Flow problem: basic definitions

- Basic setup: we are given a directed graph $G = (V, E)$, which includes a special node s called the *source* and a node t called the *sink*. Each edge (u, v) is associated with a capacity $c(u, v) > 0$.
- Convention: $c(u, v) = 0$ for $(u, v) \notin E$.

Flow problem: basic definitions

- Basic setup: we are given a directed graph $G = (V, E)$, which includes a special node s called the *source* and a node t called the *sink*. Each edge (u, v) is associated with a capacity $c(u, v) > 0$.
- Convention: $c(u, v) = 0$ for $(u, v) \notin E$.

Definition

A *flow* is a function $f : V^2 \to \mathbb{R}$ satisfying:

1. *skew symmetry* $\forall u, v \in V,\ f(u, v) = -f(v, u)$;
2. *conservation of flow* $\forall u \in V - \{s, t\},\ \sum_{v \in V} f(v, u) = 0$.
3. *capacity constraints* $\forall u, v \in V,\ f(u, v) \leq c(u, v)$.

The *value* of a flow f is $|f| := \sum_{v \in V} f(s, v)$.

Flow problem: basic definitions

- Basic setup: we are given a directed graph $G = (V, E)$, which includes a special node s called the *source* and a node t called the *sink*. Each edge (u, v) is associated with a capacity $c(u, v) > 0$.
- Convention: $c(u, v) = 0$ for $(u, v) \notin E$.

Definition

A *flow* is a function $f : V^2 \to \mathbb{R}$ satisfying:

1. *skew symmetry* $\forall u, v \in V,\ f(u, v) = -f(v, u)$;
2. *conservation of flow* $\forall u \in V - \{s, t\},\ \sum_{v \in V} f(v, u) = 0$.
3. *capacity constraints* $\forall u, v \in V,\ f(u, v) \leq c(u, v)$.

The *value* of a flow f is $|f| := \sum_{v \in V} f(s, v)$.

The maximum flow problem: given G and capacities on its edges, compute a flow with maximum value.

Cuts

- Given a graph $G = (V, E)$, a *cut* is a partition of V into two sets A and B. That is, $A \cap B = \emptyset$, $A \cup B = V$.

Cuts

- Given a graph $G = (V, E)$, a *cut* is a partition of V into two sets A and B. That is, $A \cap B = \emptyset$, $A \cup B = V$.
- Given a graph with source s and sink t, an $s - t$ *cut* is a cut (A, B) such that s is in A and t is in B.

Cuts

- Given a graph $G = (V, E)$, a *cut* is a partition of V into two sets A and B. That is, $A \cap B = \emptyset$, $A \cup B = V$.
- Given a graph with source s and sink t, an $s - t$ *cut* is a cut (A, B) such that s is in A and t is in B.
- The capacity of a cut (A, B) is $\sum_{u \in A, v \in B} c(u, v)$.

Cuts

- Given a graph $G = (V, E)$, a *cut* is a partition of V into two sets A and B. That is, $A \cap B = \emptyset$, $A \cup B = V$.
- Given a graph with source s and sink t, an *$s - t$ cut* is a cut (A, B) such that s is in A and t is in B.
- The capacity of a cut (A, B) is $\sum_{u \in A, v \in B} c(u, v)$.
- Let f be a flow, *the flow across a cut* (A, B) is $\sum_{u \in A, v \in B} f(u, v)$.

Cuts

- Given a graph $G = (V, E)$, a *cut* is a partition of V into two sets A and B. That is, $A \cap B = \emptyset$, $A \cup B = V$.
- Given a graph with source s and sink t, an *$s - t$ cut* is a cut (A, B) such that s is in A and t is in B.
- The capacity of a cut (A, B) is $\sum_{u \in A, v \in B} c(u, v)$.
- Let f be a flow, *the flow across a cut* (A, B) is $\sum_{u \in A, v \in B} f(u, v)$.

Lemma

For any $s - t$ cut (A, B) and any flow f, the value of f is $f(A, B)$.

Cuts

- Given a graph $G = (V, E)$, a *cut* is a partition of V into two sets A and B. That is, $A \cap B = \emptyset$, $A \cup B = V$.
- Given a graph with source s and sink t, an *$s - t$ cut* is a cut (A, B) such that s is in A and t is in B.
- The capacity of a cut (A, B) is $\sum_{u \in A, v \in B} c(u, v)$.
- Let f be a flow, *the flow across a cut* (A, B) is $\sum_{u \in A, v \in B} f(u, v)$.

Lemma

For any $s - t$ cut (A, B) and any flow f, the value of f is $f(A, B)$.

Lemma

For any $s - t$ cut (A, B) and any flow f, $|f| \leq c(A, B)$.

Ford-Fulkerson Algorithm and Max Flow Min Cut Theorem

- Given a flow f on a graph with capacities c, the *residual capacity* of a directed edge (u, v) is $r(u, v) := c(u, v) - f(u, v)$.

Ford-Fulkerson Algorithm and Max Flow Min Cut Theorem

- Given a flow f on a graph with capacities c, the *residual capacity* of a directed edge (u, v) is $r(u, v) := c(u, v) - f(u, v)$.
- Given a flow f in a graph G with capacities c, the *residual graph* is a graph G_f where (u, v) is an edge iff $r(u, v) > 0$.

Ford-Fulkerson Algorithm and Max Flow Min Cut Theorem

- Given a flow f on a graph with capacities c, the *residual capacity* of a directed edge (u, v) is $r(u, v) := c(u, v) - f(u, v)$.
- Given a flow f in a graph G with capacities c, the *residual graph* is a graph G_f where (u, v) is an edge iff $r(u, v) > 0$.

Lemma

Given a flow f on a graph G, and the associated residual graph G_f, let f' be a function that maps V^2 to $\mathbb{R}$, then

1. *f' is a flow iff $f + f'$ is a flow on G.*
2. *If f' is a flow on G_f, then $|f + f'| = |f| + |f'|$, and $|f - f'| = |f| - |f'|$.*

The Max Flow Min Cut Theorem

- Given a flow f on a graph G with source s and sink t, an *augmenting path* is a directed path from s to t in the residual graph G_f.

The Max Flow Min Cut Theorem

- Given a flow f on a graph G with source s and sink t, an *augmenting path* is a directed path from s to t in the residual graph G_f.

Theorem (Max Flow Min Cut Theorem)

The following statements are equivalent:

1. *f is a maximum flow on G with capacities c,*
2. *there is a $s - t$ cut (A, B) with $c(A, B) = |f|$,*
3. *There exists no augmenting path in G_f.*

- Given a flow f on a graph G with source s and sink t, an *augmenting path* is a directed path from s to t in the residual graph G_f.

Theorem (Max Flow Min Cut Theorem)

The following statements are equivalent:

1. *f is a maximum flow on G with capacities c,*
2. *there is a $s - t$ cut (A, B) with $c(A, B) = |f|$,*
3. *There exists no augmenting path in G_f.*

Theorem (Hall's Theorem)

A bipartite $n \times n$ graph $G = (U, V, E)$ has a perfect matching if and only if for any $S \subseteq U$, $|\delta(S)| \geq |S|$, where $\delta(S)$ denotes the set of nodes in V that have a neighbor in S.

Theorem (Hall's Theorem)

A bipartite $n \times n$ graph $G = (U, V, E)$ has a perfect matching if and only if for any $S \subseteq U$, $|\delta(S)| \geq |S|$, where $\delta(S)$ denotes the set of nodes in V that have a neighbor in S.

Definition

A *vertex cover* of a graph $G = (V, E)$ is a set of vertices $S \subseteq V$ such that each edge in E is incident to at least one node in S. A *minimum* vertex cover is a vertex cover of minimum cardinality.

Combinatorial Consequences of Max Flow Min Cut

Theorem (Hall's Theorem)

A bipartite $n \times n$ graph $G = (U, V, E)$ has a perfect matching if and only if for any $S \subseteq U$, $|\delta(S)| \geq |S|$, where $\delta(S)$ denotes the set of nodes in V that have a neighbor in S.

Definition

A *vertex cover* of a graph $G = (V, E)$ is a set of vertices $S \subseteq V$ such that each edge in E is incident to at least one node in S. A *minimum* vertex cover is a vertex cover of minimum cardinality.

Theorem (Kőnig-Egerváry Theorem)

In a bipartite graph, the size of a maximum matching is equal to the size of a minimum vertex cover.

Algorithms for Max Flow / Min Cut

The Ford-Fulkerson algorithm: Start with a flow f that is zero everywhere. Find an augmenting path in the residual graph G_f; let r be the minimum residual capacity along the path; augment f by a flow of value r along the path. Repeat, until no augmenting path can be found.

Algorithms for Max Flow / Min Cut

The Ford-Fulkerson algorithm: Start with a flow f that is zero everywhere. Find an augmenting path in the residual graph G_f; let r be the minimum residual capacity along the path; augment f by a flow of value r along the path. Repeat, until no augmenting path can be found.

Theorem

The Ford-Fulkerson algorithm terminates when all capacities are integers. When it terminates, the Ford-Fulkerson algorithm returns a maximum flow.

Algorithms for Max Flow / Min Cut

The Ford-Fulkerson algorithm: Start with a flow f that is zero everywhere. Find an augmenting path in the residual graph G_f; let r be the minimum residual capacity along the path; augment f by a flow of value r along the path. Repeat, until no augmenting path can be found.

Theorem

The Ford-Fulkerson algorithm terminates when all capacities are integers. When it terminates, the Ford-Fulkerson algorithm returns a maximum flow.

Claim

The Ford-Fulkerson algorithm can take exponential time to terminate.

Edmonds-Karp Algorithm

The Edmonds and Karp algorithm: When looking for an augmenting path, use one with the minimum length.

Edmonds-Karp Algorithm

The Edmonds and Karp algorithm: When looking for an augmenting path, use one with the minimum length.

Theorem

The Edmonds-Karp algorithm finds a maximum flow in time $O(m^2 n)$.

Edmonds-Karp Algorithm Analysis

Lemma

The length of the shortest augmenting path cannot decrease in the implementation of Edmonds-Karp algorithm.

Lemma

We can augment along shortest augmenting paths of the same length at most m times before the length of the shortest augmenting paths strictly increases.

Edmonds-Karp Algorithm Analysis

Lemma

The length of the shortest augmenting path cannot decrease in the implementation of Edmonds-Karp algorithm.

Lemma

We can augment along shortest augmenting paths of the same length at most m times before the length of the shortest augmenting paths strictly increases.

Definition

The *level graph* of a graph G is the directed BFS tree rooted at the source s, with the sideways and backward edges removed.

Runtime of max flow min cut algorithm

- The Edmond-Karp algorithm runs in *strongly polynomial time*.

Runtime of max flow min cut algorithm

- The Edmond-Karp algorithm runs in *strongly polynomial time*.
- The Dinitz Algorithm: In each round, use a *blocking flow* instead of just one augmenting path. Runs in time $O(mn^2)$.
  - In spirit, similar to Hopcroft and Karp's algorithm for unweighted bipartite matching.
- Other algorithms..

Application: Baseball Elimination Problem

Problem: We are given n baseball teams and the number of winning matches of each team. We are also given the number of matches in the future between each pair of teams. That is, between each pair of teams i and j, we are given m_{ij}, the number of matches that are going to be played between team i and team j. We ask, given this data, a polynomial-time algorithm to decide whether a given team (say, team 1) has no chance to win the champion no matter what happens in the future.