Matching, Graphs and Matroids

Harry Huang

University of Wisconsin Madison

Apr.23.2025

Table of Contents

Introduction to Graphs and Bipartite Graphs

- A **graph** $G = (V, E)$ consists of a set of vertices V and a set of edges E.

Introduction to Graphs and Bipartite Graphs

- A **graph** $G = (V, E)$ consists of a set of vertices V and a set of edges E.
- Each edge connects two vertices. For example: $E = \{(u, v)\}$ with $u, v \in V$.

Introduction to Graphs and Bipartite Graphs

- A **graph** $G = (V, E)$ consists of a set of vertices V and a set of edges E.
- Each edge connects two vertices. For example: $E = \{(u, v)\}$ with $u, v \in V$.
- A **bipartite graph** is a graph where vertices can be divided into two disjoint sets L and R such that every edge connects a vertex from L to a vertex from R.

Introduction to Graphs and Bipartite Graphs

- A **graph** $G = (V, E)$ consists of a set of vertices V and a set of edges E.
- Each edge connects two vertices. For example: $E = \{(u, v)\}$ with $u, v \in V$.
- A **bipartite graph** is a graph where vertices can be divided into two disjoint sets L and R such that every edge connects a vertex from L to a vertex from R.
- Example: $L = \{u\}, R = \{v\}, E = \{(u, v)\}$

Introduction to Graphs and Bipartite Graphs

- A **graph** $G = (V, E)$ consists of a set of vertices V and a set of edges E.
- Each edge connects two vertices. For example: $E = \{(u, v)\}$ with $u, v \in V$.
- A **bipartite graph** is a graph where vertices can be divided into two disjoint sets L and R such that every edge connects a vertex from L to a vertex from R.
- Example: $L = \{u\}, R = \{v\}, E = \{(u, v)\}$
- No edges exist between vertices within the same set.

Introduction to Bipartite Matching

- A **matching** in a bipartite graph is a set of edges with no shared endpoints.

Introduction to Bipartite Matching

- A **matching** in a bipartite graph is a set of edges with no shared endpoints.
- Given a bipartite graph $G = (L \cup R, E)$, a matching $M \subseteq E$ pairs vertices from L to R.

Introduction to Bipartite Matching

- A **matching** in a bipartite graph is a set of edges with no shared endpoints.
- Given a bipartite graph $G = (L \cup R, E)$, a matching $M \subseteq E$ pairs vertices from L to R.
- Each vertex is incident to at most one edge in the matching.

Introduction to Bipartite Matching

- A **matching** in a bipartite graph is a set of edges with no shared endpoints.
- Given a bipartite graph $G = (L \cup R, E)$, a matching $M \subseteq E$ pairs vertices from L to R.
- Each vertex is incident to at most one edge in the matching.
- Example: $L = \{u\}, R = \{v\}, M = \{(u, v)\}$ is a valid matching.

Introduction to Bipartite Matching

- A **matching** in a bipartite graph is a set of edges with no shared endpoints.
- Given a bipartite graph $G = (L \cup R, E)$, a matching $M \subseteq E$ pairs vertices from L to R.
- Each vertex is incident to at most one edge in the matching.
- Example: $L = \{u\}, R = \{v\}, M = \{(u, v)\}$ is a valid matching.
- Goal: Find a matching of maximum size—i.e., the largest possible set of such edges.

Introduction to Network Flow

- A **flow network** is a directed graph $G = (V, E)$ with:
  - A **source** node s and a **sink** node t
  - Each edge (u, v) has a non-negative **capacity** $c(u, v)$

Introduction to Network Flow

- A **flow network** is a directed graph $G = (V, E)$ with:
  - A **source** node s and a **sink** node t
  - Each edge (u, v) has a non-negative **capacity** $c(u, v)$
- A **flow** assigns a value $f(u, v)$ to each edge such that:
  - Capacity constraint: $0 \leq f(u, v) \leq c(u, v)$
  - Flow conservation: For all $v \in V \setminus \{s, t\}$,

$$\sum_u f(u, v) = \sum_w f(v, w)$$

Introduction to Network Flow

- A **flow network** is a directed graph $G = (V, E)$ with:
  - A **source** node s and a **sink** node t
  - Each edge (u, v) has a non-negative **capacity** $c(u, v)$
- A **flow** assigns a value $f(u, v)$ to each edge such that:
  - Capacity constraint: $0 \leq f(u, v) \leq c(u, v)$
  - Flow conservation: For all $v \in V \setminus \{s, t\}$,

$$\sum_u f(u, v) = \sum_w f(v, w)$$

- The **maximum flow problem** seeks the greatest total flow from s to t

Reducing Bipartite Matching to Flow

- Given bipartite graph $G = (L \cup R, E)$, construct a flow network:
  - Add source s and sink t
  - Add edges from s to all nodes in L
  - Add edges from all nodes in R to t
  - Keep original edges between L and R

Reducing Bipartite Matching to Flow

- Given bipartite graph $G = (L \cup R, E)$, construct a flow network:
  - Add source s and sink t
  - Add edges from s to all nodes in L
  - Add edges from all nodes in R to t
  - Keep original edges between L and R
- Assign capacity 1 to all edges

Reducing Bipartite Matching to Flow

- Given bipartite graph $G = (L \cup R, E)$, construct a flow network:
  - Add source s and sink t
  - Add edges from s to all nodes in L
  - Add edges from all nodes in R to t
  - Keep original edges between L and R
- Assign capacity 1 to all edges
- Solve max-flow from s to t

Interpreting the Flow as a Matching

- After computing max-flow:
  - Each unit of flow corresponds to a matched edge between L and R

Interpreting the Flow as a Matching

- After computing max-flow:
  - Each unit of flow corresponds to a matched edge between L and R
- Since all capacities are 1:
  - No vertex in L or R is matched more than once

Interpreting the Flow as a Matching

- After computing max-flow:
  - Each unit of flow corresponds to a matched edge between L and R
- Since all capacities are 1:
  - No vertex in L or R is matched more than once
- The set of edges with flow $= 1$ between L and R gives a maximum matching

Interpreting the Flow as a Matching

- After computing max-flow:
  - Each unit of flow corresponds to a matched edge between L and R
- Since all capacities are 1:
  - No vertex in L or R is matched more than once
- The set of edges with flow $= 1$ between L and R gives a maximum matching
- This method solves bipartite matching in polynomial time using flow algorithms (e.g., Edmonds–Karp)

Non-Bipartite Matching

- In general graphs, we are given an undirected graph $G = (V, E)$.

Non-Bipartite Matching

- In general graphs, we are given an undirected graph $G = (V, E)$.
- A **matching** is a set of edges $M \subseteq E$ such that no two edges in M share a common vertex.

Non-Bipartite Matching

- In general graphs, we are given an undirected graph $G = (V, E)$.
- A **matching** is a set of edges $M \subseteq E$ such that no two edges in M share a common vertex.
- A **maximum matching** is a matching of the largest possible size.

Non-Bipartite Matching

- In general graphs, we are given an undirected graph $G = (V, E)$.
- A **matching** is a set of edges $M \subseteq E$ such that no two edges in M share a common vertex.
- A **maximum matching** is a matching of the largest possible size.
- Unlike bipartite graphs, non-bipartite graphs may contain **odd cycles**, which require special handling.

Why Edmonds' Blossom Algorithm?

- In bipartite graphs, alternating paths are sufficient to find augmenting paths.

Why Edmonds' Blossom Algorithm?

- In bipartite graphs, alternating paths are sufficient to find augmenting paths.
- In non-bipartite graphs, odd cycles (called **blossoms**) can prevent simple alternating path searches.

Why Edmonds' Blossom Algorithm?

- In bipartite graphs, alternating paths are sufficient to find augmenting paths.
- In non-bipartite graphs, odd cycles (called **blossoms**) can prevent simple alternating path searches.
- **Edmonds' Blossom Algorithm** (1965) was the first polynomial-time algorithm for finding maximum matchings in general graphs.

Why Edmonds' Blossom Algorithm?

- In bipartite graphs, alternating paths are sufficient to find augmenting paths.
- In non-bipartite graphs, odd cycles (called **blossoms**) can prevent simple alternating path searches.
- **Edmonds' Blossom Algorithm** (1965) was the first polynomial-time algorithm for finding maximum matchings in general graphs.
- It works by:
  - Searching for **augmenting paths**
  - **Shrinking** blossoms (odd-length cycles) into single vertices
  - Recursively finding matchings in the contracted graph

Blossom Algorithm: Key Steps

- Start with an empty matching M

Blossom Algorithm: Key Steps

- Start with an empty matching M
- While there exists an **augmenting path**:
  - Use BFS or DFS to find alternating paths from unmatched vertices
  - If an **odd cycle** is detected, contract it into a single pseudo-vertex

Blossom Algorithm: Key Steps

- Start with an empty matching M
- While there exists an **augmenting path**:
  - Use BFS or DFS to find alternating paths from unmatched vertices
  - If an **odd cycle** is detected, contract it into a single pseudo-vertex
- If an augmenting path is found, **augment** the matching along it

Blossom Algorithm: Key Steps

- Start with an empty matching M
- While there exists an **augmenting path**:
  - Use BFS or DFS to find alternating paths from unmatched vertices
  - If an **odd cycle** is detected, contract it into a single pseudo-vertex
- If an augmenting path is found, **augment** the matching along it
- Expand any contracted blossoms and update the matching accordingly

Blossom Algorithm: Key Steps

- Start with an empty matching M
- While there exists an **augmenting path**:
  - Use BFS or DFS to find alternating paths from unmatched vertices
  - If an **odd cycle** is detected, contract it into a single pseudo-vertex
- If an augmenting path is found, **augment** the matching along it
- Expand any contracted blossoms and update the matching accordingly
- Repeat until no more augmenting paths exist

Introduction to Matroids

- A **matroid** is a combinatorial structure that generalizes the concept of independence in vector spaces and graphs.

Introduction to Matroids

- A **matroid** is a combinatorial structure that generalizes the concept of independence in vector spaces and graphs.
- Formally, a matroid is a pair $M = (E, \mathcal{I})$ where:
  - E is a finite set (called the **ground set**)
  - $\mathcal{I} \subseteq 2^E$ is a collection of **independent sets**

Introduction to Matroids

- A **matroid** is a combinatorial structure that generalizes the concept of independence in vector spaces and graphs.
- Formally, a matroid is a pair $M = (E, \mathcal{I})$ where:
  - E is a finite set (called the **ground set**)
  - $\mathcal{I} \subseteq 2^E$ is a collection of **independent sets**
- The sets in $\mathcal{I}$ must satisfy specific axioms.

Matroid Axioms

The collection $\mathcal{I}$ of independent sets must satisfy:

- **(I1) Non-empty:** $\emptyset \in \mathcal{I}$

Matroid Axioms

The collection $\mathcal{I}$ of independent sets must satisfy:

- **(I1) Non-empty:** $\emptyset \in \mathcal{I}$
- **(I2) Hereditary:** If $A \in \mathcal{I}$ and $B \subseteq A$, then $B \in \mathcal{I}$

These axioms ensure a well-behaved notion of independence, similar to linear independence or forest edges in graphs.

Matroid Axioms

The collection $\mathcal{I}$ of independent sets must satisfy:

- **(I1) Non-empty:** $\emptyset \in \mathcal{I}$
- **(I2) Hereditary:** If $A \in \mathcal{I}$ and $B \subseteq A$, then $B \in \mathcal{I}$
- **(I3) Exchange:** If $A, B \in \mathcal{I}$ and $|A| < |B|$, then there exists $e \in B \setminus A$ such that $A \cup \{e\} \in \mathcal{I}$

These axioms ensure a well-behaved notion of independence, similar to linear independence or forest edges in graphs.

Examples of Matroids

- **Uniform Matroid:** $E = \{1, 2, \ldots, n\}$, independent sets are all subsets of size $\leq k$

Examples of Matroids

- **Uniform Matroid:** $E = \{1, 2, \ldots, n\}$, independent sets are all subsets of size $\leq k$
- **Graphic Matroid:** E is the set of edges in a graph, and a set is independent if it forms a forest (acyclic)

These examples help unify different domains—graphs, algebra, and combinatorics—under the same framework.

Examples of Matroids

- **Uniform Matroid:** $E = \{1, 2, \ldots, n\}$, independent sets are all subsets of size $\leq k$
- **Graphic Matroid:** E is the set of edges in a graph, and a set is independent if it forms a forest (acyclic)
- **Linear Matroid:** E is a set of vectors, and a set is independent if the vectors are linearly independent

These examples help unify different domains—graphs, algebra, and combinatorics—under the same framework.

Introduction to Partition Matroid

- A **partition matroid** is a type of matroid where the ground set is divided into disjoint parts, each with a capacity.

Introduction to Partition Matroid

- A **partition matroid** is a type of matroid where the ground set is divided into disjoint parts, each with a capacity.
- Let E be partitioned into k disjoint subsets: $E = E_1 \cup E_2 \cup \cdots \cup E_k$

Introduction to Partition Matroid

- A **partition matroid** is a type of matroid where the ground set is divided into disjoint parts, each with a capacity.
- Let E be partitioned into k disjoint subsets: $E = E_1 \cup E_2 \cup \cdots \cup E_k$
- Each part E_i has an integer bound r_i

Introduction to Partition Matroid

- A **partition matroid** is a type of matroid where the ground set is divided into disjoint parts, each with a capacity.
- Let E be partitioned into k disjoint subsets: $E = E_1 \cup E_2 \cup \cdots \cup E_k$
- Each part E_i has an integer bound r_i
- A set $S \subseteq E$ is **independent** if for all i, $|S \cap E_i| \leq r_i$

Introduction to Partition Matroid

- A **partition matroid** is a type of matroid where the ground set is divided into disjoint parts, each with a capacity.
- Let E be partitioned into k disjoint subsets: $E = E_1 \cup E_2 \cup \cdots \cup E_k$
- Each part E_i has an integer bound r_i
- A set $S \subseteq E$ is **independent** if for all i, $|S \cap E_i| \leq r_i$
- That is, we can select at most r_i elements from each group E_i

Bipartite Matching via Matroid Intersection

- Consider bipartite graph $G = (L \cup R, E)$

Bipartite Matching via Matroid Intersection

- Consider bipartite graph $G = (L \cup R, E)$
- Goal: Find the largest matching — a subset of edges $M \subseteq E$ with no shared endpoints

Bipartite Matching via Matroid Intersection

- Consider bipartite graph $G = (L \cup R, E)$
- Goal: Find the largest matching — a subset of edges $M \subseteq E$ with no shared endpoints
- Idea: Model the matching problem as an intersection of two matroids on the ground set E

Bipartite Matching via Matroid Intersection

- Consider bipartite graph $G = (L \cup R, E)$
- Goal: Find the largest matching — a subset of edges $M \subseteq E$ with no shared endpoints
- Idea: Model the matching problem as an intersection of two matroids on the ground set E
- This allows us to use polynomial-time algorithms for matroid intersection

Two Partition Matroids from Matching Constraints

We define two matroids over the edge set E in a bipartite graph $G = (L \cup R, E)$:

- **Matroid** $\mathcal{M}_L = (E, \mathcal{I}_L)$:
  - $I \in \mathcal{I}_L$ if no two edges in I share a vertex in L
  - Partition E by vertex in L: edges incident to $u \in L$ form a part
  - Each part has capacity $1 \rightarrow$ at most one edge per $u \in L$

Two Partition Matroids from Matching Constraints

We define two matroids over the edge set E in a bipartite graph $G = (L \cup R, E)$:

- **Matroid** $\mathcal{M}_L = (E, \mathcal{I}_L)$:
  - $I \in \mathcal{I}_L$ if no two edges in I share a vertex in L
  - Partition E by vertex in L: edges incident to $u \in L$ form a part
  - Each part has capacity $1 \rightarrow$ at most one edge per $u \in L$
- **Matroid** $\mathcal{M}_R = (E, \mathcal{I}_R)$:
  - $I \in \mathcal{I}_R$ if no two edges in I share a vertex in R
  - Partition E by vertex in R: edges incident to $v \in R$ form a part
  - Each part has capacity $1 \rightarrow$ at most one edge per $v \in R$

Two Partition Matroids from Matching Constraints

We define two matroids over the edge set E in a bipartite graph $G = (L \cup R, E)$:

- **Matroid** $\mathcal{M}_L = (E, \mathcal{I}_L)$:
  - $I \in \mathcal{I}_L$ if no two edges in I share a vertex in L
  - Partition E by vertex in L: edges incident to $u \in L$ form a part
  - Each part has capacity $1 \to$ at most one edge per $u \in L$
- **Matroid** $\mathcal{M}_R = (E, \mathcal{I}_R)$:
  - $I \in \mathcal{I}_R$ if no two edges in I share a vertex in R
  - Partition E by vertex in R: edges incident to $v \in R$ form a part
  - Each part has capacity $1 \to$ at most one edge per $v \in R$
- Both $\mathcal{M}_L$ and $\mathcal{M}_R$ are partition matroids.

Matchings as Intersection of Two Matroids

- A subset $I \subseteq E$ is a **matching** if:
  - No two edges in I share a vertex (either in L or in R)

Matchings as Intersection of Two Matroids

- A subset $I \subseteq E$ is a **matching** if:
  - No two edges in I share a vertex (either in L or in R)
- This is exactly the condition:

$$I \in \mathcal{I}_L \cap \mathcal{I}_R$$

Matchings as Intersection of Two Matroids

- A subset $I \subseteq E$ is a **matching** if:
  - No two edges in I share a vertex (either in L or in R)
- This is exactly the condition:

$$I \in \mathcal{I}_L \cap \mathcal{I}_R$$

- So a matching is an edge set that is independent in both $\mathcal{M}_L$ and $\mathcal{M}_R$

Matchings as Intersection of Two Matroids

- A subset $I \subseteq E$ is a **matching** if:
  - No two edges in I share a vertex (either in L or in R)
- This is exactly the condition:

$$I \in \mathcal{I}_L \cap \mathcal{I}_R$$

- So a matching is an edge set that is independent in both $\mathcal{M}_L$ and $\mathcal{M}_R$
- Thus, **maximum bipartite matching** $=$ **maximum-size common independent set** in $\mathcal{M}_L$ and $\mathcal{M}_R$

Matchings as Intersection of Two Matroids

- A subset $I \subseteq E$ is a **matching** if:
  - No two edges in I share a vertex (either in L or in R)
- This is exactly the condition:

$$I \in \mathcal{I}_L \cap \mathcal{I}_R$$

- So a matching is an edge set that is independent in both $\mathcal{M}_L$ and $\mathcal{M}_R$
- Thus, **maximum bipartite matching** $=$ **maximum-size common independent set** in $\mathcal{M}_L$ and $\mathcal{M}_R$
- We can solve it using **matroid intersection algorithms**

Lawler's Algorithm for Matroid Intersection

- Given two matroids $\mathcal{M}_1 = (E, \mathcal{I}_1)$ and $\mathcal{M}_2 = (E, \mathcal{I}_2)$ over the same ground set E

Lawler's Algorithm for Matroid Intersection

- Given two matroids $\mathcal{M}_1 = (E, \mathcal{I}_1)$ and $\mathcal{M}_2 = (E, \mathcal{I}_2)$ over the same ground set E
- Goal: Find the largest set $I \subseteq E$ such that $I \in \mathcal{I}_1 \cap \mathcal{I}_2$

Lawler's Algorithm for Matroid Intersection

- Given two matroids $\mathcal{M}_1 = (E, \mathcal{I}_1)$ and $\mathcal{M}_2 = (E, \mathcal{I}_2)$ over the same ground set E
- Goal: Find the largest set $I \subseteq E$ such that $I \in \mathcal{I}_1 \cap \mathcal{I}_2$
- **Lawler's Algorithm**:

Lawler's Algorithm for Matroid Intersection

- Given two matroids $\mathcal{M}_1 = (E, \mathcal{I}_1)$ and $\mathcal{M}_2 = (E, \mathcal{I}_2)$ over the same ground set E
- Goal: Find the largest set $I \subseteq E$ such that $I \in \mathcal{I}_1 \cap \mathcal{I}_2$
- **Lawler's Algorithm**:
  1. Start with $I = \emptyset$

Lawler's Algorithm for Matroid Intersection

- Given two matroids $\mathcal{M}_1 = (E, \mathcal{I}_1)$ and $\mathcal{M}_2 = (E, \mathcal{I}_2)$ over the same ground set E
- Goal: Find the largest set $I \subseteq E$ such that $I \in \mathcal{I}_1 \cap \mathcal{I}_2$
- **Lawler's Algorithm**:
  1. Start with $I = \emptyset$
  2. While there exists an **augmenting path** in the **exchange graph**:
     - Use a shortest path search to find an element $e \notin I$ and $f \in I$ that can be exchanged

Lawler's Algorithm for Matroid Intersection

- Given two matroids $\mathcal{M}_1 = (E, \mathcal{I}_1)$ and $\mathcal{M}_2 = (E, \mathcal{I}_2)$ over the same ground set E
- Goal: Find the largest set $I \subseteq E$ such that $I \in \mathcal{I}_1 \cap \mathcal{I}_2$
- **Lawler's Algorithm**:
  1. Start with $I = \emptyset$
  2. While there exists an **augmenting path** in the **exchange graph**:
     - Use a shortest path search to find an element $e \notin I$ and $f \in I$ that can be exchanged
  3. Augment I by replacing f with e

Lawler's Algorithm for Matroid Intersection

- Given two matroids $\mathcal{M}_1 = (E, \mathcal{I}_1)$ and $\mathcal{M}_2 = (E, \mathcal{I}_2)$ over the same ground set E
- Goal: Find the largest set $I \subseteq E$ such that $I \in \mathcal{I}_1 \cap \mathcal{I}_2$
- **Lawler's Algorithm**:
  1. Start with $I = \emptyset$
  2. While there exists an **augmenting path** in the **exchange graph**:
     - Use a shortest path search to find an element $e \notin I$ and $f \in I$ that can be exchanged
  3. Augment I by replacing f with e
  4. Repeat until no more augmenting paths exist

Lawler's Algorithm for Matroid Intersection

- Given two matroids $\mathcal{M}_1 = (E, \mathcal{I}_1)$ and $\mathcal{M}_2 = (E, \mathcal{I}_2)$ over the same ground set E
- Goal: Find the largest set $I \subseteq E$ such that $I \in \mathcal{I}_1 \cap \mathcal{I}_2$
- **Lawler's Algorithm**:
  1. Start with $I = \emptyset$
  2. While there exists an **augmenting path** in the **exchange graph**:
     - Use a shortest path search to find an element $e \notin I$ and $f \in I$ that can be exchanged
  3. Augment I by replacing f with e
  4. Repeat until no more augmenting paths exist
- Runs in polynomial time using independence oracles for both matroids

Solving Matching via Matroid Intersection

- Use the **matroid intersection algorithm** to find the largest set $I \subseteq E$ that is independent in both $\mathcal{M}_L$ and $\mathcal{M}_R$

Solving Matching via Matroid Intersection

- Use the **matroid intersection algorithm** to find the largest set $I \subseteq E$ that is independent in both $\mathcal{M}_L$ and $\mathcal{M}_R$
- This gives a maximum matching in the bipartite graph

Solving Matching via Matroid Intersection

- Use the **matroid intersection algorithm** to find the largest set $I \subseteq E$ that is independent in both $\mathcal{M}_L$ and $\mathcal{M}_R$
- This gives a maximum matching in the bipartite graph
- Time complexity: $O(nm)$ using augmenting path methods, where $n = |V|$, $m = |E|$

Solving Matching via Matroid Intersection

- Use the **matroid intersection algorithm** to find the largest set $I \subseteq E$ that is independent in both $\mathcal{M}_L$ and $\mathcal{M}_R$
- This gives a maximum matching in the bipartite graph
- Time complexity: $O(nm)$ using augmenting path methods, where $n = |V|$, $m = |E|$
- Conceptually unifies matching and matroid theory!

Conclusion

- **Matching** is a fundamental concept in graph theory with applications in scheduling, resource allocation, and optimization.

Conclusion

- **Matching** is a fundamental concept in graph theory with applications in scheduling, resource allocation, and optimization.
- **Bipartite matching** can be solved efficiently using:
  - **Network flow**, by modeling matching as a maximum flow problem
  - **Matroid intersection**, by capturing vertex constraints using partition matroids

Conclusion

- **Matching** is a fundamental concept in graph theory with applications in scheduling, resource allocation, and optimization.
- **Bipartite matching** can be solved efficiently using:
  - **Network flow**, by modeling matching as a maximum flow problem
  - **Matroid intersection**, by capturing vertex constraints using partition matroids
- **Non-bipartite matching** is more complex and requires the **Edmonds' Blossom Algorithm** to handle odd-length cycles.

Conclusion

- **Matching** is a fundamental concept in graph theory with applications in scheduling, resource allocation, and optimization.
- **Bipartite matching** can be solved efficiently using:
  - **Network flow**, by modeling matching as a maximum flow problem
  - **Matroid intersection**, by capturing vertex constraints using partition matroids
- **Non-bipartite matching** is more complex and requires the **Edmonds' Blossom Algorithm** to handle odd-length cycles.
- **Matroids** provide a unifying framework for understanding independence across different domains and lead to elegant algorithmic solutions.

Conclusion

- **Matching** is a fundamental concept in graph theory with applications in scheduling, resource allocation, and optimization.
- **Bipartite matching** can be solved efficiently using:
  - **Network flow**, by modeling matching as a maximum flow problem
  - **Matroid intersection**, by capturing vertex constraints using partition matroids
- **Non-bipartite matching** is more complex and requires the **Edmonds' Blossom Algorithm** to handle odd-length cycles.
- **Matroids** provide a unifying framework for understanding independence across different domains and lead to elegant algorithmic solutions.
- These concepts are not only theoretically rich, but also practically powerful in solving real-world discrete optimization problems.