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 e is associated with a capacity $c_e \geq 0$.
- This is called a *flow network*.

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 e is associated with a capacity $c_e \geq 0$.
- This is called a *flow network*.

Definition

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

1. *capacity conditions* $\forall e \in E$, $0 \leq f(e) \leq c_e$.
2. *conservation conditions* $\forall u \in V \setminus \{s, t\}$,

$$\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e).$$

The *value* of a flow f is $|f| := \sum_{e \text{ out of } s} f(e)$.

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 e is associated with a capacity $c_e \geq 0$.
- This is called a *flow network*.

Definition

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

1. *capacity conditions* $\forall e \in E$, $0 \leq f(e) \leq c_e$.
2. *conservation conditions* $\forall u \in V \setminus \{s, t\}$,

$$\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e).$$

The *value* of a flow f is $|f| := \sum_{e \text{ out of } s} f(e)$.

The maximum flow problem: given a flow network, compute a flow with maximum value.

The Residual Graph

Definition

Given a flow f in a flow network G, the *residual graph* G_f is defined as follows.

- G_f has the same set of nodes as G (including s and t);
- for each $e = (u, v)$ of G on which $f(e) < c_e$, e is in G_f and has capacity $c_e - f(e)$; (such an edge is called a *forward edge* in G_f)
- for each $e = (u, v)$ of G on which $f(e) > 0$, (v, u) is in G_f, and has capacity $f(e)$. (such an edge is called a *backward edge* in G_f)

The Residual Graph

Definition

Given a flow f in a flow network G, the *residual graph* G_f is defined as follows.

- G_f has the same set of nodes as G (including s and t);
- for each $e = (u, v)$ of G on which $f(e) < c_e$, e is in G_f and has capacity $c_e - f(e)$; (such an edge is called a *forward edge* in G_f)
- for each $e = (u, v)$ of G on which $f(e) > 0$, (v, u) is in G_f, and has capacity $f(e)$. (such an edge is called a *backward edge* in G_f)

The capacities in G_f are sometimes called the *residual capacities*. Note that the residual capacities are all strictly positive.

Augmenting paths

Definition

Any (simple) s-t path in the residual graph G_f is called an *augmenting path*. In an augmenting path P in the residual graph G_f, the minimum residual capacity is called the *bottleneck*, denoted as $\text{bottleneck}(P, f)$.

Augmenting paths

Definition

Any (simple) s-t path in the residual graph G_f is called an *augmenting path*. In an augmenting path P in the residual graph G_f, the minimum residual capacity is called the *bottleneck*, denoted as $\text{bottleneck}(P, f)$.

Given a flow f in G, if there is an s-t path P in G_f, let b be $\text{bottleneck}(P, f)$; we *augment* along P by doing the following for each edge $e = (u, v)$ in P:

Augmenting paths

Definition

Any (simple) s-t path in the residual graph G_f is called an *augmenting path*. In an augmenting path P in the residual graph G_f, the minimum residual capacity is called the *bottleneck*, denoted as $\text{bottleneck}(P, f)$.

Given a flow f in G, if there is an s-t path P in G_f, let b be $\text{bottleneck}(P, f)$; we *augment* along P by doing the following for each edge $e = (u, v)$ in P:

- if e is a forward edge, then we increase $f(e)$ in G by b;

Augmenting paths

Definition

Any (simple) s-t path in the residual graph G_f is called an *augmenting path*. In an augmenting path P in the residual graph G_f, the minimum residual capacity is called the *bottleneck*, denoted as $\text{bottleneck}(P, f)$.

Given a flow f in G, if there is an s-t path P in G_f, let b be $\text{bottleneck}(P, f)$; we *augment* along P by doing the following for each edge $e = (u, v)$ in P:

- if e is a forward edge, then we increase $f(e)$ in G by b;
- if $e = (u, v)$ is a backward edge in G_f, then edge (v, u) is in G with $f((v, u)) > 0$, we decrease $f((v, u))$ in G by b.

Augmenting paths

Definition

Any (simple) s-t path in the residual graph G_f is called an *augmenting path*. In an augmenting path P in the residual graph G_f, the minimum residual capacity is called the *bottleneck*, denoted as $\text{bottleneck}(P, f)$.

Given a flow f in G, if there is an s-t path P in G_f, let b be $\text{bottleneck}(P, f)$; we *augment* along P by doing the following for each edge $e = (u, v)$ in P:

- if e is a forward edge, then we increase $f(e)$ in G by b;
- if $e = (u, v)$ is a backward edge in G_f, then edge (v, u) is in G with $f((v, u)) > 0$, we decrease $f((v, u))$ in G by b.

Proposition

The result of an augmentation, f', is a flow in G.

Ford-Fulkerson Algorithm

The Ford-Fulkerson algorithm:

- Initialize: $f(e) \leftarrow 0$ for all $e \in E$.

Ford-Fulkerson Algorithm

The Ford-Fulkerson algorithm:

- Initialize: $f(e) \leftarrow 0$ for all $e \in E$.
- Iterate: construct G_f and search for an augmenting path. If no augmenting path can be found, terminate and return $f(e)$'s as the flow. Otherwise augment along the path and repeat.

Ford-Fulkerson Algorithm

The Ford-Fulkerson algorithm:

- Initialize: $f(e) \leftarrow 0$ for all $e \in E$.
- Iterate: construct G_f and search for an augmenting path. If no augmenting path can be found, terminate and return $f(e)$'s as the flow. Otherwise augment along the path and repeat.

Running time: each round takes $O(m)$ time, but how many rounds?

Ford-Fulkerson Algorithm

The Ford-Fulkerson algorithm:

- Initialize: $f(e) \leftarrow 0$ for all $e \in E$.
- Iterate: construct G_f and search for an augmenting path. If no augmenting path can be found, terminate and return $f(e)$'s as the flow. Otherwise augment along the path and repeat.

Running time: each round takes $O(m)$ time, but how many rounds?
Suppose all capacities are integers. Let C be $\sum_{e \text{ out of } s} c_e$.

Claim

The Ford-Fulkerson algorithm terminates in at most C rounds.

Ford-Fulkerson Algorithm

The Ford-Fulkerson algorithm:

- Initialize: $f(e) \leftarrow 0$ for all $e \in E$.
- Iterate: construct G_f and search for an augmenting path. If no augmenting path can be found, terminate and return $f(e)$'s as the flow. Otherwise augment along the path and repeat.

Running time: each round takes $O(m)$ time, but how many rounds? Suppose all capacities are integers. Let C be $\sum_{e \text{ out of } s} c_e$.

Claim

The Ford-Fulkerson algorithm terminates in at most C rounds.

Running time $O(Cm)$.

Proof of Correctness

- By the proposition we showed, $f(e)$'s returned by Ford-Fulkerson is indeed a flow.

Proof of Correctness

- By the proposition we showed, $f(e)$'s returned by Ford-Fulkerson is indeed a flow.
- It remains to show that it is a maximum flow.

Proof of Correctness

- By the proposition we showed, $f(e)$'s returned by Ford-Fulkerson is indeed a flow.
- It remains to show that it is a maximum flow.
- A "primal-dual" argument: give many upper bounds on the value of any flow, and then show that the flow returned by Ford-Fulkerson is equal to one of these bounds.

Proof of Correctness

- By the proposition we showed, $f(e)$'s returned by Ford-Fulkerson is indeed a flow.
- It remains to show that it is a maximum flow.
- A "primal-dual" argument: give many upper bounds on the value of any flow, and then show that the flow returned by Ford-Fulkerson is equal to one of these bounds.
- As a corollary, this will also show that maximum "primal" value (flow here) is equal to the minimum "dual" value (the best upper bound here).

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.
- In a flow network, the capacity of a cut (A, B) is

$$c(A, B) := \sum_{e \text{ out of } A} c_e.$$

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.
- In a flow network, the capacity of a cut (A, B) is

$$c(A, B) := \sum_{e \text{ out of } A} c_e.$$

- Given a flow f and an s-t cut (A, B),

$$f^{\text{out}}(A) := \sum_{e \text{ out of } A} f(e), \quad f^{\text{in}}(A) := \sum_{e \text{ into } A} f(e).$$

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.
- In a flow network, the capacity of a cut (A, B) is

$$c(A, B) := \sum_{e \text{ out of } A} c_e.$$

- Given a flow f and an s-t cut (A, B),

$$f^{\text{out}}(A) := \sum_{e \text{ out of } A} f(e), \quad f^{\text{in}}(A) := \sum_{e \text{ into } A} f(e).$$

Lemma

For any s-t cut (A, B) and any flow f, the value of f, $|f|$, is $f^{out}(A) - f^{in}(A)$.

The Max Flow Min Cut Theorem

Corollary

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

The Max Flow Min Cut Theorem

Corollary

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

Theorem (Max-Flow Min-Cut Theorem)

The following statements are equivalent:

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

The Max Flow Min Cut Theorem

Corollary

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

Theorem (Max-Flow Min-Cut Theorem)

The following statements are equivalent:

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

- This immediately implies the correctness of Ford-Fulkerson algorithm.

The Max Flow Min Cut Theorem

Corollary

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

Theorem (Max-Flow Min-Cut Theorem)

The following statements are equivalent:

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

- This immediately implies the correctness of Ford-Fulkerson algorithm.
- The s-t cut (A, B) with $c(A, B) = |f|$ must have the minimum capacity among all s-t cuts, and hence is called a *minimum cut*.

Some other immediate consequences

- For flow networks with integer capacities, there is always an inter-valued maximum flow.

Some other immediate consequences

- For flow networks with integer capacities, there is always an inter-valued maximum flow.
  - This is an example where an algorithm has an implication that doesn't look algorithmic.

Some other immediate consequences

- For flow networks with integer capacities, there is always an inter-valued maximum flow.
  - This is an example where an algorithm has an implication that doesn't look algorithmic.
- For all flow networks (even whose capacities are not integral), a maximum flow exists.

Some other immediate consequences

- For flow networks with integer capacities, there is always an inter-valued maximum flow.
  - This is an example where an algorithm has an implication that doesn't look algorithmic.
- For all flow networks (even whose capacities are not integral), a maximum flow exists.
- Observation: given a maximum flow in a flow network, it takes additional $O(m)$ time to find a minimum cut.