a only one perfect matching, then can find it using non-adaptive decision queries.

Finding unique perfect matching

For each edge e, in parallel:
does $G - e$ have a perfect matching?

- If no, select e.

Search to decision: unique solution

- If a graph has a only one perfect matching, then can find it using non-adaptive decision queries.

Finding unique perfect matching

For each edge e, in parallel:
does $G - e$ have a perfect matching?

- If no, select e.
- If yes, discard e.

Search to weighted-decision: Randomized Reduction

- [MVV87] Isolating weight assignment: a weight assignment on the edges such that there is only one maximum weight perfect matching.

Search to weighted-decision: Randomized Reduction

- [MVV87] Isolating weight assignment: a weight assignment on the edges such that there is only one maximum weight perfect matching.
- Can we find unique max weight PM using non-adaptive weighted-decision queries?

Search to weighted-decision: Randomized Reduction

- [MVV87] Isolating weight assignment: a weight assignment on the edges such that there is only one maximum weight perfect matching.
- Can we find unique max weight PM using non-adaptive weighted-decision queries?

Finding unique max weight perfect matching

- Find w^* using weighted-decision queries.

Search to weighted-decision: Randomized Reduction

- [MVV87] Isolating weight assignment: a weight assignment on the edges such that there is only one maximum weight perfect matching.
- Can we find unique max weight PM using non-adaptive weighted-decision queries?

Finding unique max weight perfect matching

- Find w^* using weighted-decision queries.
- For each edge e, in parallel:
  does $G - e$ have a perfect matching with weight $\geq w^*$?

Search to weighted-decision: Randomized Reduction

- [MVV87] Isolating weight assignment: a weight assignment on the edges such that there is only one maximum weight perfect matching.
- Can we find unique max weight PM using non-adaptive weighted-decision queries?

Finding unique max weight perfect matching

- Find w^* using weighted-decision queries.
- For each edge e, in parallel:
  does $G - e$ have a perfect matching with weight $\geq w^*$?
  - If no, select e.

Search to weighted-decision: Randomized Reduction

- [MVV87] Isolating weight assignment: a weight assignment on the edges such that there is only one maximum weight perfect matching.
- Can we find unique max weight PM using non-adaptive weighted-decision queries?

Finding unique max weight perfect matching

- Find w^* using weighted-decision queries.
- For each edge e, in parallel:
  does $G - e$ have a perfect matching with weight $\geq w^*$?
  - If no, select e.
  - If yes, discard e.

Search to weighted-decision: Randomized Reduction

- But how do we guarantee unique max weight perfect matching?

Search to weighted-decision: Randomized Reduction

- But how do we guarantee unique max weight perfect matching?
- weights are poly bounded, but number of PMs is exponential.

Search to weighted-decision: Randomized Reduction

- But how do we guarantee unique max weight perfect matching?
- weights are poly bounded, but number of PMs is exponential.

Isolation Lemma [MVV87]

Wake up!

Search to weighted-decision: Randomized Reduction

- But how do we guarantee unique max weight perfect matching?
- weights are poly bounded, but number of PMs is exponential.

Isolation Lemma [MVV87]

Assign each edge a random weight independently from $\{0, 1, 2 \ldots, 2m\}$. Then,

Search to weighted-decision: Randomized Reduction

- But how do we guarantee unique max weight perfect matching?
- weights are poly bounded, but number of PMs is exponential.

Isolation Lemma [MVV87]

Assign each edge a random weight independently from $\{0, 1, 2 \ldots, 2m\}$. Then,

$$\Pr\{\text{there is only one max weight PM}\} \geq 1/2.$$

Search to weighted-decision: Randomized Reduction

- But how do we guarantee unique max weight perfect matching?
- weights are poly bounded, but number of PMs is exponential.

Isolation Lemma [MVV87]

Assign each edge a random weight independently from $\{0, 1, 2 \ldots, 2m\}$. Then,

$$\Pr\{\text{there is only one max weight PM}\} \geq 1/2.$$

- Works for an arbitrary family of sets.

Search to weighted-decision: Randomized Reduction

- But how do we guarantee unique max weight perfect matching?
- weights are poly bounded, but number of PMs is exponential.

Isolation Lemma [MVV87]

Assign each edge a random weight independently from $\{0, 1, 2 \ldots, 2m\}$. Then,

$$\Pr\{\text{there is only one max weight PM}\} \geq 1/2.$$

- Works for an arbitrary family of sets.
- Derandomizing Isolation Lemma remains an open question.

Search to weighted-decision: Deterministic Reduction

Main technical result

- Given two matroids,
- construct a weight assignment such that there is only one max weight common base (rainbow spanning tree)
- using $O(\log^2 n)$ rounds of weighted-decision queries.

Search to weighted-decision: Deterministic Reduction

Main technical result

- Given two matroids,
- construct a weight assignment such that there is only one max weight common base (rainbow spanning tree)
- using $O(\log^2 n)$ rounds of weighted-decision queries.

Algorithm at a high level

$S \leftarrow$ set of all common bases
while $(|S| > 1)$
  Update w to enforce some tie breaks in S.
  $S \leftarrow$ set of max weight common bases.

Search to weighted-decision: Deterministic Reduction

Main technical result

- Given two matroids,
- construct a weight assignment such that there is only one max weight common base (rainbow spanning tree)
- using $O(\log^2 n)$ rounds of weighted-decision queries.

Algorithm at a high level

$S \leftarrow$ set of all common bases
while $(|S| > 1)$
  Update w to enforce some tie breaks in S.
  $S \leftarrow$ set of max weight common bases.

- In $O(\log n)$ rounds, unique max weight common base.

Search to weighted-decision: Deterministic Reduction

Main technical result

- Given two matroids,
- construct a weight assignment such that there is only one max weight common base (rainbow spanning tree)
- using $O(\log^2 n)$ rounds of weighted-decision queries.

Algorithm at a high level

$S \leftarrow$ set of all common bases
while ($|S| > 1$)
    Update w to enforce some tie breaks in S.
    $S \leftarrow$ set of max weight common bases.

- In $O(\log n)$ rounds, unique max weight common base.
- Crucially use a succinct representation of the set of max weight common bases.

Succinct representation of all MSTs

- First question: How do we succinctly represent all maximum weight bases of a Matroid?
- How do we succinctly represent all maximum weight spanning trees in a graph?

Succinct representation of all MSTs

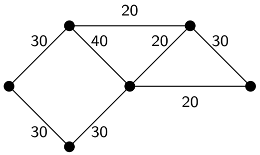


Figure: Graph G

Succinct representation of all MSTs

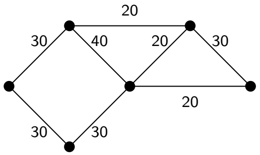


Figure: Graph G

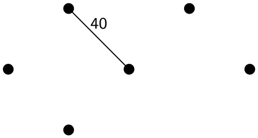


Figure: Graph G_{40}

Succinct representation of all MSTs

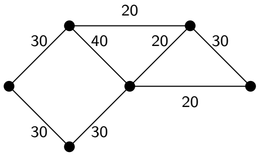


Figure: Graph G

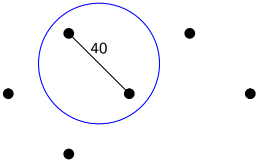


Figure: Graph G_{40}

Succinct representation of all MSTs

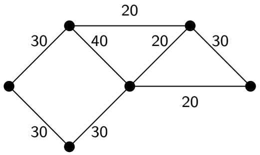


Figure: Graph G

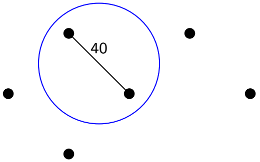


Figure: Graph G_{40}

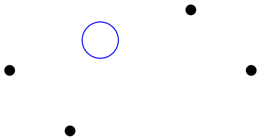


Figure: Graph G_{30}

Succinct representation of all MSTs

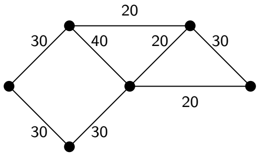


Figure: Graph G

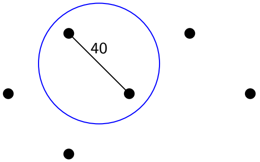


Figure: Graph G_{40}

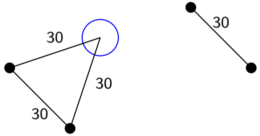


Figure: Graph G_{30}

Succinct representation of all MSTs

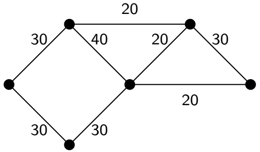


Figure: Graph G

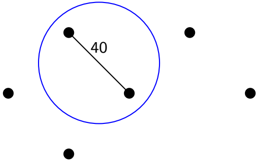


Figure: Graph G_{40}

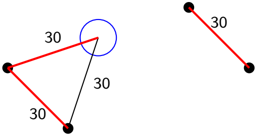


Figure: Graph G_{30}

Succinct representation of all MSTs

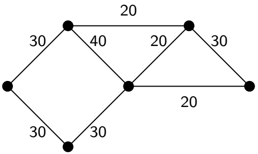


Figure: Graph G

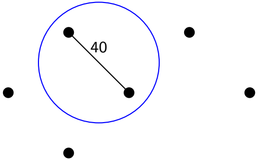


Figure: Graph G_{40}

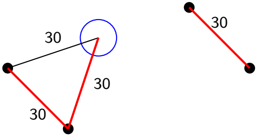


Figure: Graph G_{30}

Succinct representation of all MSTs

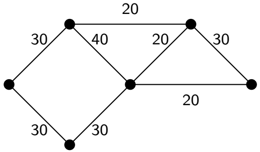


Figure: Graph G

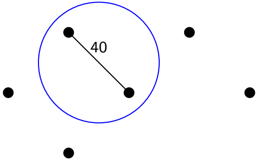


Figure: Graph G_{40}

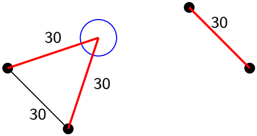


Figure: Graph G_{30}

Succinct representation of all MSTs

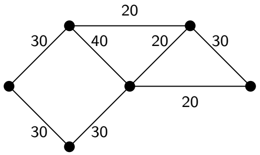


Figure: Graph G

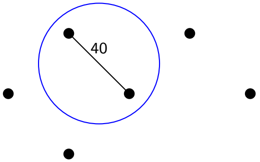


Figure: Graph G_{40}

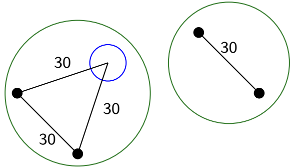


Figure: Graph G_{30}

Succinct representation of all MSTs

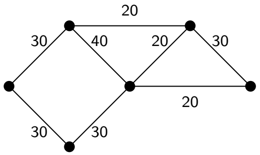


Figure: Graph G

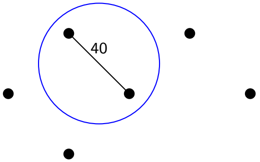


Figure: Graph G_{40}

Figure: Graph G_{20}

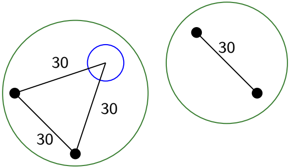


Figure: Graph G_{30}

Succinct representation of all MSTs

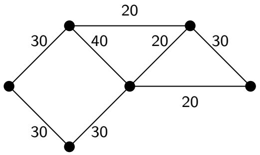


Figure: Graph G

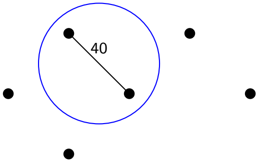


Figure: Graph G_{40}

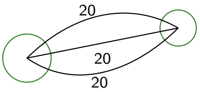


Figure: Graph G_{20}

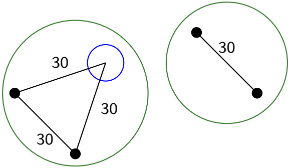


Figure: Graph G_{30}

Succinct representation of all MSTs

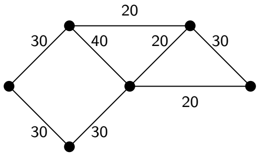


Figure: Graph G

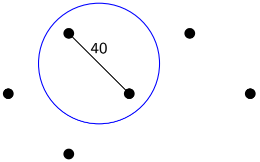


Figure: Graph G_{40}

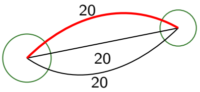


Figure: Graph G_{20}

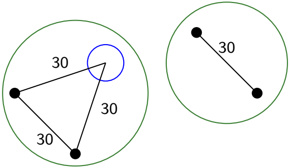


Figure: Graph G_{30}

Succinct representation of all MSTs

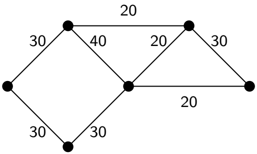


Figure: Graph G

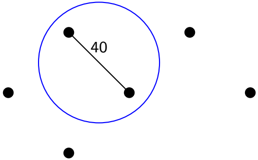


Figure: Graph G_{40}

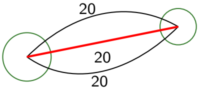


Figure: Graph G_{20}

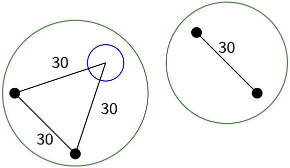


Figure: Graph G_{30}

Succinct representation of all MSTs

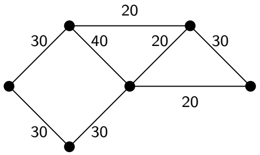


Figure: Graph G

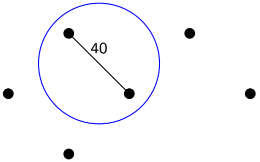


Figure: Graph G_{40}

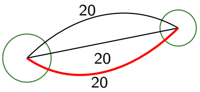


Figure: Graph G_{20}

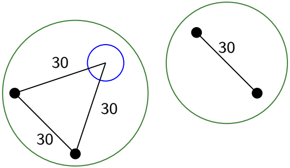


Figure: Graph G_{30}

Succinct representation of all MSTs

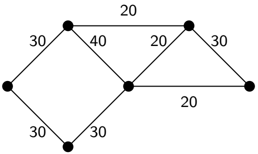


Figure: Graph G

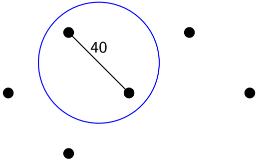


Figure: Graph G_{40}

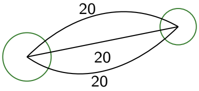


Figure: Graph G_{20}

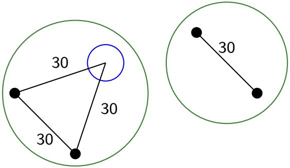


Figure: Graph G_{30}

MSTs in G = {largest forests in G_{40}} $\times$ {largest forests in G_{30}} $\times$ {largest forests in G_{20}}

Succinct representation of all MSTs

Observation: Every MST takes

- 1 edge from G_{40}
- 3 edges from G_{30}
- and 1 edge from G_{20}.

Succinct representation of all max weight common bases

Weight splitting theorem

Succinct representation of all max weight common bases

Weight splitting theorem

- Given two matroids M_1 and M_2 with a weight assignment w,

Succinct representation of all max weight common bases

Weight splitting theorem

- Given two matroids M_1 and M_2 with a weight assignment w,
- there exists a weight splitting $w = w_1 + w_2$ such that

Succinct representation of all max weight common bases

Weight splitting theorem

- Given two matroids M_1 and M_2 with a weight assignment w,
- there exists a weight splitting $w = w_1 + w_2$ such that
- set of max weight common bases =

$$\{\text{max weight bases in } M_1 \text{ w.r.t. } w_1\}$$
$$\cap$$
$$\{\text{max weight bases in } M_2 \text{ w.r.t. } w_2\}$$

Weight-splitting for Bipartite Perfect Matching

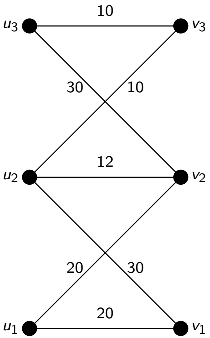


Figure: Weight-splitting

Three perfect matchings

Weight-splitting for Bipartite Perfect Matching

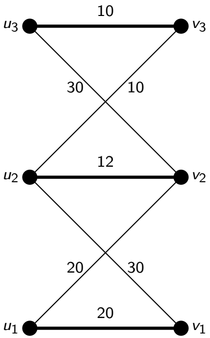


Figure: Weight-splitting

Three perfect matchings

- 10+12+20=42

Weight-splitting for Bipartite Perfect Matching

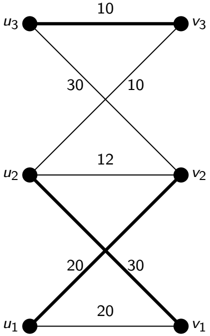


Figure: Weight-splitting

Three perfect matchings

- 10+12+20=42
- 10+20+30=60

Weight-splitting for Bipartite Perfect Matching

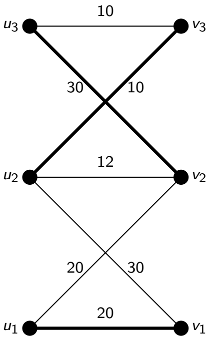


Figure: Weight-splitting

Three perfect matchings

- 10+12+20=42
- 10+20+30=60
- 30+10+20=60

Weight-splitting for Bipartite Perfect Matching

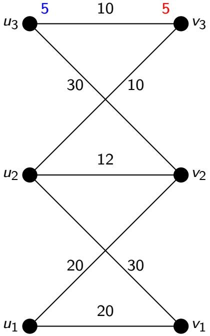


Figure: Weight-splitting

Three perfect matchings

- 10+12+20=42
- 10+20+30=60
- 30+10+20=60

Weight-splitting for Bipartite Perfect Matching

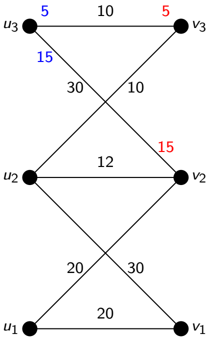


Figure: Weight-splitting

Three perfect matchings

- 10+12+20=42
- 10+20+30=60
- 30+10+20=60

Weight-splitting for Bipartite Perfect Matching

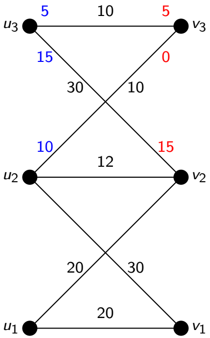


Figure: Weight-splitting

Three perfect matchings

- 10+12+20=42
- 10+20+30=60
- 30+10+20=60

Weight-splitting for Bipartite Perfect Matching

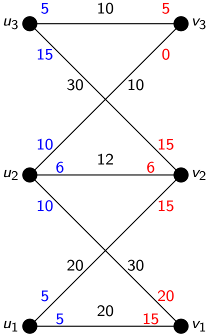


Figure: Weight-splitting

Three perfect matchings

- 10+12+20=42
- 10+20+30=60
- 30+10+20=60

Weight-splitting for Bipartite Perfect Matching

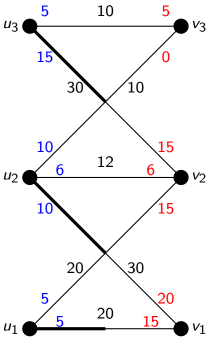


Figure: Weight-splitting

Three perfect matchings

- $10+12+20=42$
- $10+20+30=60$
- $30+10+20=60$

Weight-splitting for Bipartite Perfect Matching

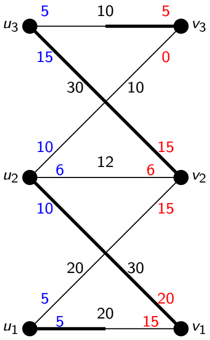


Figure: Weight-splitting

Three perfect matchings

- 10+12+20=42
- 10+20+30=60
- 30+10+20=60

Weight-splitting for Bipartite Perfect Matching

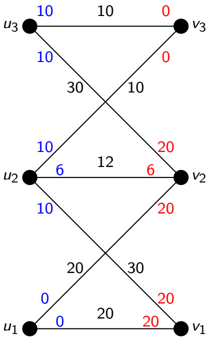


Figure: Weight-splitting

Three perfect matchings

- 10+12+20=42
- 10+20+30=60
- 30+10+20=60

Weight-splitting for Bipartite Perfect Matching

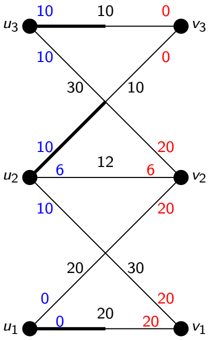


Figure: Weight-splitting

Three perfect matchings

- 10+12+20=42
- 10+20+30=60
- 30+10+20=60

Weight-splitting for Bipartite Perfect Matching

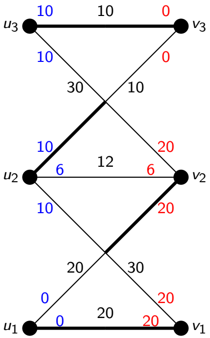


Figure: Weight-splitting

Three perfect matchings

- 10+12+20=42
- 10+20+30=60
- 30+10+20=60

Weight-splitting for Bipartite Perfect Matching

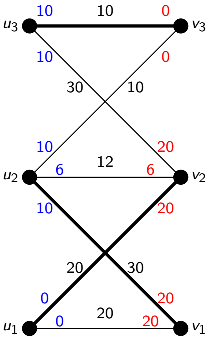


Figure: Weight-splitting

Three perfect matchings

- 10+12+20=42
- 10+20+30=60
- 30+10+20=60

Weight-splitting for Bipartite Perfect Matching

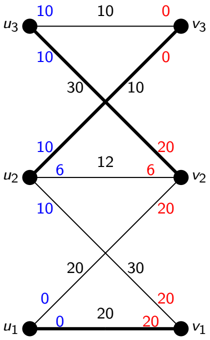


Figure: Weight-splitting

Three perfect matchings

- 10+12+20=42
- 10+20+30=60
- 30+10+20=60

Weight-splitting for Bipartite Perfect Matching

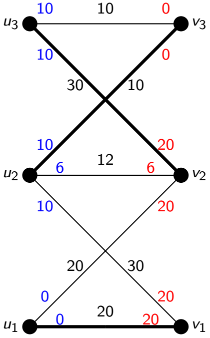


Figure: Weight-splitting

Three perfect matchings

- 10+12+20=42
- 10+20+30=60
- 30+10+20=60

Obs: A perfect matching maximizes w_1 and w_2 $\implies$ it maximizes $w_1 + w_2$.

Weight-splitting for Bipartite Perfect Matching

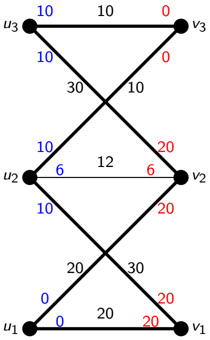


Figure: Weight-splitting

Three perfect matchings

- 10+12+20=42
- 10+20+30=60
- 30+10+20=60

Obs: A perfect matching maximizes w_1 and w_2 $\implies$ it maximizes $w_1 + w_2$.

Thm: All maximum weight perfect matchings can be obtained this way.

Ideas

Algorithm at a high level

$S \leftarrow$ set of all common bases
while $(|S| > 1)$
    Update w to enforce some tie breaks in S.
    $S \leftarrow$ set of max weight common bases.

Ideas

Algorithm at a high level

$S \leftarrow$ set of all common bases
while $(|S| > 1)$
    Update w to enforce some tie breaks in S.
    $S \leftarrow$ set of max weight common bases.

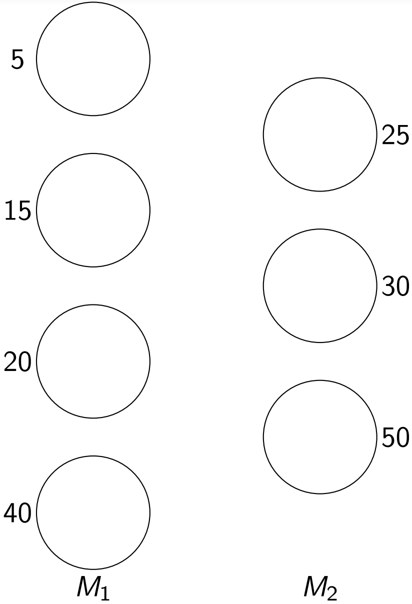
- How do we update w to break ties?

Ideas

Algorithm at a high level

$S \leftarrow$ set of all common bases
while $(|S| > 1)$
    Update w to enforce some tie breaks in S.
    $S \leftarrow$ set of max weight common bases.

- How do we update w to break ties?
- Consider two max weight common bases and their symmetric difference.



Consider two max weight common bases.

Figure: A cycle in two common bases

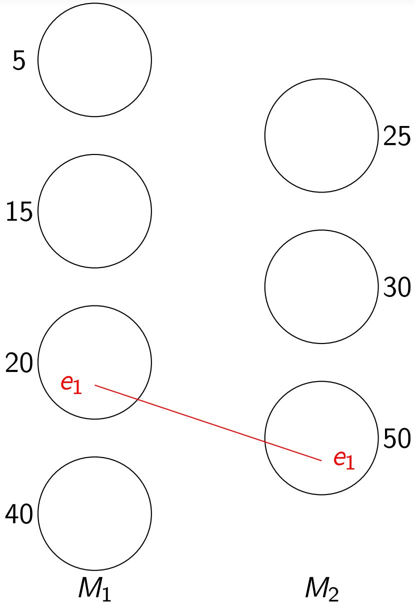


Figure: A cycle in two common bases

Recall: each max weight base has the same number of elements from any weight class.

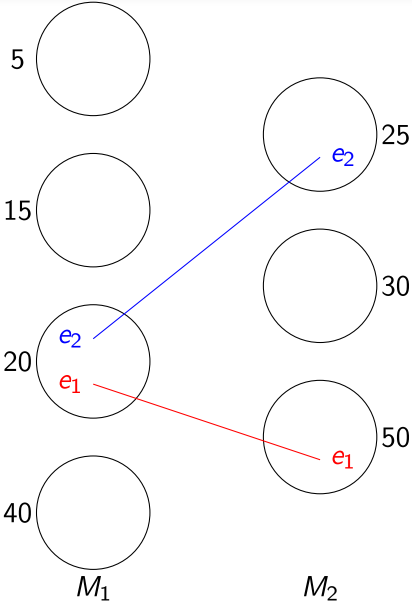


Figure: A cycle in two common bases

Recall: each max weight base has the same number of elements from any weight class.

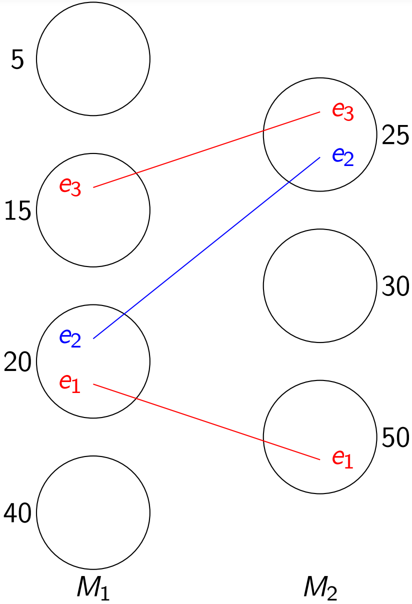


Figure: A cycle in two common bases

Recall: each max weight base has the same number of elements from any weight class.

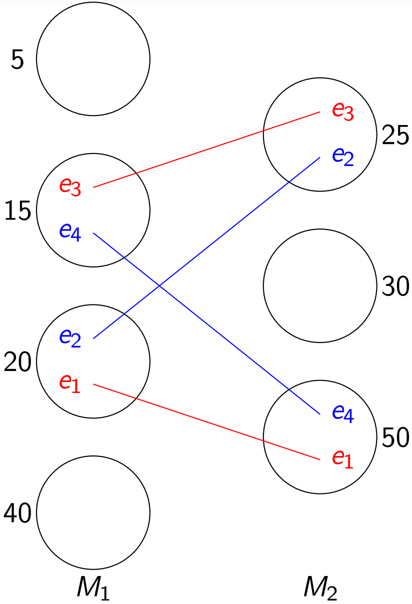


Figure: A cycle in two common bases

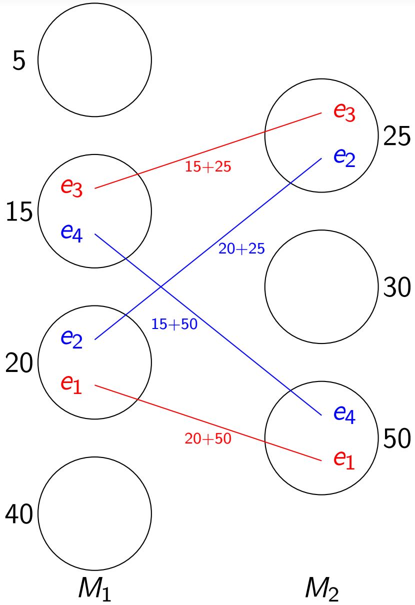


Figure: A cycle in two common bases

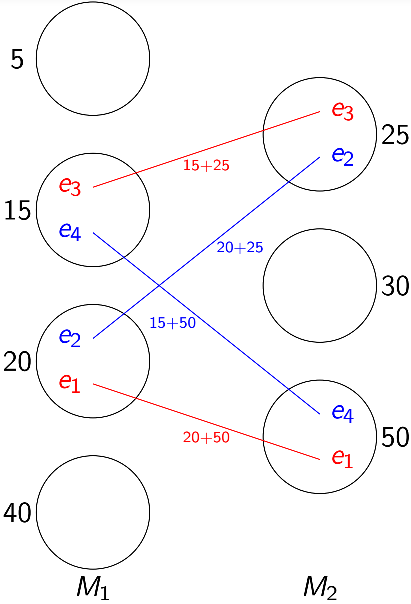


Figure: A cycle in two common bases

Obs. Alternating sum of the cycle

$$w(e_1)+w(e_3)-w(e_2)-w(e_4)$$

$$= 20 + 50 + (15 + 25)$$
$$-(20 + 25) - (15 + 50)$$
$$= 0.$$

Algorithm

Observation

- Weight splitting defines a bipartite graph on the elements.
- Each cycle in this graph has zero alternating sum.

Algorithm

Observation

- Weight splitting defines a bipartite graph on the elements.
- Each cycle in this graph has zero alternating sum.

Algorithm

For $i = 1$ to $\log n$:

- Update w to give nonzero alternating weight to all cycles of length $\leq 2^i$ (need Hashing techinques).
- Recompute weight-splitting and the bipartite graph (need weighted-decision query) [Har07].

Algorithm

Observation

- Weight splitting defines a bipartite graph on the elements.
- Each cycle in this graph has zero alternating sum.

Algorithm

For $i = 1$ to $\log n$:

- Update w to give nonzero alternating weight to all cycles of length $\leq 2^i$ (need Hashing techinques).
- Recompute weight-splitting and the bipartite graph (need weighted-decision query) [Har07].

Termination:

- After i-th iteration, the bipartite graph will not have any cycle of length $\leq 2^i$.

Algorithm

Observation

- Weight splitting defines a bipartite graph on the elements.
- Each cycle in this graph has zero alternating sum.

Algorithm

For $i = 1$ to $\log n$:

- Update w to give nonzero alternating weight to all cycles of length $\leq 2^i$ (need Hashing techinques).
- Recompute weight-splitting and the bipartite graph (need weighted-decision query) [Har07].

Termination:

- After i-th iteration, the bipartite graph will not have any cycle of length $\leq 2^i$.
- After $\log n$ iteration, no cycles remain, and hence unique max weight common base.

Algorithm

Algorithm

For $i = 1$ to $\log n$:

- Update w to give nonzero alternating weight to all cycles of length $\leq 2^i$ (need Hashing techinques).
- Recompute weight-splitting and the bipartite graph (need weighted-decision query) [Har07].

Termination:

- After i-th iteration, the bipartite graph will not have any cycle of length $\leq 2^i$.
- After $\log n$ iteration, no cycles remain, and hence unique max weight common base.

Efficiency:

- When there are no cycles of length $\leq 2^i$, the number of cycles of length $\leq 2^{i+1}$ is polynomial.

Questions

- Derandomize the Isolation Lemma even for Bipartite Matching.

Questions

- Derandomize the Isolation Lemma even for Bipartite Matching.
- Search to decision reduction (in parallel) for bipartite matching.

Questions

- Derandomize the Isolation Lemma even for Bipartite Matching.
- Search to decision reduction (in parallel) for bipartite matching.
- Search to weighted-decision: for what all optimization problems?

Nima Anari and Vijay V. Vazirani.
Matching is as easy as the decision problem, in the NC model.
In Thomas Vidick, editor, *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, volume 151 of *LIPIcs*, pages 54:1–54:25. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

Stephen A. Fenner, Rohit Gurjar, and Thomas Thierauf.
Bipartite perfect matching is in quasi-nc.
In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA*, pages 754–763, 2016.

Shafi Goldwasser and Ofer Grossman.
Bipartite perfect matching in pseudo-deterministic NC.
In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 87:1–87:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

Nicholas J. A. Harvey.
An algebraic algorithm for weighted linear matroid intersection.
In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, page 444453, USA, 2007. Society for Industrial and Applied Mathematics.

Richard M. Karp, Eli Upfal, and Avi Wigderson.
Constructing a perfect matching is in random NC.
Combinatorica, 6(1):35–48, 1986.

Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani.
Matching is as easy as matrix inversion.
Combinatorica, 7:105–113, 1987.