Application of Network Flow 1: Bipartite Matchings

Definition

Given an undirected graph $G = (V, E)$, a set of edges $M \subseteq E$ is a *matching* if each node in V is incident to at most one edge in M.

Application of Network Flow 1: Bipartite Matchings

Definition

Given an undirected graph $G = (V, E)$, a set of edges $M \subseteq E$ is a *matching* if each node in V is incident to at most one edge in M.

The maximum bipartite matching problem:

- Input: a bipartite graph $G = (U, V, E)$. (Recall: this means all edges have one node in U and the other in V.)
- Output: a matching M with the maximum cardinality.

Application of Network Flow 1: Bipartite Matchings

Definition

Given an undirected graph $G = (V, E)$, a set of edges $M \subseteq E$ is a *matching* if each node in V is incident to at most one edge in M.

The maximum bipartite matching problem:

- Input: a bipartite graph $G = (U, V, E)$. (Recall: this means all edges have one node in U and the other in V.)
- Output: a matching M with the maximum cardinality.
- *Reduce* the problem to a max flow problem.

Application of Network Flow 1: Bipartite Matchings

Definition

Given an undirected graph $G = (V, E)$, a set of edges $M \subseteq E$ is a *matching* if each node in V is incident to at most one edge in M.

The maximum bipartite matching problem:

- Input: a bipartite graph $G = (U, V, E)$. (Recall: this means all edges have one node in U and the other in V.)
- Output: a matching M with the maximum cardinality.
- *Reduce* the problem to a max flow problem.
  - Add a source s connected to all nodes in U, and a sink t connected to all nodes in V.

Application of Network Flow 1: Bipartite Matchings

Definition

Given an undirected graph $G = (V, E)$, a set of edges $M \subseteq E$ is a *matching* if each node in V is incident to at most one edge in M.

The maximum bipartite matching problem:

- Input: a bipartite graph $G = (U, V, E)$. (Recall: this means all edges have one node in U and the other in V.)
- Output: a matching M with the maximum cardinality.
- *Reduce* the problem to a max flow problem.
  - Add a source s connected to all nodes in U, and a sink t connected to all nodes in V.
  - Direct the edges from s to U to V to t.

Application of Network Flow 1: Bipartite Matchings

Definition

Given an undirected graph $G = (V, E)$, a set of edges $M \subseteq E$ is a *matching* if each node in V is incident to at most one edge in M.

The maximum bipartite matching problem:

- Input: a bipartite graph $G = (U, V, E)$. (Recall: this means all edges have one node in U and the other in V.)
- Output: a matching M with the maximum cardinality.
- *Reduce* the problem to a max flow problem.
  - Add a source s connected to all nodes in U, and a sink t connected to all nodes in V.
  - Direct the edges from s to U to V to t.
  - Let all capacities be 1.

Application of Network Flow 1: Bipartite Matchings

Definition

Given an undirected graph $G = (V, E)$, a set of edges $M \subseteq E$ is a *matching* if each node in V is incident to at most one edge in M.

The maximum bipartite matching problem:

- Input: a bipartite graph $G = (U, V, E)$. (Recall: this means all edges have one node in U and the other in V.)
- Output: a matching M with the maximum cardinality.
- *Reduce* the problem to a max flow problem.
  - Add a source s connected to all nodes in U, and a sink t connected to all nodes in V.
  - Direct the edges from s to U to V to t.
  - Let all capacities be 1.
- We can find a maximum bipartite matching in time $O(mn)$.

Further consequences

Definition

In a biparitite graph $G = (U, V, E)$ with $|U| = |V|$, a matching M is said to be *perfect* if $|M| = |U|$.

Further consequences

Definition

In a biparitite graph $G = (U, V, E)$ with $|U| = |V|$, a matching M is said to be *perfect* if $|M| = |U|$.

In an undirected graph $G = (V, E)$, a node u is a neighbor of another node v if $(u, v) \in E$.

Further consequences

Definition

In a biparitite graph $G = (U, V, E)$ with $|U| = |V|$, a matching M is said to be *perfect* if $|M| = |U|$.

In an undirected graph $G = (V, E)$, a node u is a neighbor of another node v if $(u, v) \in E$. For a set of nodes $S \subseteq V$, let us denote by $\delta(S)$ the "neighbors" of S, i.e., a node is in $\delta(S)$ if it has a neighbor in S.

Theorem

Hall's Theorem, a.k.a. Marriage Theorem A bipartite graph $G = (U, V, E)$ has a perfect matching if and only if for any $S \subseteq U$, $|\delta(S)| \geq |S|$.

Further consequences

Definition

In a biparitite graph $G = (U, V, E)$ with $|U| = |V|$, a matching M is said to be *perfect* if $|M| = |U|$.

In an undirected graph $G = (V, E)$, a node u is a neighbor of another node v if $(u, v) \in E$. For a set of nodes $S \subseteq V$, let us denote by $\delta(S)$ the "neighbors" of S, i.e., a node is in $\delta(S)$ if it has a neighbor in S.

Theorem

Hall's Theorem, a.k.a. Marriage Theorem A bipartite graph $G = (U, V, E)$ has a perfect matching if and only if for any $S \subseteq U$, $|\delta(S)| \geq |S|$.

Proof idea: To show a perfect matching exists, argue that the min cut in the corresponding flow network has capacity n.

Further consequences

Definition

In a biparitite graph $G = (U, V, E)$ with $|U| = |V|$, a matching M is said to be *perfect* if $|M| = |U|$.

In an undirected graph $G = (V, E)$, a node u is a neighbor of another node v if $(u, v) \in E$. For a set of nodes $S \subseteq V$, let us denote by $\delta(S)$ the "neighbors" of S, i.e., a node is in $\delta(S)$ if it has a neighbor in S.

Theorem

Hall's Theorem, a.k.a. Marriage Theorem A bipartite graph $G = (U, V, E)$ has a perfect matching if and only if for any $S \subseteq U$, $|\delta(S)| \geq |S|$.

Proof idea: To show a perfect matching exists, argue that the min cut in the corresponding flow network has capacity n.
(One can also show the theorem directly via induction. Try it!)

A Glimpse at Better Algorithms

- Reducing to network flows and solving by Ford-Fulkerson is not the fastest algorithm to find maximum bipartite matchings.

A Glimpse at Better Algorithms

- Reducing to network flows and solving by Ford-Fulkerson is not the fastest algorithm to find maximum bipartite matchings.
- Fastest algorithm known: Hopcroft-Karp algorithm, which runs in time $O(m\sqrt{n})$.

A Glimpse at Better Algorithms

- Reducing to network flows and solving by Ford-Fulkerson is not the fastest algorithm to find maximum bipartite matchings.
- Fastest algorithm known: Hopcroft-Karp algorithm, which runs in time $O(m\sqrt{n})$.
- Basic idea: In each iteration, instead of augmenting along a path, look for a *maximal* set of vertex-disjoint *shortest* augmenting paths, and augment along all of them.

A Glimpse at Better Algorithms

- Reducing to network flows and solving by Ford-Fulkerson is not the fastest algorithm to find maximum bipartite matchings.
- Fastest algorithm known: Hopcroft-Karp algorithm, which runs in time $O(m\sqrt{n})$.
- Basic idea: In each iteration, instead of augmenting along a path, look for a *maximal* set of vertex-disjoint *shortest* augmenting paths, and augment along all of them.
- Similar ideas (of augmenting along a collection of shortest paths that "block" s from t) lead to faster algorithms for the max flow problem: Dinic's algorithm, running in time $O(mn^2)$.
- (The algorithm by Edmonds and Karp that run in time $O(m^2 n)$ is an important predecessor.