Network Flows

Lecture 17
March 19, 2015

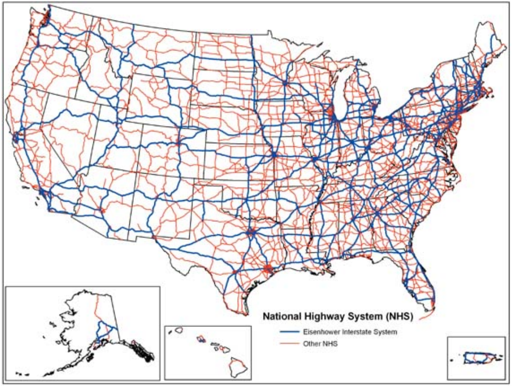
Everything flows

Panta rei – everything flows (literally).
Heraclitus (535–475 BC)

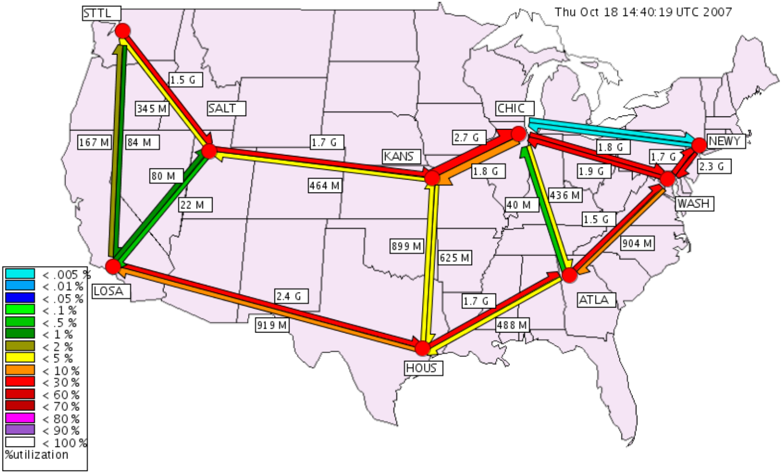
Part I

Network Flows: Introduction and Setup

Transportation/Road Network



Internet Backbone Network



Common Features of Flow Networks

1. **Network** represented by a (directed) *graph* $\mathbf{G = (V, E)}$.
2. Each edge $\mathbf{e}$ has a **capacity** $\mathbf{c(e) \geq 0}$ that limits amount of *traffic* on $\mathbf{e}$.
3. *Source(s)* of traffic/data.
4. *Sink(s)* of traffic/data.
5. Traffic *flows* from sources to sinks.
6. Traffic is *switched/interchanged* at nodes.
7. **Flow** abstract term to indicate stuff (traffic/data/etc) that **flows** from sources to sinks.

Common Features of Flow Networks

1. **Network** represented by a (directed) *graph* $G = (V, E)$.
2. Each edge e has a **capacity** $c(e) \geq 0$ that limits amount of *traffic* on e.
3. *Source(s)* of traffic/data.
4. *Sink(s)* of traffic/data.
5. Traffic *flows* from sources to sinks.
6. Traffic is *switched/interchanged* at nodes.
7. **Flow** abstract term to indicate stuff (traffic/data/etc) that **flows** from sources to sinks.

Common Features of Flow Networks

1. **Network** represented by a (directed) *graph* $G = (V, E)$.
2. Each edge e has a **capacity** $c(e) \geq 0$ that limits amount of *traffic* on e.
3. *Source(s)* of traffic/data.
4. *Sink(s)* of traffic/data.
5. Traffic *flows* from sources to sinks.
6. Traffic is *switched/interchanged* at nodes.
7. **Flow** abstract term to indicate stuff (traffic/data/etc) that **flows** from sources to sinks.

Common Features of Flow Networks

1. **Network** represented by a (directed) *graph* $G = (V, E)$.
2. Each edge e has a **capacity** $c(e) \geq 0$ that limits amount of *traffic* on e.
3. *Source(s)* of traffic/data.
4. *Sink(s)* of traffic/data.
5. Traffic *flows* from sources to sinks.
6. Traffic is *switched/interchanged* at nodes.
7. **Flow** abstract term to indicate stuff (traffic/data/etc) that **flows** from sources to sinks.

Common Features of Flow Networks

1. **Network** represented by a (directed) *graph* $G = (V, E)$.
2. Each edge e has a **capacity** $c(e) \geq 0$ that limits amount of *traffic* on e.
3. *Source(s)* of traffic/data.
4. *Sink(s)* of traffic/data.
5. Traffic *flows* from sources to sinks.
6. Traffic is *switched/interchanged* at nodes.
7. **Flow** abstract term to indicate stuff (traffic/data/etc) that **flows** from sources to sinks.

Common Features of Flow Networks

1. **Network** represented by a (directed) *graph* $G = (V, E)$.
2. Each edge e has a **capacity** $c(e) \geq 0$ that limits amount of *traffic* on e.
3. *Source(s)* of traffic/data.
4. *Sink(s)* of traffic/data.
5. Traffic *flows* from sources to sinks.
6. Traffic is *switched/interchanged* at nodes.
7. **Flow** abstract term to indicate stuff (traffic/data/etc) that **flows** from sources to sinks.

Common Features of Flow Networks

1. **Network** represented by a (directed) *graph* $G = (V, E)$.
2. Each edge e has a **capacity** $c(e) \geq 0$ that limits amount of *traffic* on e.
3. *Source(s)* of traffic/data.
4. *Sink(s)* of traffic/data.
5. Traffic *flows* from sources to sinks.
6. Traffic is *switched/interchanged* at nodes.
7. **Flow** abstract term to indicate stuff (traffic/data/etc) that **flows** from sources to sinks.

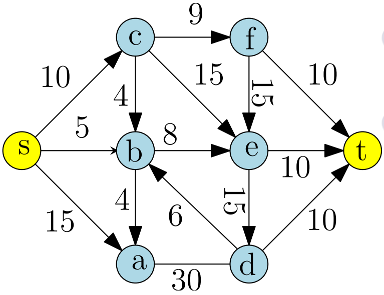
Common Features of Flow Networks

1. **Network** represented by a (directed) *graph* $G = (V, E)$.
2. Each edge e has a **capacity** $c(e) \geq 0$ that limits amount of *traffic* on e.
3. *Source(s)* of traffic/data.
4. *Sink(s)* of traffic/data.
5. Traffic *flows* from sources to sinks.
6. Traffic is *switched/interchanged* at nodes.
7. **Flow** abstract term to indicate stuff (traffic/data/etc) that **flows** from sources to sinks.

Single Source/Single Sink Flows

Simple setting:

1. Single source s and single sink t.
2. Every other node v is an **internal** node.
3. Flow originates at s and terminates at t.



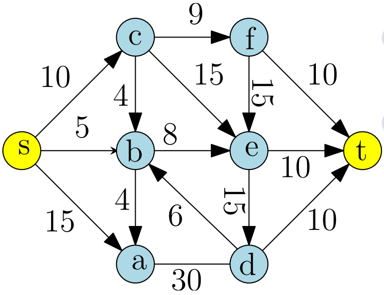
1. Each edge e has a capacity $c(e) \geq 0$.
2. Sometimes assume: Source $s \in V$ has no incoming edges, and sink $t \in V$ has no outgoing edges.

1. Assumptions: All capacities are integer, and every vertex has at least one edge incident to it.

Single Source/Single Sink Flows

Simple setting:

1. Single source s and single sink t.
2. Every other node v is an **internal** node.
3. Flow originates at s and terminates at t.



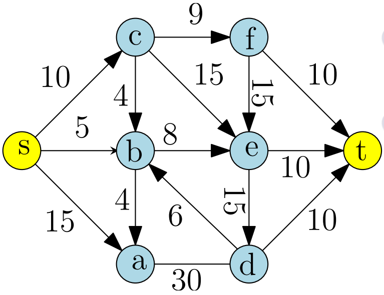
1. Each edge e has a capacity $c(e) \geq 0$.
2. Sometimes assume: Source $s \in V$ has no incoming edges, and sink $t \in V$ has no outgoing edges.

1. Assumptions: All capacities are integer, and every vertex has at least one edge incident to it.

Single Source/Single Sink Flows

Simple setting:

1. Single source s and single sink t.
2. Every other node v is an **internal** node.
3. Flow originates at s and terminates at t.



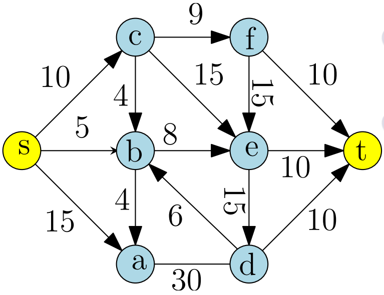
1. Each edge e has a capacity $c(e) \geq 0$.
2. Sometimes assume: Source $s \in V$ has no incoming edges, and sink $t \in V$ has no outgoing edges.

1. Assumptions: All capacities are integer, and every vertex has at least one edge incident to it.

Single Source/Single Sink Flows

Simple setting:

1. Single source s and single sink t.
2. Every other node v is an **internal** node.
3. Flow originates at s and terminates at t.



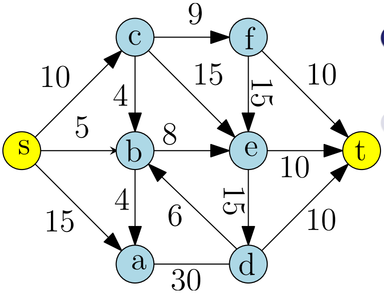
1. Each edge e has a capacity $c(e) \geq 0$.
2. Sometimes assume: Source $s \in V$ has no incoming edges, and sink $t \in V$ has no outgoing edges.

1. Assumptions: All capacities are integer, and every vertex has at least one edge incident to it.

Single Source/Single Sink Flows

Simple setting:

1. Single source s and single sink t.
2. Every other node v is an **internal** node.
3. Flow originates at s and terminates at t.



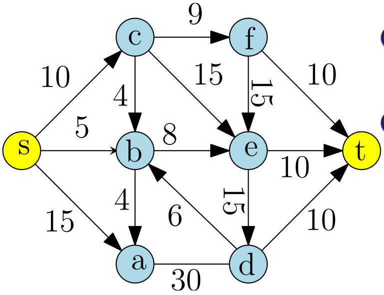
1. Each edge e has a capacity $c(e) \geq 0$.
2. Sometimes assume: Source $s \in V$ has no incoming edges, and sink $t \in V$ has no outgoing edges.

1. Assumptions: All capacities are integer, and every vertex has at least one edge incident to it.

Single Source/Single Sink Flows

Simple setting:

1. Single source s and single sink t.
2. Every other node v is an **internal** node.
3. Flow originates at s and terminates at t.



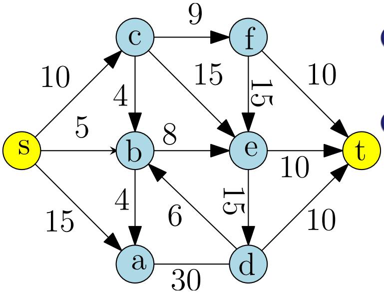
1. Each edge e has a capacity $c(e) \geq 0$.
2. Sometimes assume: Source $s \in V$ has no incoming edges, and sink $t \in V$ has no outgoing edges.

1. Assumptions: All capacities are integer, and every vertex has at least one edge incident to it.

Single Source/Single Sink Flows

Simple setting:

1. Single source s and single sink t.
2. Every other node v is an **internal** node.
3. Flow originates at s and terminates at t.



1. Each edge e has a capacity $c(e) \geq 0$.
2. Sometimes assume:
   Source $s \in V$ has no incoming edges, and sink $t \in V$ has no outgoing edges.

1. Assumptions: All capacities are integer, and every vertex has at least one edge incident to it.

Definition of Flow

1. Two ways to define flows...
2. edge based, or
3. path based.
4. Essentially equivalent but have different uses.
5. Edge based definition is more compact.

Definition of Flow

1. Two ways to define flows...
2. edge based, or
3. path based.
4. Essentially equivalent but have different uses.
5. Edge based definition is more compact.

Definition of Flow

1. Two ways to define flows...
2. edge based, or
3. path based.
4. Essentially equivalent but have different uses.
5. Edge based definition is more compact.

Definition of Flow

1. Two ways to define flows...
2. edge based, or
3. path based.
4. Essentially equivalent but have different uses.
5. Edge based definition is more compact.

Definition of Flow

1. Two ways to define flows...
2. edge based, or
3. path based.
4. Essentially equivalent but have different uses.
5. Edge based definition is more compact.

Definition of Flow

1. Two ways to define flows...
2. edge based, or
3. path based.
4. Essentially equivalent but have different uses.
5. Edge based definition is more compact.

Edge Based Definition of Flow

Definition

Flow in network $G = (V, E)$, is function $f : E \to \mathbb{R}^{\geq 0}$ s.t.

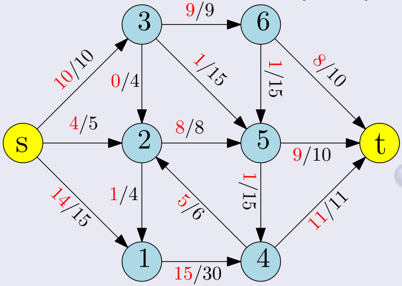


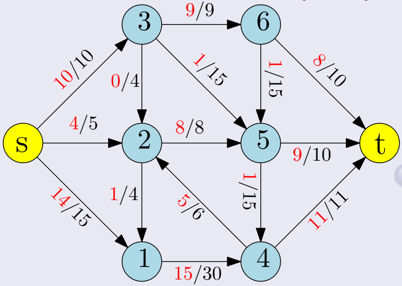
Figure: Flow with value.

1. Capacity Constraint: For each edge e, $f(e) \leq c(e)$.

Edge Based Definition of Flow

Definition

Flow in network $G = (V, E)$, is function $f : E \to \mathbb{R}^{\geq 0}$ s.t.



1. Capacity Constraint: For each edge e, $f(e) \leq c(e)$.

Figure: Flow with value.

Edge Based Definition of Flow

Definition

Flow in network $G = (V, E)$, is function $f : E \rightarrow \mathbb{R}^{\geq 0}$ s.t.

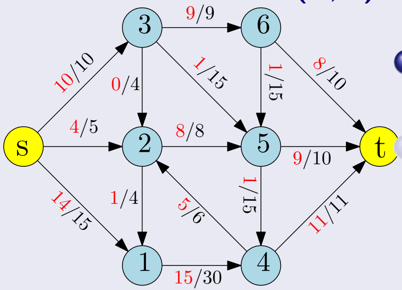


Figure: Flow with value.

1. Capacity Constraint: For each edge e, $f(e) \leq c(e)$.
2. Conservation Constraint: For each vertex $v \neq s, t$.

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

Edge Based Definition of Flow

Definition

Flow in network $G = (V, E)$, is function $f : E \rightarrow \mathbb{R}^{\geq 0}$ s.t.

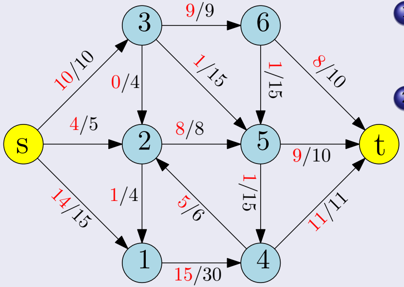


Figure: Flow with value.

1. Capacity Constraint: For each edge e, $f(e) \leq c(e)$.
2. Conservation Constraint: For each vertex $v \neq s, t$.

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

3. Value of flow= (total flow out of source) − (total flow in to source).

Definition

Flow in network $G = (V, E)$, is function $f : E \rightarrow \mathbb{R}^{\geq 0}$ s.t.

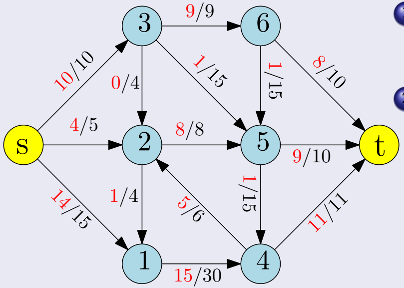


Figure: Flow with value.

1. Capacity Constraint: For each edge e, $f(e) \leq c(e)$.
2. Conservation Constraint: For each vertex $v \neq s, t$.
$$\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$$
3. Value of flow= (total flow out of source) − (total flow in to source).

Flow...

Conservation of flow law is also known as **Kirchhoff's law**.

More Definitions and Notation

Notation

1. The inflow into a vertex v is $f^{\text{in}}(v) = \sum_{e \text{ into } v} f(e)$ and the outflow is $f^{\text{out}}(v) = \sum_{e \text{ out of } v} f(e)$
2. For a set of vertices A, $f^{\text{in}}(A) = \sum_{e \text{ into } A} f(e)$. Outflow $f^{\text{out}}(A)$ is defined analogously

Definition

For a network $G = (V, E)$ with source s, the **value** of flow f is defined as $v(f) = f^{\text{out}}(s) - f^{\text{in}}(s)$.

More Definitions and Notation

Notation

1. The inflow into a vertex v is $f^{\text{in}}(v) = \sum_{e \text{ into } v} f(e)$ and the outflow is $f^{\text{out}}(v) = \sum_{e \text{ out of } v} f(e)$
2. For a set of vertices A, $f^{\text{in}}(A) = \sum_{e \text{ into } A} f(e)$. Outflow $f^{\text{out}}(A)$ is defined analogously

Definition

For a network $G = (V, E)$ with source s, the **value** of flow f is defined as $v(f) = f^{\text{out}}(s) - f^{\text{in}}(s)$.

More Definitions and Notation

Notation

1. The inflow into a vertex v is $f^{\text{in}}(v) = \sum_{e \text{ into } v} f(e)$ and the outflow is $f^{\text{out}}(v) = \sum_{e \text{ out of } v} f(e)$
2. For a set of vertices A, $f^{\text{in}}(A) = \sum_{e \text{ into } A} f(e)$. Outflow $f^{\text{out}}(A)$ is defined analogously

Definition

For a network $G = (V, E)$ with source s, the **value** of flow f is defined as $v(f) = f^{\text{out}}(s) - f^{\text{in}}(s)$.

A Path Based Definition of Flow

Intuition: Flow goes from source s to sink t along a path.

$\mathcal{P}$: set of all paths from s to t. $|\mathcal{P}|$ can be **exponential** in n.

Definition (Flow by paths.)

A **flow** in network $G = (V, E)$, is function $f : \mathcal{P} \rightarrow \mathbb{R}^{\geq 0}$ s.t.

1. Capacity Constraint: For each edge e, total flow on e is $\leq c(e)$.

$$\sum_{p \in \mathcal{P} : e \in p} f(p) \leq c(e)$$

2. Conservation Constraint: No need! Automatic.

Value of flow: $\sum_{p \in \mathcal{P}} f(p)$.

A Path Based Definition of Flow

Intuition: Flow goes from source s to sink t along a path.

$\mathcal{P}$: set of all paths from s to t. $|\mathcal{P}|$ can be **exponential** in n.

Definition (Flow by paths.)

A **flow** in network $G = (V, E)$, is function $f : \mathcal{P} \rightarrow \mathbb{R}^{\geq 0}$ s.t.

1. Capacity Constraint: For each edge e, total flow on e is $\leq c(e)$.

$$\sum_{p \in \mathcal{P} : e \in p} f(p) \leq c(e)$$

2. Conservation Constraint: No need! Automatic.

Value of flow: $\sum_{p \in \mathcal{P}} f(p)$.

A Path Based Definition of Flow

Intuition: Flow goes from source s to sink t along a path.

$\mathcal{P}$: set of all paths from s to t. $|\mathcal{P}|$ can be **exponential** in n.

Definition (Flow by paths.)

A **flow** in network $G = (V, E)$, is function $f : \mathcal{P} \to \mathbb{R}^{\geq 0}$ s.t.

1. Capacity Constraint: For each edge e, total flow on e is $\leq c(e)$.

$$\sum_{p \in \mathcal{P} : e \in p} f(p) \leq c(e)$$

2. Conservation Constraint: No need! Automatic.

Value of flow: $\sum_{p \in \mathcal{P}} f(p)$.

A Path Based Definition of Flow

Intuition: Flow goes from source s to sink t along a path.

$\mathcal{P}$: set of all paths from s to t. $|\mathcal{P}|$ can be **exponential** in n.

Definition (Flow by paths.)

A **flow** in network $G = (V, E)$, is function $f : \mathcal{P} \rightarrow \mathbb{R}^{\geq 0}$ s.t.

1. Capacity Constraint: For each edge e, total flow on e is $\leq c(e)$.

$$\sum_{p \in \mathcal{P} : e \in p} f(p) \leq c(e)$$

2. Conservation Constraint: No need! Automatic.

Value of flow: $\sum_{p \in \mathcal{P}} f(p)$.

A Path Based Definition of Flow

Intuition: Flow goes from source s to sink t along a path.

$\mathcal{P}$: set of all paths from s to t. $|\mathcal{P}|$ can be **exponential** in n.

Definition (Flow by paths.)

A **flow** in network $G = (V, E)$, is function $f : \mathcal{P} \rightarrow \mathbb{R}^{\geq 0}$ s.t.

1. Capacity Constraint: For each edge e, total flow on e is $\leq c(e)$.

$$\sum_{p \in \mathcal{P}: e \in p} f(p) \leq c(e)$$

2. Conservation Constraint: No need! Automatic.

Value of flow: $\sum_{p \in \mathcal{P}} f(p)$.

A Path Based Definition of Flow

Intuition: Flow goes from source s to sink t along a path.

$\mathcal{P}$: set of all paths from s to t. $|\mathcal{P}|$ can be **exponential** in n.

Definition (Flow by paths.)

A **flow** in network $G = (V, E)$, is function $f : \mathcal{P} \to \mathbb{R}^{\geq 0}$ s.t.

1. Capacity Constraint: For each edge e, total flow on e is $\leq c(e)$.

$$\sum_{p \in \mathcal{P} : e \in p} f(p) \leq c(e)$$

2. Conservation Constraint: No need! Automatic.

Value of flow: $\sum_{p \in \mathcal{P}} f(p)$.

A Path Based Definition of Flow

Intuition: Flow goes from source s to sink t along a path.

$\mathcal{P}$: set of all paths from s to t. $|\mathcal{P}|$ can be **exponential** in n.

Definition (Flow by paths.)

A **flow** in network $G = (V, E)$, is function $f : \mathcal{P} \to \mathbb{R}^{\geq 0}$ s.t.

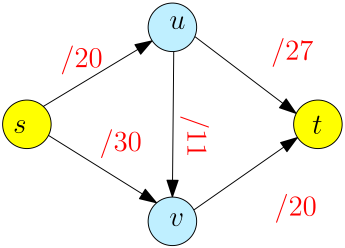
1. Capacity Constraint: For each edge e, total flow on e is $\leq c(e)$.

$$\sum_{p \in \mathcal{P} : e \in p} f(p) \leq c(e)$$

2. Conservation Constraint: No need! Automatic.

Value of flow: $\sum_{p \in \mathcal{P}} f(p)$.

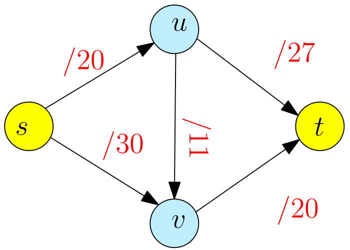
Example



$\mathcal{P} = \{p_1, p_2, p_3\}$
$p_1 : s \rightarrow u \rightarrow t$
$p_2 : s \rightarrow u \rightarrow v \rightarrow t$
$p_3 : s \rightarrow v \rightarrow t$

$f(p_1) = 10, f(p_2) = 4, f(p_3) = 6$

Example



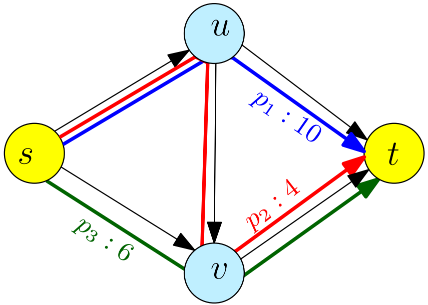
$\mathcal{P} = \{p_1, p_2, p_3\}$

$p_1 : s \rightarrow u \rightarrow t$

$p_2 : s \rightarrow u \rightarrow v \rightarrow t$

$p_3 : s \rightarrow v \rightarrow t$

$f(p_1) = 10, f(p_2) = 4, f(p_3) = 6$



Path based flow implies edge based flow

Lemma

Given a path based flow $f : \mathcal{P} \rightarrow \mathbb{R}^{\geq 0}$ *there is an edge based flow* $f' : E \rightarrow \mathbb{R}^{\geq 0}$ *of the same value.*

Proof.

For each edge e define $f'(e) = \sum_{p: e \in p} f(p)$.

Exercise: Verify capacity and conservation constraints for f'.

Exercise: Verify that value of f and f' are equal □

Path based flow implies edge based flow

Lemma

Given a path based flow $f : \mathcal{P} \rightarrow \mathbb{R}^{\geq 0}$ there is an edge based flow $f' : E \rightarrow \mathbb{R}^{\geq 0}$ of the same value.

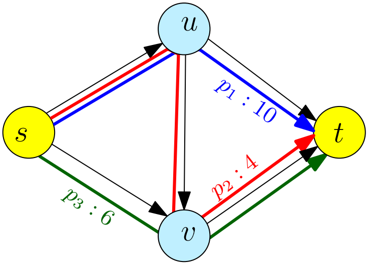
Proof.

For each edge e define $f'(e) = \sum_{p:e \in p} f(p)$.

Exercise: Verify capacity and conservation constraints for f'.

Exercise: Verify that value of f and f' are equal □

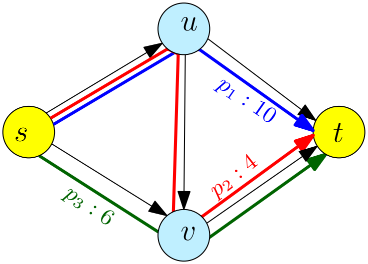
Example



$\mathcal{P} = \{p_1, p_2, p_3\}$
$p_1 : s \to u \to t$
$p_2 : s \to u \to v \to t$
$p_3 : s \to v \to t$

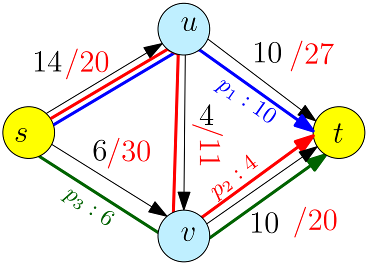
$f(p_1) = 10, f(p_2) = 4, f(p_3) = 6$

Example



$\mathcal{P} = \{p_1, p_2, p_3\}$
$p_1 : s \to u \to t$
$p_2 : s \to u \to v \to t$
$p_3 : s \to v \to t$

$f(p_1) = 10, f(p_2) = 4, f(p_3) = 6$



$f'(s \to u) = 14$
$f'(u \to v) = 4$
$f'(s \to v) = 6$
$f'(u \to t) = 10$
$f'(v \to t) = 10$

Flow Decomposition

Edge based flow to Path based Flow

Lemma

Given an edge based flow $f_1 : E \rightarrow \mathbb{R}^{\geq 0}$, there is a path based flow $f : \mathcal{P} \rightarrow \mathbb{R}^{\geq 0}$ of same value. Moreover, f assigns non-negative flow to at most m paths where $|E| = m$ and $|V| = n$. Given f_1, the path based flow can be computed in $O(mn)$ time.

Flow Decomposition

Edge based flow to Path based Flow

Proof Idea.

1. Remove all edges with $f_1(e) = 0$.
2. Find a path p from s to t.
3. Assign $f(p)$ to be $\min_{e \in p} f_1(e)$.
4. Reduce $f_1(e)$ for all $e \in p$ by $f(p)$.
5. Repeat until no path from s to t.
6. In each iteration at least on edge has flow reduced to zero.
7. Hence, at most m iterations. Can be implemented in $O(m(m+n))$ time. $O(mn)$ time requires care. □

Flow Decomposition

Edge based flow to Path based Flow

Proof Idea.

1. Remove all edges with $f_1(e) = 0$.
2. Find a path p from s to t.
3. Assign $f(p)$ to be $\min_{e \in p} f_1(e)$.
4. Reduce $f_1(e)$ for all $e \in p$ by $f(p)$.
5. Repeat until no path from s to t.
6. In each iteration at least on edge has flow reduced to zero.
7. Hence, at most m iterations. Can be implemented in $O(m(m+n))$ time. $O(mn)$ time requires care.

□

Flow Decomposition

Edge based flow to Path based Flow

Proof Idea.

1. Remove all edges with $f_1(e) = 0$.
2. Find a path p from s to t.
3. Assign $f(p)$ to be $\min_{e \in p} f_1(e)$.
4. Reduce $f_1(e)$ for all $e \in p$ by $f(p)$.
5. Repeat until no path from s to t.
6. In each iteration at least on edge has flow reduced to zero.
7. Hence, at most m iterations. Can be implemented in $O(m(m+n))$ time. $O(mn)$ time requires care. □

Flow Decomposition

Edge based flow to Path based Flow

Proof Idea.

1. Remove all edges with $f_1(e) = 0$.
2. Find a path p from s to t.
3. Assign $f(p)$ to be $\min_{e \in p} f_1(e)$.
4. Reduce $f_1(e)$ for all $e \in p$ by $f(p)$.
5. Repeat until no path from s to t.
6. In each iteration at least on edge has flow reduced to zero.
7. Hence, at most m iterations. Can be implemented in $O(m(m+n))$ time. $O(mn)$ time requires care.

Flow Decomposition

Edge based flow to Path based Flow

Proof Idea.

1. Remove all edges with $f_1(e) = 0$.
2. Find a path p from s to t.
3. Assign $f(p)$ to be $\min_{e \in p} f_1(e)$.
4. Reduce $f_1(e)$ for all $e \in p$ by $f(p)$.
5. Repeat until no path from s to t.
6. In each iteration at least on edge has flow reduced to zero.
7. Hence, at most m iterations. Can be implemented in $O(m(m + n))$ time. $O(mn)$ time requires care.

□

Flow Decomposition

Edge based flow to Path based Flow

Proof Idea.

1. Remove all edges with $f_1(e) = 0$.
2. Find a path p from s to t.
3. Assign $f(p)$ to be $\min_{e \in p} f_1(e)$.
4. Reduce $f_1(e)$ for all $e \in p$ by $f(p)$.
5. Repeat until no path from s to t.
6. In each iteration at least on edge has flow reduced to zero.
7. Hence, at most m iterations. Can be implemented in $O(m(m+n))$ time. $O(mn)$ time requires care.

□

Flow Decomposition

Edge based flow to Path based Flow

Proof Idea.

1. Remove all edges with $f_1(e) = 0$.
2. Find a path p from s to t.
3. Assign $f(p)$ to be $\min_{e \in p} f_1(e)$.
4. Reduce $f_1(e)$ for all $e \in p$ by $f(p)$.
5. Repeat until no path from s to t.
6. In each iteration at least on edge has flow reduced to zero.
7. Hence, at most m iterations. Can be implemented in $O(m(m+n))$ time. $O(mn)$ time requires care. □

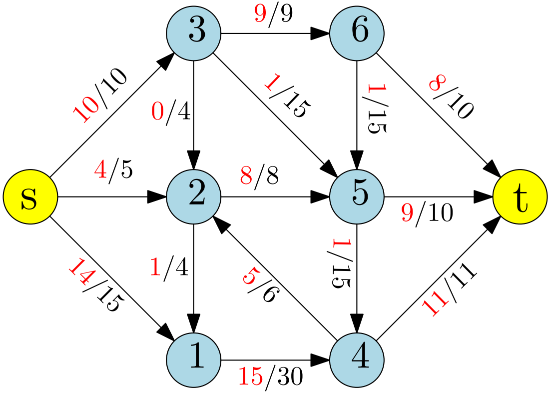
Flow Decomposition

Edge based flow to Path based Flow

Proof Idea.

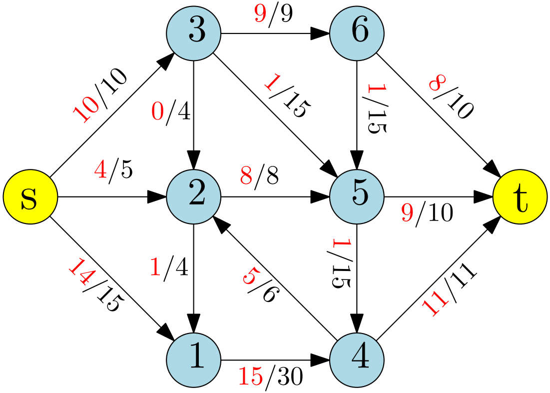
1. Remove all edges with $f_1(e) = 0$.
2. Find a path p from s to t.
3. Assign $f(p)$ to be $\min_{e \in p} f_1(e)$.
4. Reduce $f_1(e)$ for all $e \in p$ by $f(p)$.
5. Repeat until no path from s to t.
6. In each iteration at least on edge has flow reduced to zero.
7. Hence, at most m iterations. Can be implemented in $O(m(m+n))$ time. $O(mn)$ time requires care. □

Example



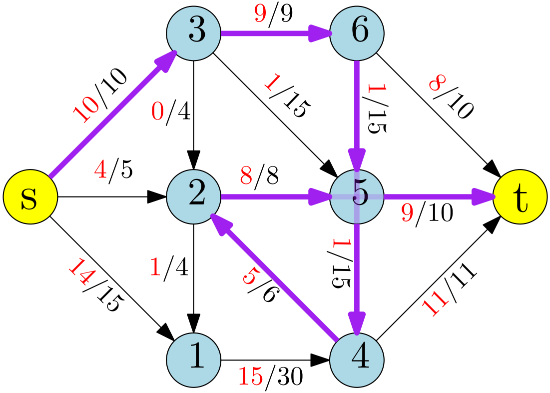
s
t
1
2
3
4
5
6
9/9
10/10
0/4
1/15
1/15
8/10
4/5
8/8
9/10
1/4
5/6
1/15
11/11
14/15
15/30

Example



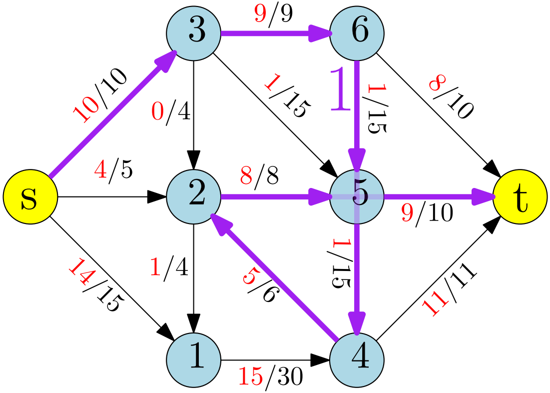
3
6
s
2
5
t
1
4
9/9
10/10
0/4
1/15
1/15
8/10
4/5
8/8
9/10
14/15
1/4
5/6
1/15
11/11
15/30

Example



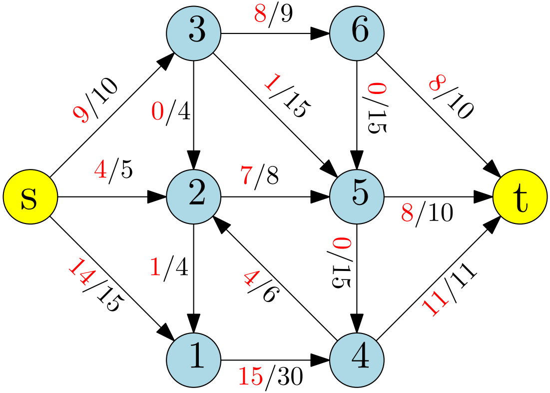
3
6
9/9
10/10
0/4
1/15
1/15
8/10
s
2
5
t
4/5
8/8
9/10
14/15
1/4
5/6
1/15
11/11
1
4
15/30

Example



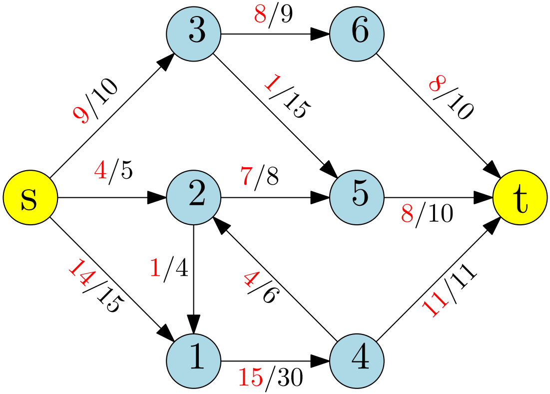
s
3
6
2
5
t
1
4
9/9
10/10
0/4
1/15
1
1/15
8/10
4/5
8/8
9/10
14/15
1/4
5/6
1/15
11/11
15/30

Example

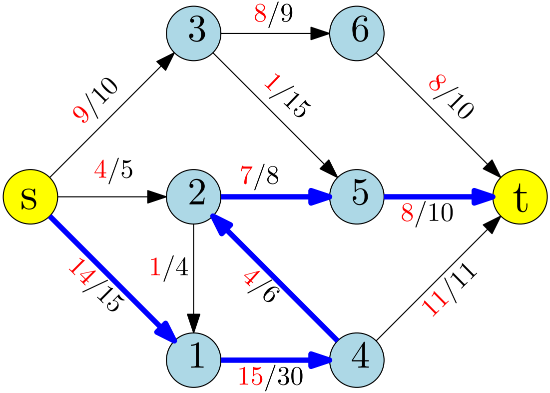


3
6
s
2
5
t
1
4
8/9
9/10
0/4
1/15
0/15
8/10
4/5
7/8
8/10
14/15
1/4
4/6
0/15
11/11
15/30

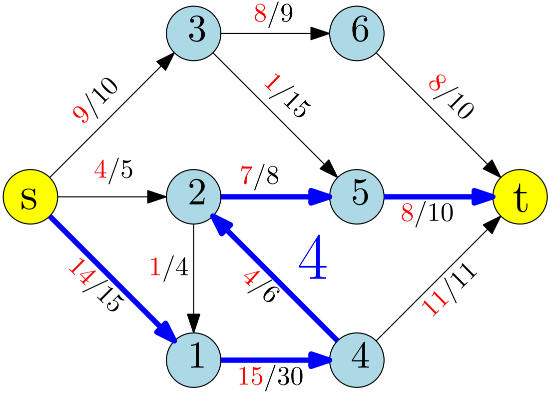
Example



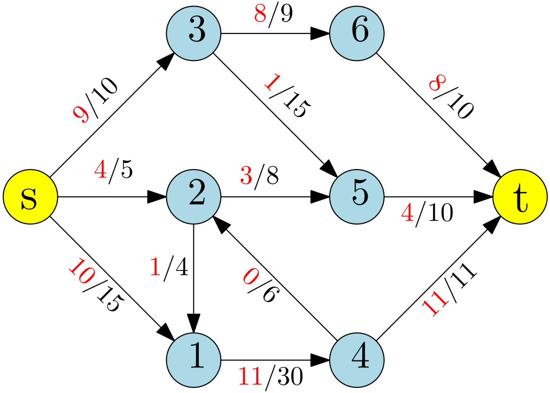
Example



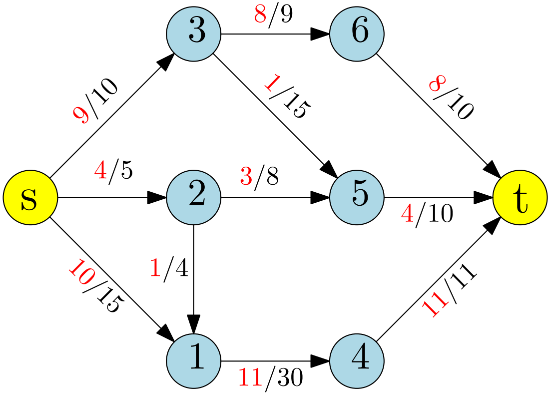
Example



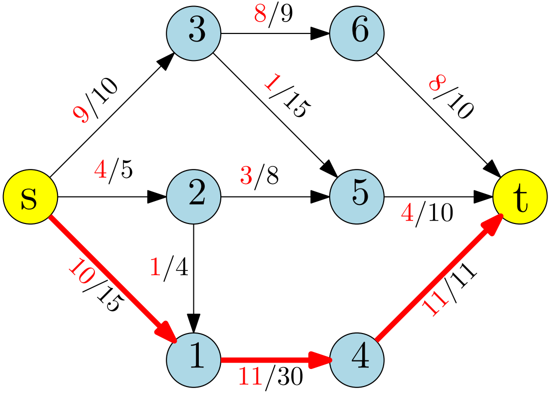
Example



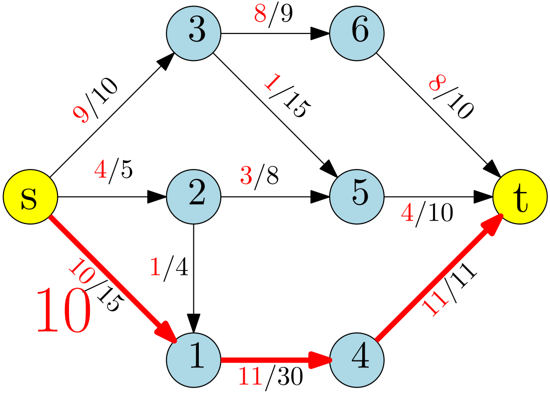
Example



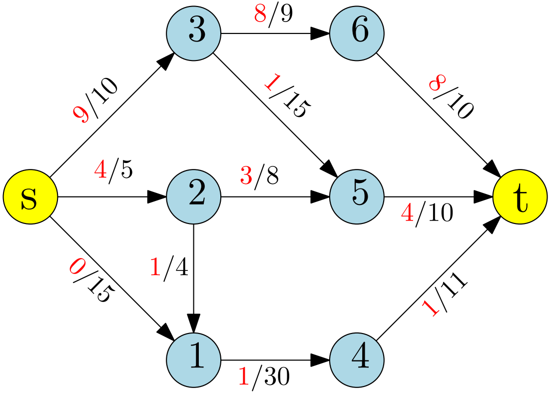
Example



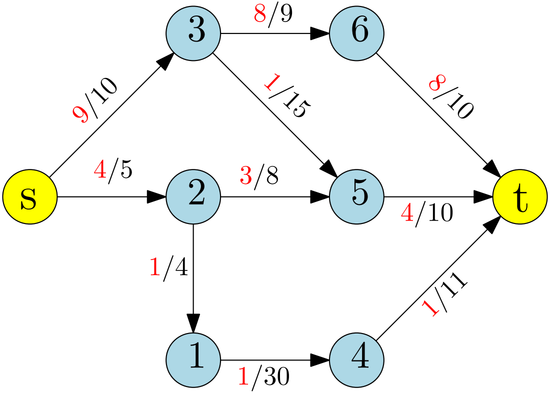
Example



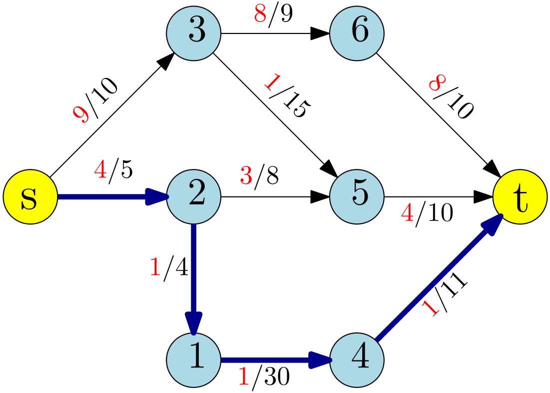
Example



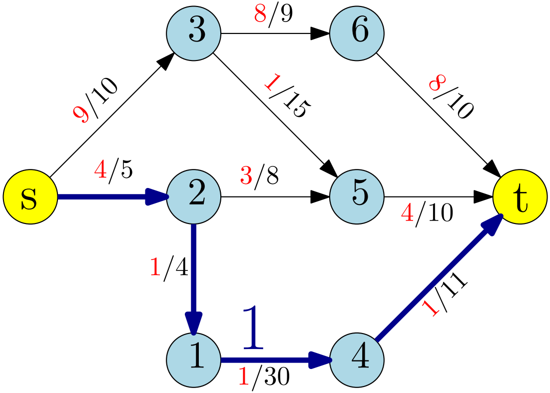
Example



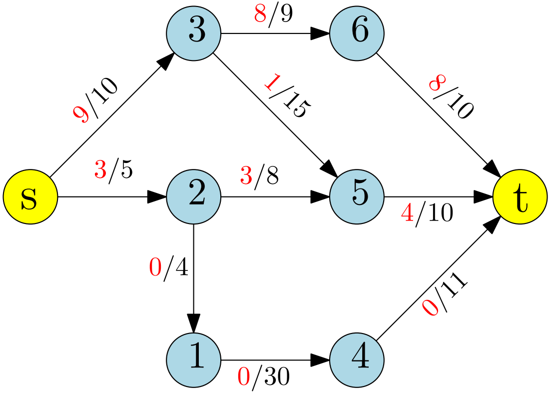
Example



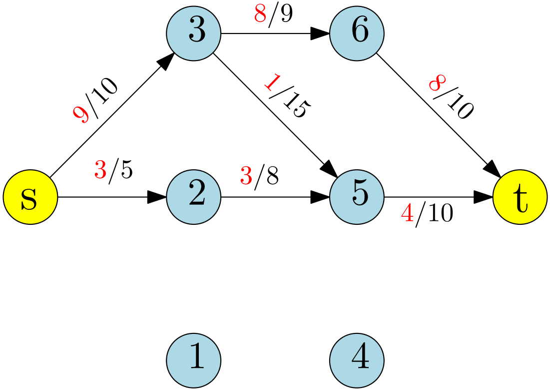
Example



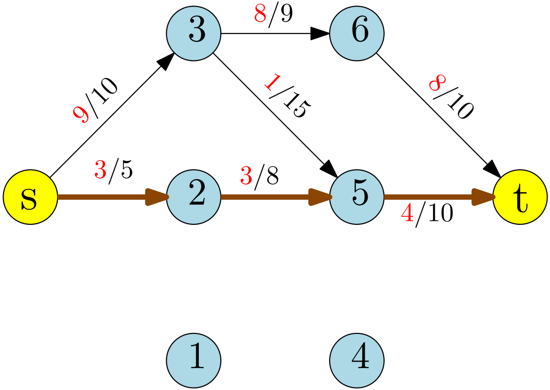
Example



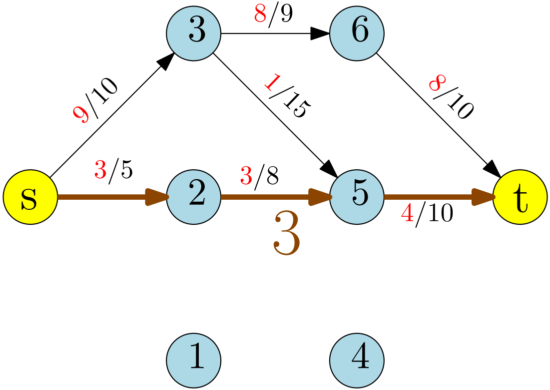
Example



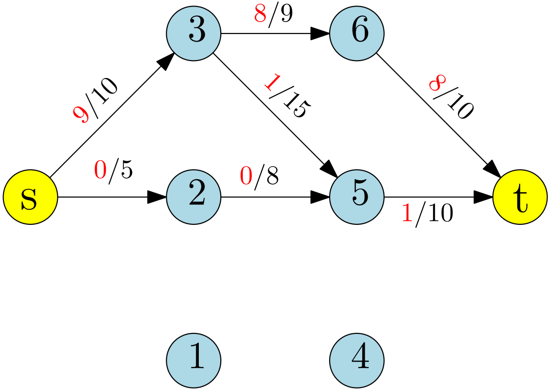
Example



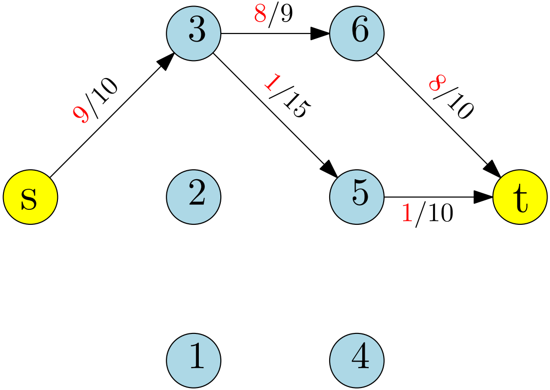
Example



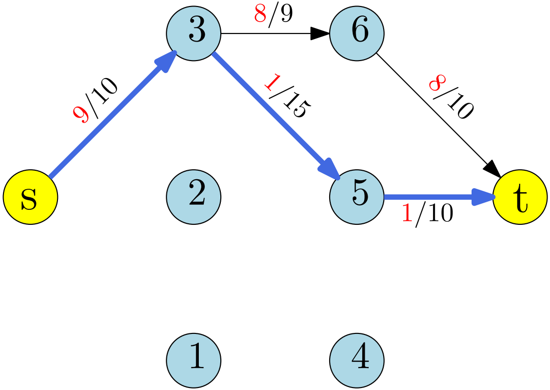
Example



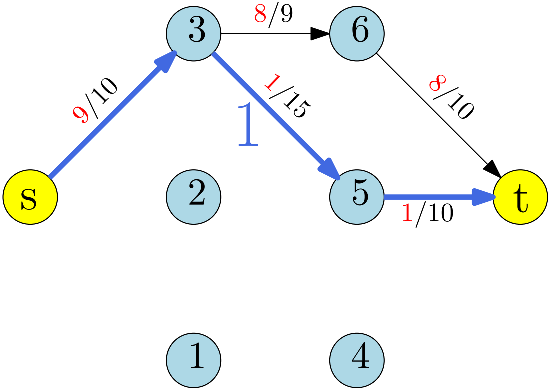
Example



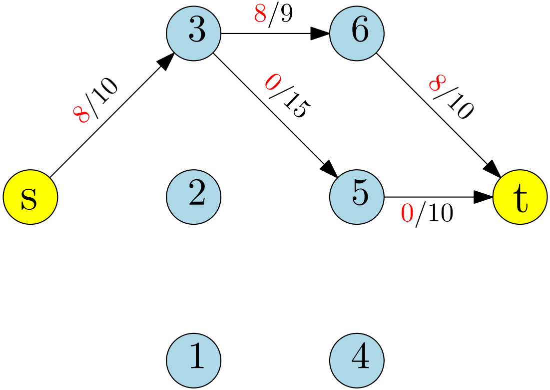
Example



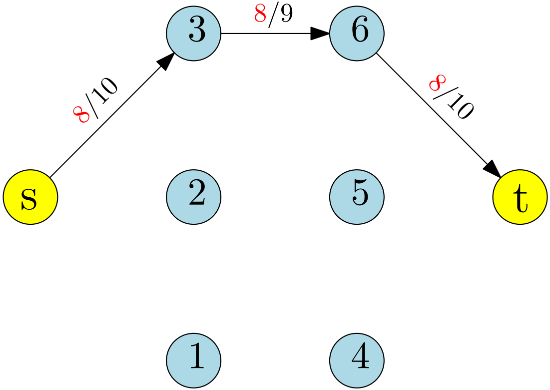
Example



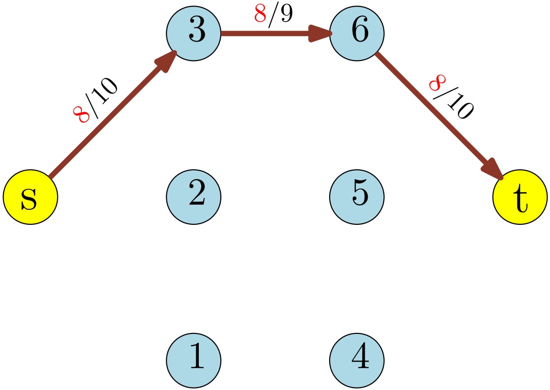
Example



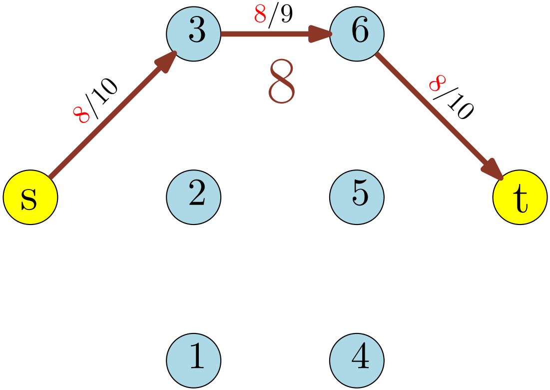
Example



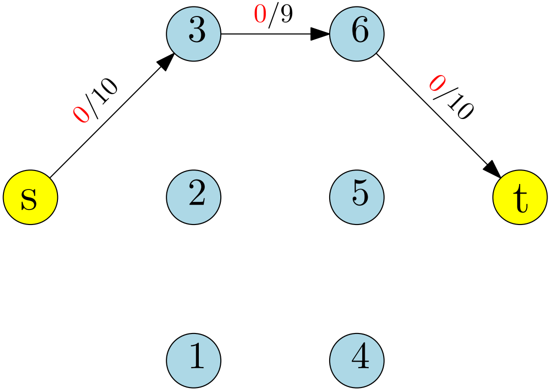
Example



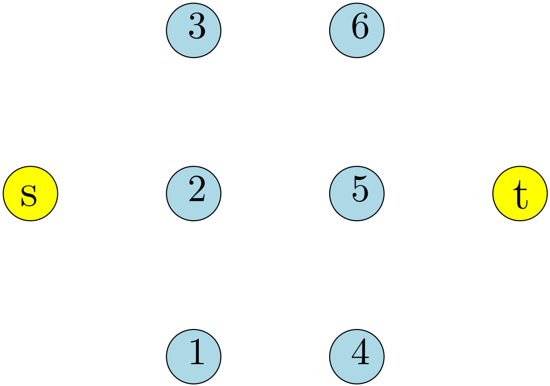
Example



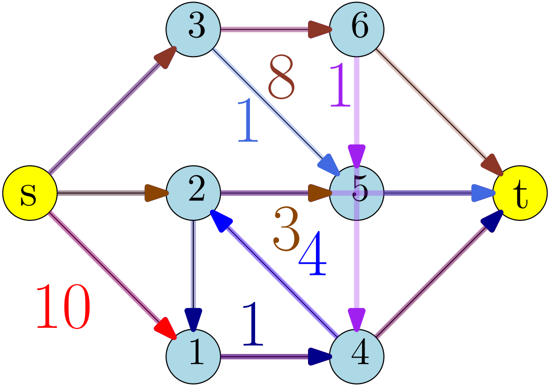
Example



Example

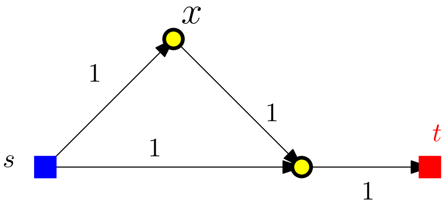


Example



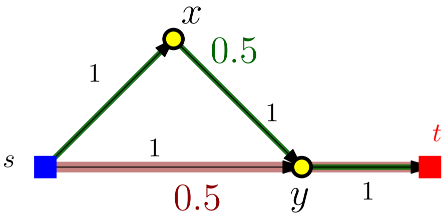
Path flow decomposition

Do not have to be efficient...



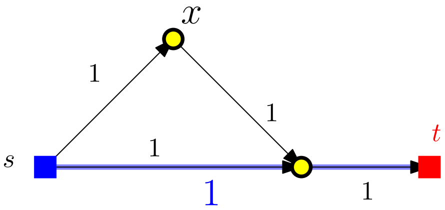
Path flow decomposition

Do not have to be efficient...



Path flow decomposition

Do not have to be efficient...



Edge vs Path based Definitions of Flow

1. Edge based flows:
   1. **compact** representation, only m values to be specified, and
   2. need to check flow conservation explicitly at each internal node.
2. Path flows:
   1. in some applications, paths more natural,
   2. not compact,
   3. no need to check flow conservation constraints.
3. Equivalence shows that we can go back and forth easily.

Edge vs Path based Definitions of Flow

1. Edge based flows:
   1. **compact** representation, only m values to be specified, and
   2. need to check flow conservation explicitly at each internal node.
2. Path flows:
   1. in some applications, paths more natural,
   2. not compact,
   3. no need to check flow conservation constraints.
3. Equivalence shows that we can go back and forth easily.

Edge vs Path based Definitions of Flow

1. Edge based flows:
    1. **compact** representation, only m values to be specified, and
    2. need to check flow conservation explicitly at each internal node.
2. Path flows:
    1. in some applications, paths more natural,
    2. not compact,
    3. no need to check flow conservation constraints.
3. Equivalence shows that we can go back and forth easily.

Edge vs Path based Definitions of Flow

1. Edge based flows:
   1. **compact** representation, only m values to be specified, and
   2. need to check flow conservation explicitly at each internal node.
2. Path flows:
   1. in some applications, paths more natural,
   2. not compact,
   3. no need to check flow conservation constraints.
3. Equivalence shows that we can go back and forth easily.

Edge vs Path based Definitions of Flow

1. Edge based flows:
    1. **compact** representation, only m values to be specified, and
    2. need to check flow conservation explicitly at each internal node.
2. Path flows:
    1. in some applications, paths more natural,
    2. not compact,
    3. no need to check flow conservation constraints.
3. Equivalence shows that we can go back and forth easily.

Edge vs Path based Definitions of Flow

1. Edge based flows:
   1. **compact** representation, only m values to be specified, and
   2. need to check flow conservation explicitly at each internal node.
2. Path flows:
   1. in some applications, paths more natural,
   2. not compact,
   3. no need to check flow conservation constraints.
3. Equivalence shows that we can go back and forth easily.

Edge vs Path based Definitions of Flow

1. Edge based flows:
    1. **compact** representation, only m values to be specified, and
    2. need to check flow conservation explicitly at each internal node.
2. Path flows:
    1. in some applications, paths more natural,
    2. not compact,
    3. no need to check flow conservation constraints.
3. Equivalence shows that we can go back and forth easily.

Edge vs Path based Definitions of Flow

1. Edge based flows:
   1. **compact** representation, only m values to be specified, and
   2. need to check flow conservation explicitly at each internal node.
2. Path flows:
   1. in some applications, paths more natural,
   2. not compact,
   3. no need to check flow conservation constraints.
3. Equivalence shows that we can go back and forth easily.

Edge vs Path based Definitions of Flow

1. Edge based flows:
   1. **compact** representation, only m values to be specified, and
   2. need to check flow conservation explicitly at each internal node.
2. Path flows:
   1. in some applications, paths more natural,
   2. not compact,
   3. no need to check flow conservation constraints.
3. Equivalence shows that we can go back and forth easily.

The Maximum-Flow Problem

1. The **network flow problem**:

Problem

Input: A network G with capacity c and source s and sink t.

Goal: Find flow of **maximum** value.

2. Question: Given a flow network, what is an *upper bound* on the maximum flow between source and sink?

The Maximum-Flow Problem

1. The **network flow problem**:

Problem

Input A network G with capacity c and source s and sink t.

Goal Find flow of **maximum** value.

2. Question: Given a flow network, what is an *upper bound* on the maximum flow between source and sink?

The Maximum-Flow Problem

1. The **network flow problem**:

Problem

Input A network G with capacity c and source s and sink t.

Goal Find flow of **maximum** value.

2. Question: Given a flow network, what is an *upper bound* on the maximum flow between source and sink?

The Maximum-Flow Problem

1. The **network flow problem**:

Problem

Input: A network G with capacity c and source s and sink t.

Goal: Find flow of **maximum** value.

2. Question: Given a flow network, what is an *upper bound* on the maximum flow between source and sink?

The Maximum-Flow Problem

1. The **network flow problem**:

Problem

Input A network G with capacity c and source s and sink t.

Goal Find flow of **maximum** value.

2. Question: Given a flow network, what is an *upper bound* on the maximum flow between source and sink?

Cuts

Definition (s-t cut)

Given a flow network an **s-t cut** is a set of edges $E' \subset E$ such that removing E' *disconnects* s from t: in other words there is no directed $s \rightarrow t$ path in $E - E'$.
The **capacity** of a cut E' is $c(E') = \sum_{e \in E'} c(e)$.

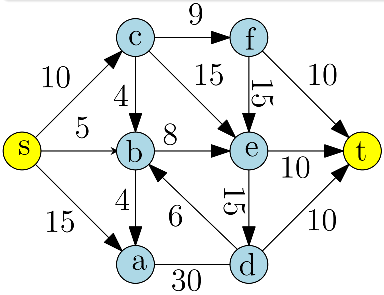
Caution:

1. Cut may leave $t \rightarrow s$ paths!
2. There might be many s-t cuts.

Cuts

Definition (s-t cut)

Given a flow network an **s-t cut** is a set of edges $E' \subset E$ such that removing E' *disconnects* s from t: in other words there is no directed $s \to t$ path in $E - E'$.
The **capacity** of a cut E' is $c(E') = \sum_{e \in E'} c(e)$.



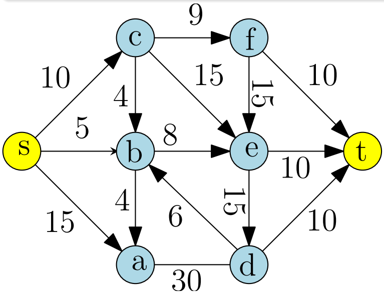
Caution:

1. Cut may leave $t \to s$ paths!
2. There might be many s-t cuts.

Cuts

Definition (s-t cut)

Given a flow network an **s-t cut** is a set of edges $E' \subset E$ such that removing E' *disconnects* s from t: in other words there is no directed $s \rightarrow t$ path in $E - E'$.
The **capacity** of a cut E' is $c(E') = \sum_{e \in E'} c(e)$.

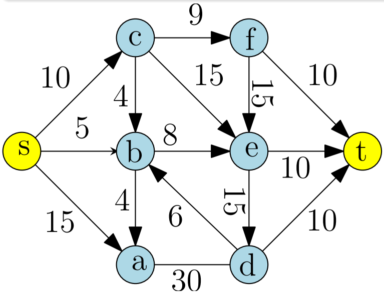


Caution:

1. Cut may leave $t \rightarrow s$ paths!
2. There might be many s-t cuts.

Definition (s-t cut)

Given a flow network an **s-t cut** is a set of edges $E' \subset E$ such that removing E' *disconnects* s from t: in other words there is no directed $s \to t$ path in $E - E'$.
The **capacity** of a cut E' is $c(E') = \sum_{e \in E'} c(e)$.



Caution:

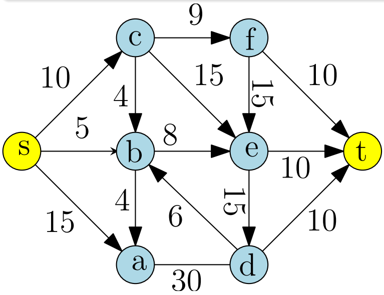
1. Cut may leave $t \to s$ paths!
2. There might be many s-t cuts.

Cuts

Definition (s-t cut)

Given a flow network an **s-t cut** is a set of edges $E' \subset E$ such that removing E' *disconnects* s from t: in other words there is no directed $s \to t$ path in $E - E'$.

The **capacity** of a cut E' is $c(E') = \sum_{e \in E'} c(e)$.

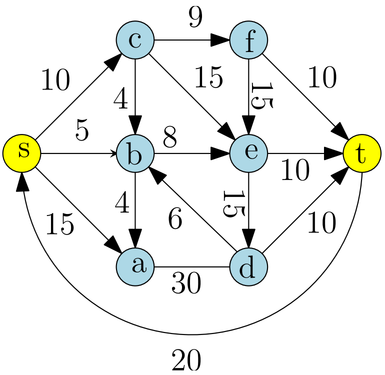


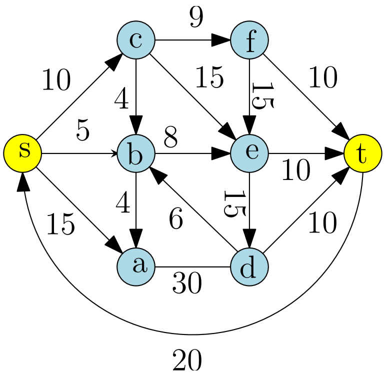
Caution:

1. Cut may leave $t \to s$ paths!
2. There might be many s-t cuts.

s — t cuts

A death by a thousand cuts

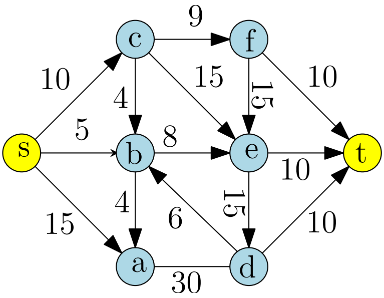




Minimal Cut

Definition (Minimal s-t cut.)

Given a s-t flow network $G = (V, E)$, $E' \subseteq E$ is a **minimal cut** if for all $e \in E'$, if $E' \setminus \{e\}$ is not a cut.

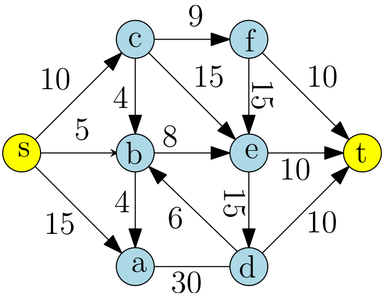


Observation: given a cut E', can check efficiently whether E' is a minimal cut or not. How?

Minimal Cut

Definition (Minimal s-t cut.)

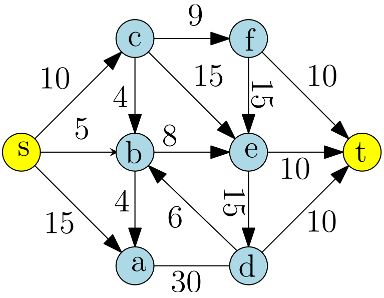
Given a s-t flow network $G = (V, E)$, $E' \subseteq E$ is a **minimal cut** if for all $e \in E'$, if $E' \setminus \{e\}$ is not a cut.



Observation: given a cut E', can check efficiently whether E' is a minimal cut or not. How?

Cuts as Vertex Partitions

1. Let $A \subset V$ such that
   1. $s \in A$, $t \notin A$, and
   2. $B = V \setminus A$ (hence $t \in B$).
2. The **cut** (A, B) is the set of edges $(A, B) = \{(u, v) \in E \mid u \in A, v \in B\}$. Cut (A, B) is set of edges leaving A.



Claim

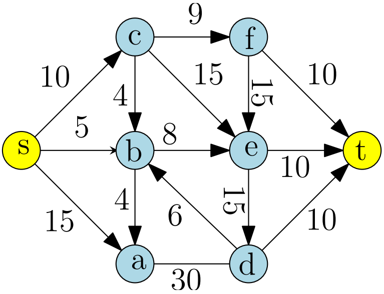
(A, B) is an s-t cut.

Proof.

Let P be any $s \to t$ path in G. Since t is not in A, P has to leave A via some edge (u, v) in (A, B). □

Cuts as Vertex Partitions

1. Let $A \subset V$ such that
   1. $s \in A$, $t \notin A$, and
   2. $B = V \setminus A$ (hence $t \in B$).
2. The **cut** (A, B) is the set of edges $(A, B) = \{(u, v) \in E \mid u \in A, v \in B\}$. Cut (A, B) is set of edges leaving A.



Claim

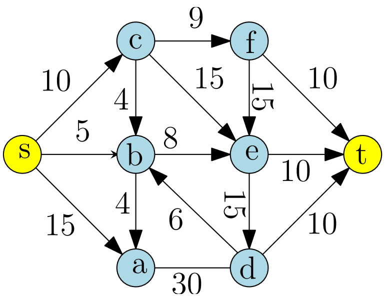
(A, B) is an s-t cut.

Proof.

Let P be any $s \to t$ path in G. Since t is not in A, P has to leave A via some edge (u, v) in (A, B). □

Cuts as Vertex Partitions

1. Let $A \subset V$ such that
   1. $s \in A$, $t \notin A$, and
   2. $B = V \setminus A$ (hence $t \in B$).
2. The **cut** (A, B) is the set of edges $(A, B) = \{(u, v) \in E \mid u \in A, v \in B\}$. Cut (A, B) is set of edges leaving A.



Claim

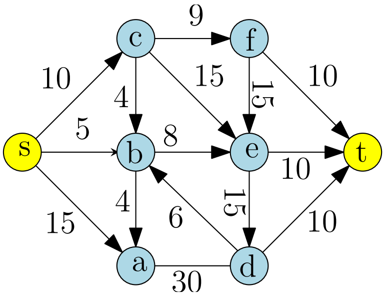
(A, B) is an s-t cut.

Proof.

Let P be any $s \to t$ path in G. Since t is not in A, P has to leave A via some edge (u, v) in (A, B). □

Cuts as Vertex Partitions

1. Let $A \subset V$ such that
   1. $s \in A$, $t \notin A$, and
   2. $B = V \setminus A$ (hence $t \in B$).
2. The **cut** (A, B) is the set of edges $(A, B) = \{(u, v) \in E \mid u \in A, v \in B\}$.
   Cut (A, B) is set of edges leaving A.



Claim

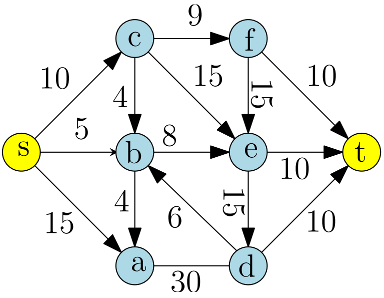
(A, B) is an s-t cut.

Proof.

Let P be any $s \to t$ path in G. Since t is not in A, P has to leave A via some edge (u, v) in (A, B). □

Cuts as Vertex Partitions

1. Let $A \subset V$ such that
   1. $s \in A$, $t \notin A$, and
   2. $B = V \setminus A$ (hence $t \in B$).
2. The **cut** (A, B) is the set of edges $(A, B) = \{(u, v) \in E \mid u \in A, v \in B\}$. Cut (A, B) is set of edges leaving A.



Claim

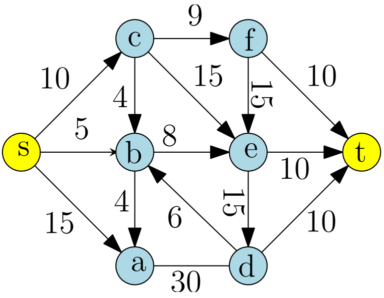
(A, B) is an s-t cut.

Proof.

Let P be any $s \to t$ path in G. Since t is not in A, P has to leave A via some edge (u, v) in (A, B). □

Cuts as Vertex Partitions

1. Let $A \subset V$ such that
   1. $s \in A$, $t \notin A$, and
   2. $B = V \setminus A$ (hence $t \in B$).
2. The **cut** (A, B) is the set of edges $(A, B) = \{(u, v) \in E \mid u \in A, v \in B\}$. Cut (A, B) is set of edges leaving A.



Claim

(A, B) is an s-t cut.

Proof.

Let P be any $s \to t$ path in G. Since t is not in A, P has to leave A via some edge (u, v) in (A, B). □

Cuts as Vertex Partitions

Lemma

Suppose E' *is an* s-t *cut. Then there is a cut* (A, B) *such that* $(A, B) \subseteq E'$.

Proof.

E' is an s-t cut implies no path from s to t in $(V, E - E')$.

1. Let A be set of all nodes reachable by s in $(V, E - E')$.
2. Since E' is a cut, $t \not\in A$.
3. $(A, B) \subseteq E'$. Why?

□

Corollary

Every minimal s-t *cut* E' *is a cut of the form* (A, B).

Cuts as Vertex Partitions

Lemma

Suppose E' is an s-t cut. Then there is a cut (A, B) such that $(A, B) \subseteq E'$.

Proof.

E' is an s-t cut implies no path from s to t in $(V, E - E')$.

1. Let A be set of all nodes reachable by s in $(V, E - E')$.
2. Since E' is a cut, $t \not\in A$.
3. $(A, B) \subseteq E'$. Why?

□

Corollary

Every minimal s-t cut E' is a cut of the form (A, B).

Cuts as Vertex Partitions

Lemma

Suppose E' is an s-t cut. Then there is a cut (A, B) such that $(A, B) \subseteq E'$.

Proof.

E' is an s-t cut implies no path from s to t in $(V, E - E')$.

1. Let A be set of all nodes reachable by s in $(V, E - E')$.
2. Since E' is a cut, $t \not\in A$.
3. $(A, B) \subseteq E'$. Why?If some edge $(u, v) \in (A, B)$ is not in E' then v will be reachable by s and should be in A, hence a contradiction. □

Corollary

Every minimal s-t cut E' is a cut of the form (A, B).

Cuts as Vertex Partitions

Lemma

Suppose E' is an s-t cut. Then there is a cut (A, B) such that $(A, B) \subseteq E'$.

Proof.

E' is an s-t cut implies no path from s to t in $(V, E - E')$.

1. Let A be set of all nodes reachable by s in $(V, E - E')$.
2. Since E' is a cut, $t \not\in A$.
3. $(A, B) \subseteq E'$. Why?If some edge $(u, v) \in (A, B)$ is not in E' then v will be reachable by s and should be in A, hence a contradiction. □

Corollary

Every minimal s-t cut E' is a cut of the form (A, B).

Cuts as Vertex Partitions

Lemma

Suppose E' is an s-t cut. Then there is a cut (A, B) such that $(A, B) \subseteq E'$.

Proof.

E' is an s-t cut implies no path from s to t in $(V, E - E')$.

1. Let A be set of all nodes reachable by s in $(V, E - E')$.
2. Since E' is a cut, $t \not\in A$.
3. $(A, B) \subseteq E'$. Why? If some edge $(u, v) \in (A, B)$ is not in E' then v will be reachable by s and should be in A, hence a contradiction. □

Corollary

Every minimal s-t cut E' is a cut of the form (A, B).

Cuts as Vertex Partitions

Lemma

Suppose E' is an s-t cut. Then there is a cut (A, B) such that $(A, B) \subseteq E'$.

Proof.

E' is an s-t cut implies no path from s to t in $(V, E - E')$.

1. Let A be set of all nodes reachable by s in $(V, E - E')$.
2. Since E' is a cut, $t \not\in A$.
3. $(A, B) \subseteq E'$. Why?If some edge $(u, v) \in (A, B)$ is not in E' then v will be reachable by s and should be in A, hence a contradiction. □

Corollary

Every minimal s-t cut E' is a cut of the form (A, B).

Cuts as Vertex Partitions

Lemma

Suppose E' *is an* s-t *cut. Then there is a cut* (A, B) *such that* $(A, B) \subseteq E'$.

Proof.

E' is an s-t cut implies no path from s to t in $(V, E - E')$.

1. Let A be set of all nodes reachable by s in $(V, E - E')$.
2. Since E' is a cut, $t \notin A$.
3. $(A, B) \subseteq E'$. Why?If some edge $(u, v) \in (A, B)$ is not in E' then v will be reachable by s and should be in A, hence a contradiction. □

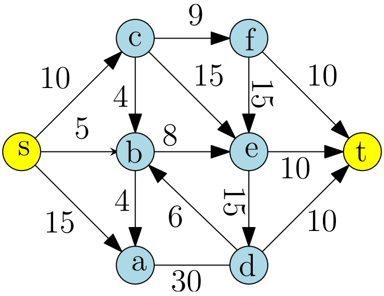
Corollary

Every minimal s-t *cut* E' *is a cut of the form* (A, B).

Minimum Cut

Definition

Given a flow network an s-t **minimum** cut is a cut E' of smallest capacity among all s-t cuts.

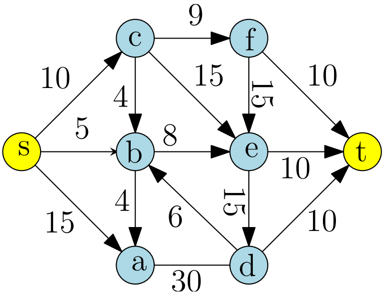


Observation: exponential number of s-t cuts and no "easy" algorithm to find a minimum cut.

Minimum Cut

Definition

Given a flow network an ***s*-*t*** **minimum** cut is a cut E' of smallest capacity among all ***s*-*t*** cuts.



Observation: exponential number of s-t cuts and no "easy" algorithm to find a minimum cut.

The Minimum-Cut Problem

Problem

Input A flow network G

Goal Find the capacity of a *minimum* s-t cut

The Minimum-Cut Problem

Problem

Input A flow network G

Goal Find the capacity of a *minimum* s-t cut

The Minimum-Cut Problem

Problem

Input: A flow network G

Goal: Find the capacity of a *minimum* s-t cut

Flows and Cuts

Lemma

For any s-t *cut* E', **maximum** s-t *flow* $\leq$ *capacity of* E'.

Proof.

1. Formal proof easier with path based definition of flow.
2. Suppose $f : \mathcal{P} \to \mathbb{R}^{\geq 0}$ is a max-flow.
3. Every path $p \in \mathcal{P}$ contains an edge $e \in E'$. Why?
4. Assign each path $p \in \mathcal{P}$ to exactly one edge $e \in E'$.
5. Let $\mathcal{P}_e$ be paths assigned to $e \in E'$. Then

$$v(f) = \sum_{p \in \mathcal{P}} f(p) = \sum_{e \in E'} \sum_{p \in \mathcal{P}_e} f(p) \leq \sum_{e \in E'} c(e).$$

Flows and Cuts

Lemma

For any s-t cut E', **maximum** *s-t flow $\leq$ capacity of E'.*

Proof.

1. Formal proof easier with path based definition of flow.
2. Suppose $f : \mathcal{P} \to \mathbb{R}^{\geq 0}$ is a max-flow.
3. Every path $p \in \mathcal{P}$ contains an edge $e \in E'$. Why?
4. Assign each path $p \in \mathcal{P}$ to exactly one edge $e \in E'$.
5. Let $\mathcal{P}_e$ be paths assigned to $e \in E'$. Then

$$v(f) = \sum_{p \in \mathcal{P}} f(p) = \sum_{e \in E'} \sum_{p \in \mathcal{P}_e} f(p) \leq \sum_{e \in E'} c(e).$$

Flows and Cuts

Lemma

*For any s-t cut E', **maximum** s-t flow $\leq$ capacity of E'.*

Proof.

1. Formal proof easier with path based definition of flow.
2. Suppose $f : \mathcal{P} \rightarrow \mathbb{R}^{\geq 0}$ is a max-flow.
3. Every path $p \in \mathcal{P}$ contains an edge $e \in E'$. Why?
4. Assign each path $p \in \mathcal{P}$ to exactly one edge $e \in E'$.
5. Let $\mathcal{P}_e$ be paths assigned to $e \in E'$. Then

$$v(f) = \sum_{p \in \mathcal{P}} f(p) = \sum_{e \in E'} \sum_{p \in \mathcal{P}_e} f(p) \leq \sum_{e \in E'} c(e).$$

Flows and Cuts

Lemma

*For any s-t cut E', **maximum** s-t flow $\leq$ capacity of E'.*

Proof.

1. Formal proof easier with path based definition of flow.
2. Suppose $f : \mathcal{P} \rightarrow \mathbb{R}^{\geq 0}$ is a max-flow.
3. Every path $p \in \mathcal{P}$ contains an edge $e \in E'$. Why?
4. Assign each path $p \in \mathcal{P}$ to exactly one edge $e \in E'$.
5. Let $\mathcal{P}_e$ be paths assigned to $e \in E'$. Then

$$v(f) = \sum_{p \in \mathcal{P}} f(p) = \sum_{e \in E'} \sum_{p \in \mathcal{P}_e} f(p) \leq \sum_{e \in E'} c(e).$$

Flows and Cuts

Lemma

For any s-t cut E', **maximum** *s-t flow $\leq$ capacity of E'.*

Proof.

1. Formal proof easier with path based definition of flow.
2. Suppose $f : \mathcal{P} \to \mathbb{R}^{\geq 0}$ is a max-flow.
3. Every path $p \in \mathcal{P}$ contains an edge $e \in E'$. Why?
4. Assign each path $p \in \mathcal{P}$ to exactly one edge $e \in E'$.
5. Let $\mathcal{P}_e$ be paths assigned to $e \in E'$. Then

$$v(f) = \sum_{p \in \mathcal{P}} f(p) = \sum_{e \in E'} \sum_{p \in \mathcal{P}_e} f(p) \leq \sum_{e \in E'} c(e).$$

Flows and Cuts

Lemma

For any s-t cut E', **maximum** *s-t flow $\leq$ capacity of E'.*

Proof.

1. Formal proof easier with path based definition of flow.
2. Suppose $f : \mathcal{P} \to \mathbb{R}^{\geq 0}$ is a max-flow.
3. Every path $p \in \mathcal{P}$ contains an edge $e \in E'$. Why?
4. Assign each path $p \in \mathcal{P}$ to exactly one edge $e \in E'$.
5. Let $\mathcal{P}_e$ be paths assigned to $e \in E'$. Then

$$v(f) = \sum_{p \in \mathcal{P}} f(p) = \sum_{e \in E'} \sum_{p \in \mathcal{P}_e} f(p) \leq \sum_{e \in E'} c(e).$$

Flows and Cuts

Lemma

For any s-t *cut* E', **maximum** s-t *flow* $\leq$ *capacity of* E'.

Proof.

1. Formal proof easier with path based definition of flow.
2. Suppose $f : \mathcal{P} \to \mathbb{R}^{\geq 0}$ is a max-flow.
3. Every path $p \in \mathcal{P}$ contains an edge $e \in E'$. Why?
4. Assign each path $p \in \mathcal{P}$ to exactly one edge $e \in E'$.
5. Let $\mathcal{P}_e$ be paths assigned to $e \in E'$. Then

$$v(f) = \sum_{p \in \mathcal{P}} f(p) = \sum_{e \in E'} \sum_{p \in \mathcal{P}_e} f(p) \leq \sum_{e \in E'} c(e).$$

Flows and Cuts

Lemma

For any s-t *cut* E', **maximum** s-t *flow* $\leq$ *capacity of* E'.

Corollary

Maximum s-t *flow* $\leq$ *minimum* s-t *cut.*

Max-Flow Min-Cut Theorem

Theorem

In any flow network:

$$\Big(\textit{value of maximum } \boldsymbol{s}\text{-}\boldsymbol{t} \textit{ flow}\Big) = \Big(\textit{capacity of minimum } \boldsymbol{s}\text{-}\boldsymbol{t} \textit{ cut}\Big).$$

1. Can compute minimum-cut from maximum flow and vice-versa!
2. Proof coming shortly.
3. Many applications:
    1. optimization
    2. graph theory
    3. combinatorics

Max-Flow Min-Cut Theorem

Theorem

In any flow network:

$$\left(\textit{value of maximum } \mathbf{s}\text{-}\mathbf{t} \textit{ flow}\right) = \left(\textit{capacity of minimum } \mathbf{s}\text{-}\mathbf{t} \textit{ cut}\right).$$

1. Can compute minimum-cut from maximum flow and vice-versa!
2. Proof coming shortly.
3. Many applications:
    1. optimization
    2. graph theory
    3. combinatorics

Max-Flow Min-Cut Theorem

Theorem

In any flow network:

$$\left(\textit{value of maximum } \mathbf{s}\text{-}\mathbf{t} \textit{ flow}\right) = \left(\textit{capacity of minimum } \mathbf{s}\text{-}\mathbf{t} \textit{ cut}\right).$$

1. Can compute minimum-cut from maximum flow and vice-versa!
2. Proof coming shortly.
3. Many applications:
    1. optimization
    2. graph theory
    3. combinatorics

Max-Flow Min-Cut Theorem

Theorem

In any flow network:

$$\left(\textit{value of maximum } \mathbf{s}\text{-}\mathbf{t} \textit{ flow}\right) = \left(\textit{capacity of minimum } \mathbf{s}\text{-}\mathbf{t} \textit{ cut}\right).$$

1. Can compute minimum-cut from maximum flow and vice-versa!
2. Proof coming shortly.
3. Many applications:
    1. optimization
    2. graph theory
    3. combinatorics

Max-Flow Min-Cut Theorem

Theorem

In any flow network:

$$\Big(\textit{value of maximum } \boldsymbol{s}\text{-}\boldsymbol{t} \textit{ flow}\Big) = \Big(\textit{capacity of minimum } \boldsymbol{s}\text{-}\boldsymbol{t} \textit{ cut}\Big).$$

1. Can compute minimum-cut from maximum flow and vice-versa!
2. Proof coming shortly.
3. Many applications:
    1. optimization
    2. graph theory
    3. combinatorics

Max-Flow Min-Cut Theorem

Theorem

In any flow network:

$$\left(\textit{value of maximum } \mathbf{s}\text{-}\mathbf{t} \textit{ flow}\right) = \left(\textit{capacity of minimum } \mathbf{s}\text{-}\mathbf{t} \textit{ cut}\right).$$

1. Can compute minimum-cut from maximum flow and vice-versa!
2. Proof coming shortly.
3. Many applications:
   1. optimization
   2. graph theory
   3. combinatorics

Max-Flow Min-Cut Theorem

Theorem

In any flow network:

$$\left(\textit{value of maximum } \boldsymbol{s}\text{-}\boldsymbol{t} \textit{ flow}\right) = \left(\textit{capacity of minimum } \boldsymbol{s}\text{-}\boldsymbol{t} \textit{ cut}\right).$$

1. Can compute minimum-cut from maximum flow and vice-versa!
2. Proof coming shortly.
3. Many applications:
    1. optimization
    2. graph theory
    3. combinatorics

The Maximum-Flow Problem

Problem

Input A network G with capacity c and source s and sink t.

Goal Find flow of **maximum** value from s to t.

Exercise: Given G, s, t as above, show that one can remove all edges into s and all edges out of t without affecting the flow value between s and t.

The Maximum-Flow Problem

Problem

Input A network G with capacity c and source s and sink t.

Goal Find flow of **maximum** value from s to t.

Exercise: Given G, s, t as above, show that one can remove all edges into s and all edges out of t without affecting the flow value between s and t.

The Maximum-Flow Problem

Problem

Input: A network G with capacity c and source s and sink t.

Goal: Find flow of **maximum** value from s to t.

Exercise: Given G, s, t as above, show that one can remove all edges into s and all edges out of t without affecting the flow value between s and t.

The Maximum-Flow Problem

Problem

Input A network G with capacity c and source s and sink t.

Goal Find flow of **maximum** value from s to t.

Exercise: Given G, s, t as above, show that one can remove all edges into s and all edges out of t without affecting the flow value between s and t.

Notes

Notes

Notes

Notes

Dinic, E. A. (1970). Algorithm for solution of a problem of maximum flow in a network with power estimation. *Soviet Math. Doklady*, 11:1277–1280.

Edmonds, J. and Karp, R. M. (1972). Theoretical improvements in algorithmic efficiency for network flow problems. *J. Assoc. Comput. Mach.*, 19(2):248–264.