

# Applications of Network Flows

Lecture 18

April 3, 2014

# Is the flow always integral?

Let  $G$  be an integral instance of network flow (i.e., all numbers are integers). Consider the following statements:

- (I) The value of the maximum flow is an integer number.
- (II) If  $f$  is a maximum flow, then  $f(e)$  is an integer, for any edge  $e \in E(G)$ .
- (III) There always exists a max flow  $g$ , such that  $g$  is a maximum flow, and  $g(e)$  is an integer, for any edge  $e \in E(G)$ .

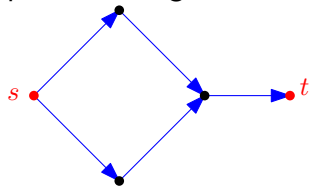
We have the following:

- (A) All the above statements are false.
- (B) All the above statements are true.
- (C) (I) is true, (II) and (III) are false.
- (D) (I) and (II) are true, and (III) is false.
- (E) (I) and (III) are true, and (II) is false.

# Why max-flow does not have to be integral...

...but the one we compute always is!

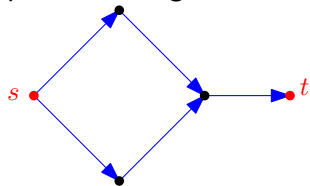
Consider the graph with all capacities being one.



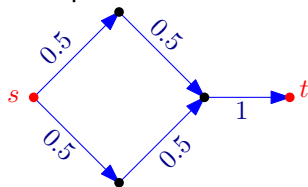
# Why max-flow does not have to be integral...

...but the one we compute always is!

Consider the graph with all capacities being one.



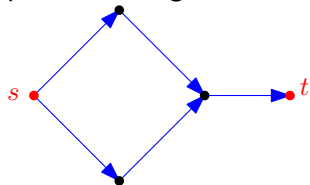
One possible max flow:



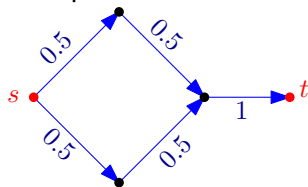
# Why max-flow does not have to be integral...

...but the one we compute always is!

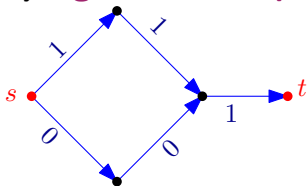
Consider the graph with all capacities being one.



One possible max flow:



Max flow as computed by **algEdmondsKarp** or **algFordFulkerson**:



# Network Flow: Facts to Remember

Flow network: directed graph  $G$ , capacities  $c$ , source  $s$ , sink  $t$ .

- 1 Maximum  $s$ - $t$  flow can be computed:
  - 1 Using Ford-Fulkerson algorithm in  $O(mC)$  time when capacities are integral and  $C$  is an upper bound on the flow.
  - 2 Using variant of algorithm, in  $O(m^2 \log C)$  time, when capacities are integral. (Polynomial time.)
  - 3 Using Edmonds-Karp algorithm, in  $O(m^2n)$  time, when capacities are rational (strongly polynomial time algorithm).

# Network Flow

Even more facts to remember

- 1 If capacities are integral then there is a maximum flow that is integral and above algorithms give an integral max flow. This is known as **integrality of flow**.
- 2 Given a flow of value  $v$ , can decompose into  $O(m + n)$  flow paths of same total value  $v$ . Integral flow implies integral flow on paths.
- 3 Maximum flow is equal to the minimum cut and minimum cut can be found in  $O(m + n)$  time given any maximum flow.

# Paths, Cycles and Acyclicity of Flows

## Definition

Given a flow network  $\mathbf{G} = (\mathbf{V}, \mathbf{E})$  and a flow  $\mathbf{f} : \mathbf{E} \rightarrow \mathbb{R}^{\geq 0}$  on the edges, the **support** of  $\mathbf{f}$  is the set of edges  $\mathbf{E}' \subseteq \mathbf{E}$  with non-zero flow on them. That is,  $\mathbf{E}' = \{\mathbf{e} \in \mathbf{E} \mid \mathbf{f}(\mathbf{e}) > 0\}$ .

**Question:** Given a flow  $\mathbf{f}$ , can there be cycles in its support?

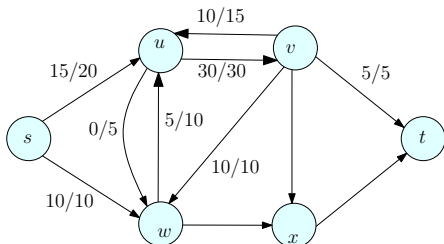


# Paths, Cycles and Acyclicity of Flows

## Definition

Given a flow network  $\mathbf{G} = (\mathbf{V}, \mathbf{E})$  and a flow  $\mathbf{f} : \mathbf{E} \rightarrow \mathbb{R}^{\geq 0}$  on the edges, the **support** of  $\mathbf{f}$  is the set of edges  $\mathbf{E}' \subseteq \mathbf{E}$  with non-zero flow on them. That is,  $\mathbf{E}' = \{\mathbf{e} \in \mathbf{E} \mid \mathbf{f}(\mathbf{e}) > 0\}$ .

**Question:** Given a flow  $\mathbf{f}$ , can there be cycles in its support?

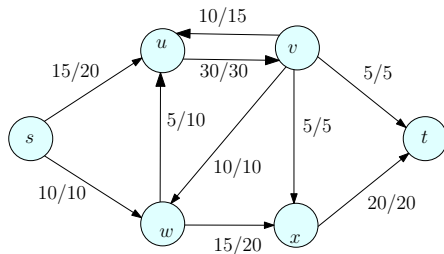


# Paths, Cycles and Acyclicity of Flows

## Definition

Given a flow network  $\mathbf{G} = (\mathbf{V}, \mathbf{E})$  and a flow  $\mathbf{f} : \mathbf{E} \rightarrow \mathbb{R}^{\geq 0}$  on the edges, the **support** of  $\mathbf{f}$  is the set of edges  $\mathbf{E}' \subseteq \mathbf{E}$  with non-zero flow on them. That is,  $\mathbf{E}' = \{\mathbf{e} \in \mathbf{E} \mid \mathbf{f}(\mathbf{e}) > 0\}$ .

**Question:** Given a flow  $\mathbf{f}$ , can there be cycles in its support?

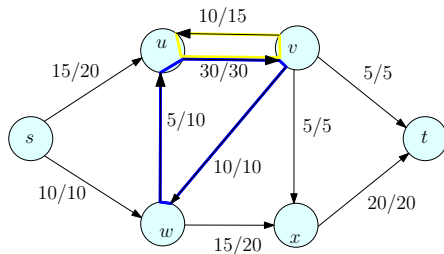


# Paths, Cycles and Acyclicity of Flows

## Definition

Given a flow network  $\mathbf{G} = (\mathbf{V}, \mathbf{E})$  and a flow  $\mathbf{f} : \mathbf{E} \rightarrow \mathbb{R}^{\geq 0}$  on the edges, the **support** of  $\mathbf{f}$  is the set of edges  $\mathbf{E}' \subseteq \mathbf{E}$  with non-zero flow on them. That is,  $\mathbf{E}' = \{\mathbf{e} \in \mathbf{E} \mid \mathbf{f}(\mathbf{e}) > 0\}$ .

**Question:** Given a flow  $\mathbf{f}$ , can there be cycles in its support?



# How fast can we detect a cycle in the flow

Given a flow network  $G$  with  $n$  vertices, and  $m$  edges, and a flow  $f$  on it, then detecting a cycle in the flow can be done in time

- (A)  $O(m + n)$ .
- (B)  $O(mC)$ .
- (C)  $O(mn)$ .
- (D)  $O(m^2n)$ .
- (E)  $O(mn^2)$ .

# Acyclicity of Flows

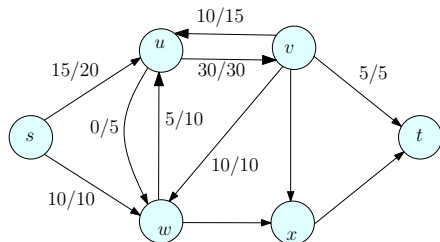
## Proposition

*In any flow network, if  $\mathbf{f}$  is a flow then there is another flow  $\mathbf{f}'$  such that the support of  $\mathbf{f}'$  is an acyclic graph and  $\mathbf{v}(\mathbf{f}') = \mathbf{v}(\mathbf{f})$ . Further if  $\mathbf{f}$  is an integral flow then so is  $\mathbf{f}'$ .*

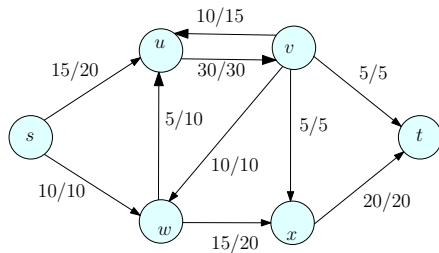
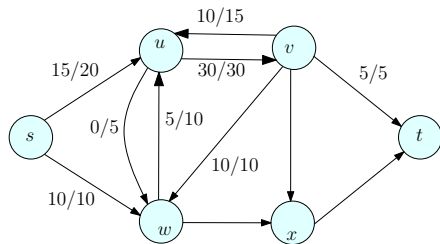
## Proof.

- 1  $\mathbf{E}' = \{\mathbf{e} \in \mathbf{E} \mid \mathbf{f}(\mathbf{e}) > \mathbf{0}\}$ , support of  $\mathbf{f}$ .
- 2 Suppose there is a directed cycle  $\mathbf{C}$  in  $\mathbf{E}'$
- 3 Let  $\mathbf{e}'$  be the edge in  $\mathbf{C}$  with least amount of flow
- 4 For each  $\mathbf{e} \in \mathbf{C}$ , reduce flow by  $\mathbf{f}(\mathbf{e}')$ . Remains a flow. Why?
- 5 Flow on  $\mathbf{e}'$  is reduced to  $\mathbf{0}$ .
- 6 Claim: Flow value from  $\mathbf{s}$  to  $\mathbf{t}$  does not change. Why?
- 7 Iterate until no cycles □

# Example

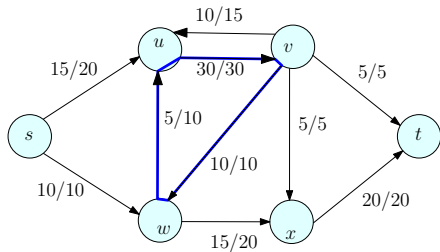
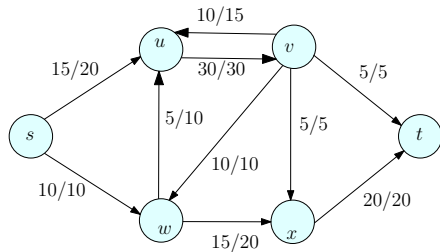


# Example



Throw away edge with no flow on it

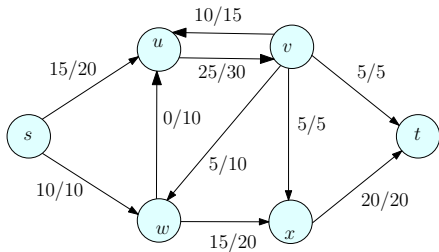
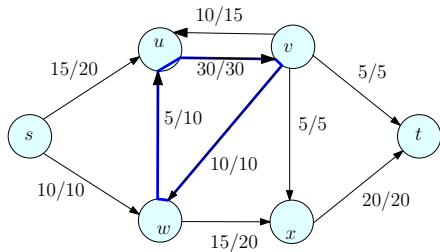
# Example



Find a cycle in the support/flow

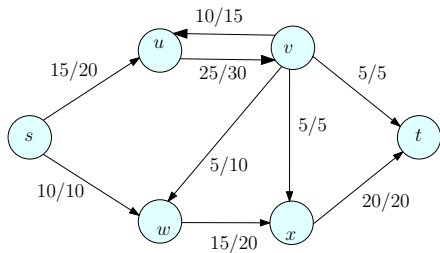
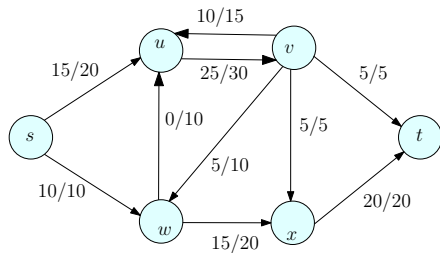


# Example



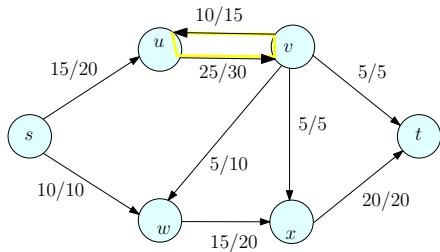
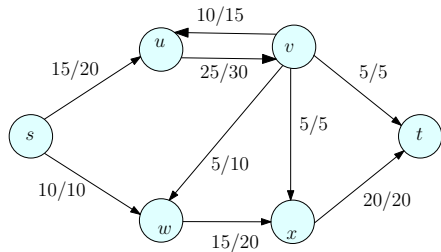
Reduce flow on cycle as much as possible

# Example



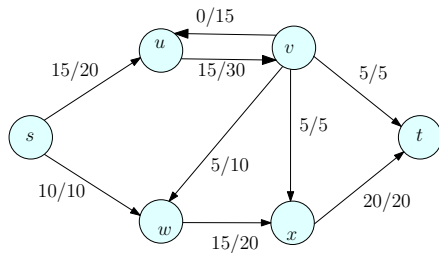
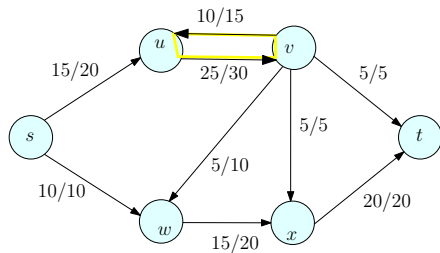
Throw away edge with no flow on it

# Example



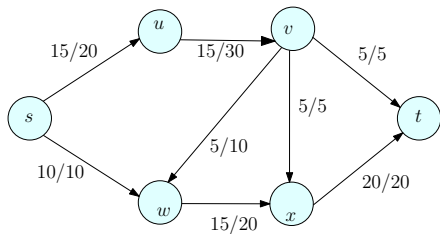
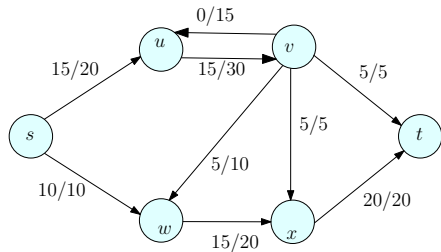
Find a cycle in the support/flow

# Example



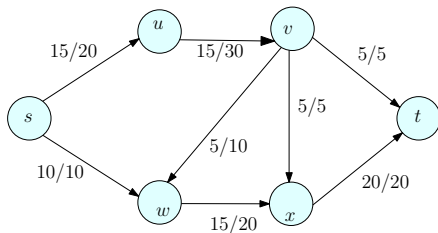
Reduce flow on cycle as much as possible

# Example

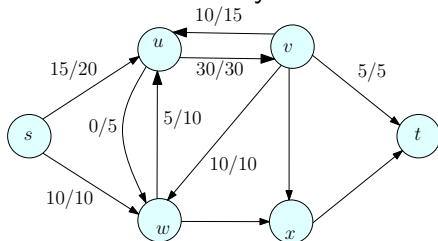


Throw away edge with no flow on it

# Example



Viola!!! An equivalent flow with no cycles in it. Original flow:



# Flow Decomposition

## Lemma

Given an edge based flow  $\mathbf{f} : \mathbf{E} \rightarrow \mathbb{R}^{\geq 0}$ , there exists a collection of paths  $\mathcal{P}$  and cycles  $\mathcal{C}$  and an assignment of flow to them  $\mathbf{f}' : \mathcal{P} \cup \mathcal{C} \rightarrow \mathbb{R}^{\geq 0}$  such that:

- 1  $|\mathcal{P} \cup \mathcal{C}| \leq m$
- 2 for each  $\mathbf{e} \in \mathbf{E}$ ,  $\sum_{\mathbf{P} \in \mathcal{P}: \mathbf{e} \in \mathbf{P}} \mathbf{f}'(\mathbf{P}) + \sum_{\mathbf{C} \in \mathcal{C}: \mathbf{e} \in \mathbf{C}} \mathbf{f}'(\mathbf{C}) = \mathbf{f}(\mathbf{e})$
- 3  $\mathbf{v}(\mathbf{f}) = \sum_{\mathbf{P} \in \mathcal{P}} \mathbf{f}'(\mathbf{P})$ .
- 4 if  $\mathbf{f}$  is integral then so are  $\mathbf{f}'(\mathbf{P})$  and  $\mathbf{f}'(\mathbf{C})$  for all  $\mathbf{P}$  and  $\mathbf{C}$

## Proof Idea.

- 1 Remove all cycles as in previous proposition.
- 2 Next, decompose into paths as in previous lecture.
- 3 Exercise: verify claims. □

# Flow Decomposition

## Lemma

Given an edge based flow  $\mathbf{f} : \mathbf{E} \rightarrow \mathbb{R}^{\geq 0}$ , there exists a collection of paths  $\mathcal{P}$  and cycles  $\mathcal{C}$  and an assignment of flow to them  $\mathbf{f}' : \mathcal{P} \cup \mathcal{C} \rightarrow \mathbb{R}^{\geq 0}$  such that:

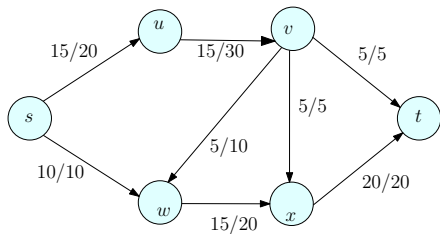
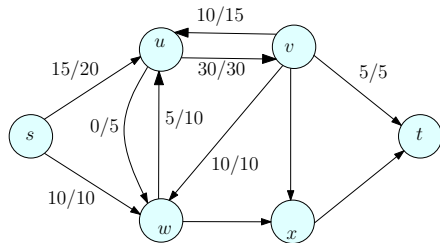
- 1  $|\mathcal{P} \cup \mathcal{C}| \leq m$
- 2 for each  $\mathbf{e} \in \mathbf{E}$ ,  $\sum_{\mathbf{P} \in \mathcal{P}: \mathbf{e} \in \mathbf{P}} \mathbf{f}'(\mathbf{P}) + \sum_{\mathbf{C} \in \mathcal{C}: \mathbf{e} \in \mathbf{C}} \mathbf{f}'(\mathbf{C}) = \mathbf{f}(\mathbf{e})$
- 3  $\mathbf{v}(\mathbf{f}) = \sum_{\mathbf{P} \in \mathcal{P}} \mathbf{f}'(\mathbf{P})$ .
- 4 if  $\mathbf{f}$  is integral then so are  $\mathbf{f}'(\mathbf{P})$  and  $\mathbf{f}'(\mathbf{C})$  for all  $\mathbf{P}$  and  $\mathbf{C}$

## Proof Idea.

- 1 Remove all cycles as in previous proposition.
- 2 Next, decompose into paths as in previous lecture.
- 3 Exercise: verify claims. □

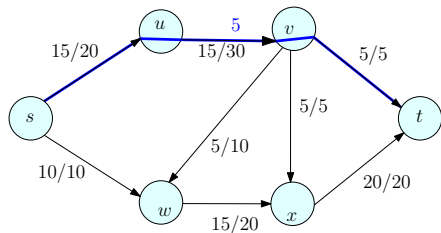
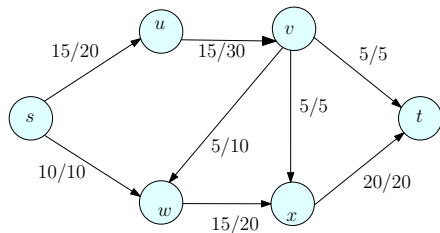


# Example



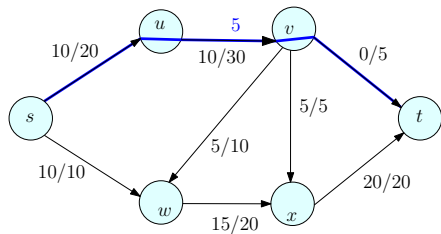
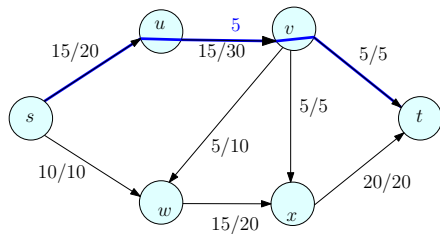
Find cycles as shown before

# Example



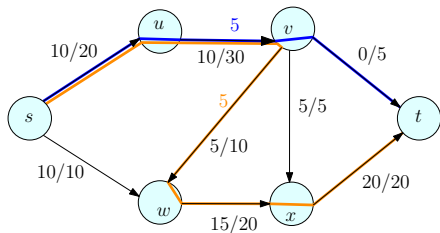
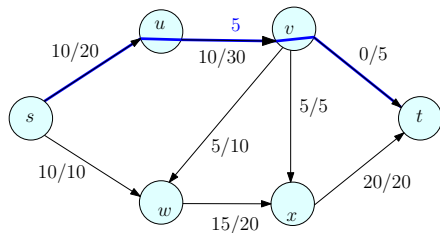
Find a source to sink path, and push max flow along it (5 unites)

# Example



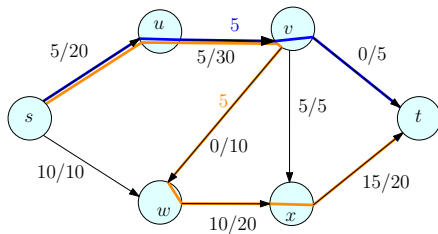
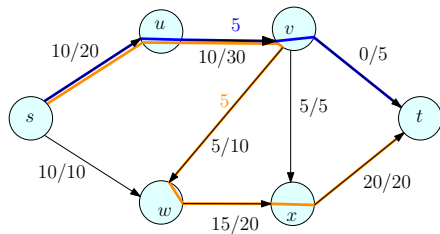
Compute remaining flow

# Example



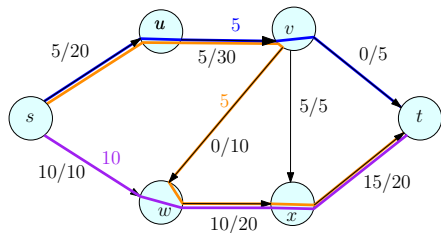
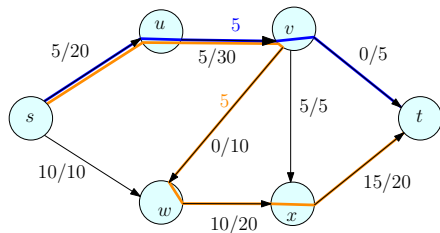
Find a source to sink path, and push max flow along it (5 units).  
Edges with **0** flow on them can not be used as they are no longer in the support of the flow.

# Example



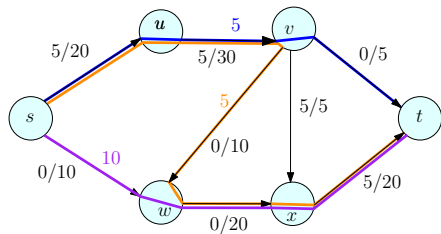
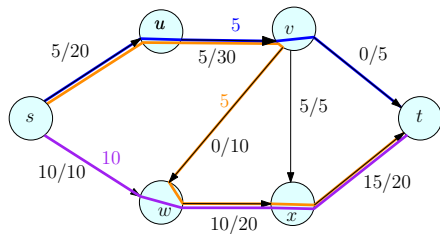
Compute remaining flow

# Example



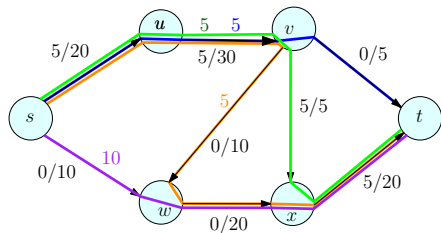
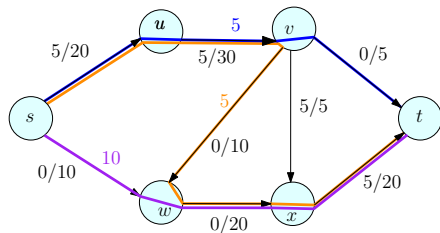
Find a source to sink path, and push max flow along it (10 unites).

# Example



Compute remaining flow

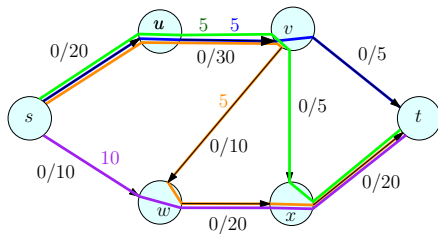
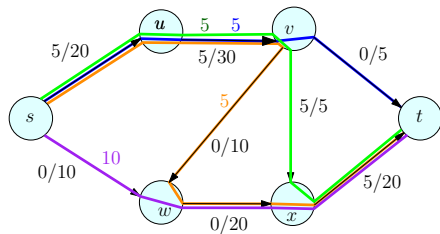
# Example



Find a source to sink path, and push max flow along it (5 unites).

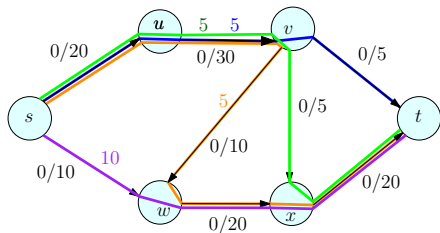
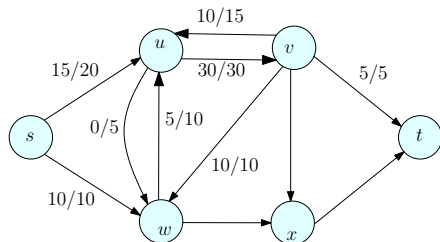


# Example



Compute remaining flow

# Example



No flow remains in the graph. We fully decomposed the flow into flow on paths. Together with the cycles, we get a decomposition of the original flow into  $m$  flows on paths and cycles.

# Flow Decomposition

## Lemma

Given an edge based flow  $\mathbf{f} : \mathbf{E} \rightarrow \mathbb{R}^{\geq 0}$ , there exists a collection of paths  $\mathcal{P}$  and cycles  $\mathcal{C}$  and an assignment of flow to them  $\mathbf{f}' : \mathcal{P} \cup \mathcal{C} \rightarrow \mathbb{R}^{\geq 0}$  such that:

- 1  $|\mathcal{P} \cup \mathcal{C}| \leq m$
- 2 for each  $\mathbf{e} \in \mathbf{E}$ ,  $\sum_{\mathbf{P} \in \mathcal{P}: \mathbf{e} \in \mathbf{P}} \mathbf{f}'(\mathbf{P}) + \sum_{\mathbf{C} \in \mathcal{C}: \mathbf{e} \in \mathbf{C}} \mathbf{f}'(\mathbf{C}) = \mathbf{f}(\mathbf{e})$
- 3  $\mathbf{v}(\mathbf{f}) = \sum_{\mathbf{P} \in \mathcal{P}} \mathbf{f}'(\mathbf{P})$ .
- 4 if  $\mathbf{f}$  is integral then so are  $\mathbf{f}'(\mathbf{P})$  and  $\mathbf{f}'(\mathbf{C})$  for all  $\mathbf{P}$  and  $\mathbf{C}$ .

Above flow decomposition can be computed in  $\mathbf{O}(m^2)$  time.

# Flow decomposition into paths and cycles

Consider an integral flow network  $G$ , and two maximum flows  $\mathbf{f}$  and  $\mathbf{g}$  in  $G$ . Assume both  $\mathbf{f}$  and  $\mathbf{g}$  are acyclic. Let  $\mathbf{P}_f$  and  $\mathbf{P}_g$  be the decomposition of the two flows into paths. Then:

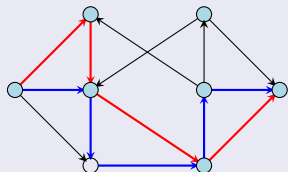
- (A)  $\mathbf{P}_f = \mathbf{P}_g$  (paths are the same).
- (B)  $|\mathbf{P}_f| = |\mathbf{P}_g|$  (i.e., number of paths is the same).
- (C)  $|\mathbf{P}_f| + |\mathbf{P}_g| = m$ .
- (D)  $|\mathbf{P}_f| * |\mathbf{P}_g| = nm$ .
- (E) None of the above.

# Part I

## Network Flow Applications I

# Edge-Disjoint Paths in Directed Graphs

## Definition



A set of paths is **edge disjoint** if no two paths share an edge.

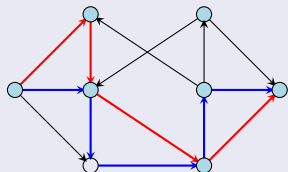
## Problem

Given a directed graph with two special vertices  $s$  and  $t$ , find the *maximum* number of edge disjoint paths from  $s$  to  $t$ .

**Applications:** Fault tolerance in routing — edges/nodes in networks can fail. Disjoint paths allow for planning backup routes in case of failures.

# Edge-Disjoint Paths in Directed Graphs

## Definition



A set of paths is **edge disjoint** if no two paths share an edge.

## Problem

Given a directed graph with two special vertices **s** and **t**, find the *maximum* number of edge disjoint paths from **s** to **t**.

**Applications:** Fault tolerance in routing — edges/nodes in networks can fail. Disjoint paths allow for planning backup routes in case of failures.

# Reduction to Max-Flow

## Problem

Given a directed graph  $G$  with two special vertices  $s$  and  $t$ , find the maximum number of edge disjoint paths from  $s$  to  $t$ .

## Reduction

Consider  $G$  as a flow network with edge capacities  $1$ , and compute max-flow.



# Correctness of Reduction

## Lemma

If  $G$  has  $k$  edge disjoint paths  $P_1, P_2, \dots, P_k$  then there is an  $s$ - $t$  flow of value  $k$  in  $G$ .

## Proof.

Set  $f(e) = 1$  if  $e$  belongs to one of the paths  $P_1, P_2, \dots, P_k$ ; otherwise set  $f(e) = 0$ . This defines a flow of value  $k$ .  $\square$

# Correctness of Reduction

## Lemma

If  $G$  has  $k$  edge disjoint paths  $P_1, P_2, \dots, P_k$  then there is an  $s$ - $t$  flow of value  $k$  in  $G$ .

## Proof.

Set  $f(e) = 1$  if  $e$  belongs to one of the paths  $P_1, P_2, \dots, P_k$ ; otherwise set  $f(e) = 0$ . This defines a flow of value  $k$ . □

# Correctness of Reduction

## Lemma

*If  $G$  has a flow of value  $k$  then there are  $k$  edge disjoint paths between  $s$  and  $t$ .*

## Proof.

- 1 Capacities are all **1** and hence there is integer flow of value  $k$ , that is  $f(e) = 0$  or  $f(e) = 1$  for each  $e$ .
- 2 Decompose flow into paths.
- 3 Flow on each path is either **1** or **0**.
- 4 Hence there are  $k$  paths  $P_1, P_2, \dots, P_k$  with flow of **1** each.
- 5 Paths are edge-disjoint since capacities are **1**. □

# Correctness of Reduction

## Lemma

*If  $G$  has a flow of value  $k$  then there are  $k$  edge disjoint paths between  $s$  and  $t$ .*

## Proof.

- 1 Capacities are all **1** and hence there is integer flow of value  $k$ , that is  $f(e) = 0$  or  $f(e) = 1$  for each  $e$ .
- 2 Decompose flow into paths.
- 3 Flow on each path is either **1** or **0**.
- 4 Hence there are  $k$  paths  $P_1, P_2, \dots, P_k$  with flow of **1** each.
- 5 Paths are edge-disjoint since capacities are **1**. □

# Running Time

## Theorem

The number of edge disjoint paths in a simple graph  $G$  can be found in  $O(mn)$  time.

## Proof.

- 1 Set capacities of edges in  $G$  to 1.
- 2 Run Ford-Fulkerson algorithm.
- 3 Maximum value of flow is  $n$  and hence run-time is  $O(nm)$ .
- 4 Decompose flow into  $k$  paths ( $k \leq n$ ).  
Takes  $O(k \times m) = O(km) = O(mn)$  time. □

## Remark

The algorithm also computes a set of edge-disjoint paths realizing this optimal solution.

# Running Time

## Theorem

The number of edge disjoint paths in a simple graph  $G$  can be found in  $O(mn)$  time.

## Proof.

- 1 Set capacities of edges in  $G$  to 1.
- 2 Run Ford-Fulkerson algorithm.
- 3 Maximum value of flow is  $n$  and hence run-time is  $O(nm)$ .
- 4 Decompose flow into  $k$  paths ( $k \leq n$ ).  
Takes  $O(k \times m) = O(km) = O(mn)$  time. □

## Remark

The algorithm also computes a set of edge-disjoint paths realizing this optimal solution.

# Menger's Theorem

## Theorem (**Menger [1927]**)

Let  $G$  be a directed graph. The minimum number of edges whose removal disconnects  $s$  from  $t$  (the minimum-cut between  $s$  and  $t$ ) is equal to the maximum number of edge-disjoint paths in  $G$  between  $s$  and  $t$ .

## Proof.

Maxflow-mincut theorem and integrality of flow. □

Menger proved his theorem before Maxflow-Mincut theorem!  
Maxflow-Mincut theorem is a generalization of Menger's theorem to capacitated graphs.

# Menger's Theorem

## Theorem (**Menger [1927]**)

Let  $G$  be a directed graph. The minimum number of edges whose removal disconnects  $s$  from  $t$  (the minimum-cut between  $s$  and  $t$ ) is equal to the maximum number of edge-disjoint paths in  $G$  between  $s$  and  $t$ .

## Proof.

Maxflow-mincut theorem and integrality of flow. □

Menger proved his theorem before Maxflow-Mincut theorem!  
Maxflow-Mincut theorem is a generalization of Menger's theorem to capacitated graphs.



# Menger's Theorem

## Theorem (**Menger [1927]**)

Let  $G$  be a directed graph. The minimum number of edges whose removal disconnects  $s$  from  $t$  (the minimum-cut between  $s$  and  $t$ ) is equal to the maximum number of edge-disjoint paths in  $G$  between  $s$  and  $t$ .

## Proof.

Maxflow-mincut theorem and integrality of flow. □

Menger proved his theorem before Maxflow-Mincut theorem!  
Maxflow-Mincut theorem is a generalization of Menger's theorem to capacitated graphs.

# Edge Disjoint Paths in Undirected Graphs

## Problem

Given an **undirected** graph  $G$ , find the maximum number of edge disjoint paths in  $G$

Reduction:

- 1 create **directed** graph  $H$  by adding directed edges  $(u, v)$  and  $(v, u)$  for each edge  $uv$  in  $G$ .
- 2 compute maximum **s-t** flow in  $H$ .

**Problem:** Both edges  $(u, v)$  and  $(v, u)$  may have non-zero flow!

**Not a Problem!** Can assume maximum flow in  $H$  is acyclic and hence cannot have non-zero flow on both  $(u, v)$  and  $(v, u)$ . Reduction works. See book for more details.

# Edge Disjoint Paths in Undirected Graphs

## Problem

Given an **undirected** graph  $G$ , find the maximum number of edge disjoint paths in  $G$

Reduction:

- 1 create **directed** graph  $H$  by adding directed edges  $(u, v)$  and  $(v, u)$  for each edge  $uv$  in  $G$ .
- 2 compute maximum **s-t** flow in  $H$ .

**Problem:** Both edges  $(u, v)$  and  $(v, u)$  may have non-zero flow!

**Not a Problem!** Can assume maximum flow in  $H$  is acyclic and hence cannot have non-zero flow on both  $(u, v)$  and  $(v, u)$ . Reduction works. See book for more details.

# Edge Disjoint Paths in Undirected Graphs

## Problem

Given an **undirected** graph  $G$ , find the maximum number of edge disjoint paths in  $G$

Reduction:

- 1 create **directed** graph  $H$  by adding directed edges  $(u, v)$  and  $(v, u)$  for each edge  $uv$  in  $G$ .
- 2 compute maximum **s-t** flow in  $H$ .

**Problem:** Both edges  $(u, v)$  and  $(v, u)$  may have non-zero flow!

**Not a Problem!** Can assume maximum flow in  $H$  is acyclic and hence cannot have non-zero flow on both  $(u, v)$  and  $(v, u)$ . Reduction works. See book for more details.

# Edge Disjoint Paths in Undirected Graphs

## Problem

Given an **undirected** graph  $G$ , find the maximum number of edge disjoint paths in  $G$

Reduction:

- 1 create **directed** graph  $H$  by adding directed edges  $(u, v)$  and  $(v, u)$  for each edge  $uv$  in  $G$ .
- 2 compute maximum **s-t** flow in  $H$ .

**Problem:** Both edges  $(u, v)$  and  $(v, u)$  may have non-zero flow!

**Not a Problem!** Can assume maximum flow in  $H$  is acyclic and hence cannot have non-zero flow on both  $(u, v)$  and  $(v, u)$ . Reduction works. See book for more details.

# Node Disjoint Paths

## Definition

A set of **s-t** paths  $\mathcal{P}$  are *internally* node-disjoint if no two paths in  $\mathcal{P}$  share a node other than **s**, **t**.

## Theorem (Menger [1927])

Let  $\mathbf{G}$  be an undirected graph. The minimum number of nodes in  $\mathbf{V} \setminus \{\mathbf{s}, \mathbf{t}\}$  whose removal disconnects **s** from **t** is equal to the maximum number of internally node-disjoint paths in  $\mathbf{G}$  between **s** and **t**.

## Theorem

The max number of internally node-disjoint paths between **s** and **t** in  $\mathbf{G}$  can be computed in  $\mathbf{O}(mn)$  time.

Via reductions to directed graph *edge*-disjoint case!

# Node Disjoint Paths

## Definition

A set of  $s$ - $t$  paths  $\mathcal{P}$  are *internally* node-disjoint if no two paths in  $\mathcal{P}$  share a node other than  $s, t$ .

## Theorem (Menger [1927])

Let  $G$  be an undirected graph. The minimum number of nodes in  $V \setminus \{s, t\}$  whose removal disconnects  $s$  from  $t$  is equal to the maximum number of internally node-disjoint paths in  $G$  between  $s$  and  $t$ .

## Theorem

The max number of internally node-disjoint paths between  $s$  and  $t$  in  $G$  can be computed in  $O(mn)$  time.

Via reductions to directed graph *edge*-disjoint case!

# Node Disjoint Paths

## Definition

A set of  $s$ - $t$  paths  $\mathcal{P}$  are *internally* node-disjoint if no two paths in  $\mathcal{P}$  share a node other than  $s, t$ .

## Theorem (Menger [1927])

Let  $G$  be an undirected graph. The minimum number of nodes in  $V \setminus \{s, t\}$  whose removal disconnects  $s$  from  $t$  is equal to the maximum number of internally node-disjoint paths in  $G$  between  $s$  and  $t$ .

## Theorem

The max number of internally node-disjoint paths between  $s$  and  $t$  in  $G$  can be computed in  $O(mn)$  time.

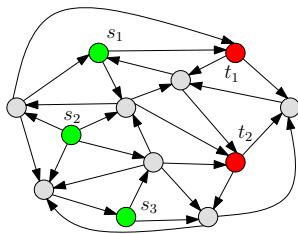
Via reductions to directed graph *edge*-disjoint case!



# Multiple Sources and Sinks

Input:

- 1 A directed graph  $\mathbf{G}$  with edge capacities  $\mathbf{c}(\mathbf{e})$ .
- 2 Source nodes  $\mathbf{s}_1, \mathbf{s}_2, \dots, \mathbf{s}_k$ .
- 3 Sink nodes  $\mathbf{t}_1, \mathbf{t}_2, \dots, \mathbf{t}_\ell$ .
- 4 Sources and sinks are *disjoint*.



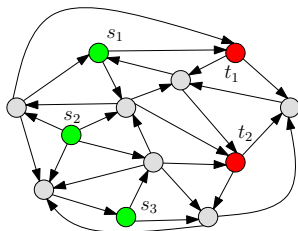
**Maximum Flow:** Send as much flow as possible from the sources to the sinks. *Sinks don't care which source they get flow from.*

**Minimum Cut:** Find a minimum capacity set of edge  $\mathbf{E}'$  such that removing  $\mathbf{E}'$  disconnects every source from every sink.

# Multiple Sources and Sinks

Input:

- 1 A directed graph  $\mathbf{G}$  with edge capacities  $\mathbf{c}(\mathbf{e})$ .
- 2 Source nodes  $\mathbf{s}_1, \mathbf{s}_2, \dots, \mathbf{s}_k$ .
- 3 Sink nodes  $\mathbf{t}_1, \mathbf{t}_2, \dots, \mathbf{t}_\ell$ .
- 4 Sources and sinks are *disjoint*.



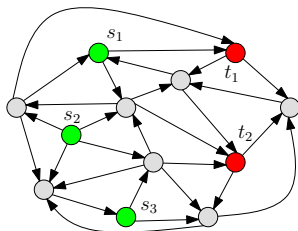
**Maximum Flow:** Send as much flow as possible from the sources to the sinks. *Sinks don't care which source they get flow from.*

**Minimum Cut:** Find a minimum capacity set of edge  $\mathbf{E}'$  such that removing  $\mathbf{E}'$  disconnects every source from every sink.

# Multiple Sources and Sinks

Input:

- 1 A directed graph  $\mathbf{G}$  with edge capacities  $\mathbf{c}(\mathbf{e})$ .
- 2 Source nodes  $\mathbf{s}_1, \mathbf{s}_2, \dots, \mathbf{s}_k$ .
- 3 Sink nodes  $\mathbf{t}_1, \mathbf{t}_2, \dots, \mathbf{t}_\ell$ .
- 4 Sources and sinks are *disjoint*.



**Maximum Flow:** Send as much flow as possible from the sources to the sinks. *Sinks don't care which source they get flow from.*

**Minimum Cut:** Find a minimum capacity set of edge  $\mathbf{E}'$  such that removing  $\mathbf{E}'$  disconnects every source from every sink.

# Multiple Sources and Sinks: Formal Definition

Input:

- 1 A directed graph  $\mathbf{G}$  with edge capacities  $\mathbf{c}(\mathbf{e})$ .
- 2 Source nodes  $\mathbf{s}_1, \mathbf{s}_2, \dots, \mathbf{s}_k$ .
- 3 Sink nodes  $\mathbf{t}_1, \mathbf{t}_2, \dots, \mathbf{t}_\ell$ .
- 4 Sources and sinks are *disjoint*.

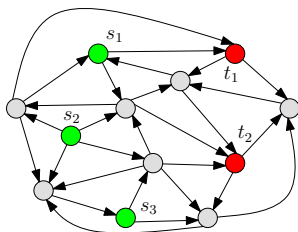
A function  $\mathbf{f} : \mathbf{E} \rightarrow \mathbb{R}^{\geq 0}$  is a **flow** if:

- 1 For each  $\mathbf{e} \in \mathbf{E}$ ,  $\mathbf{f}(\mathbf{e}) \leq \mathbf{c}(\mathbf{e})$ , and
- 2 for each  $\mathbf{v}$  which is not a source or a sink  $\mathbf{f}^{\text{in}}(\mathbf{v}) = \mathbf{f}^{\text{out}}(\mathbf{v})$ .

**Goal:**  $\max \sum_{i=1}^k (\mathbf{f}^{\text{out}}(\mathbf{s}_i) - \mathbf{f}^{\text{in}}(\mathbf{s}_i))$ , that is, flow out of sources.

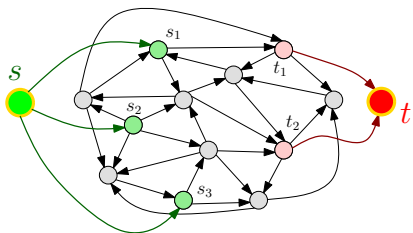
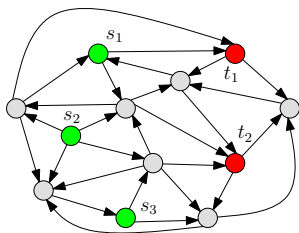
# Reduction to Single-Source Single-Sink

- 1 Add a **source** node **s** and a **sink** node **t**.
- 2 Add edges  $(s, s_1), (s, s_2), \dots, (s, s_k)$ .
- 3 Add edges  $(t_1, t), (t_2, t), \dots, (t_\ell, t)$ .
- 4 Set the capacity of the new edges to be  $\infty$ .



# Reduction to Single-Source Single-Sink

- 1 Add a **source** node **s** and a **sink** node **t**.
- 2 Add edges  $(s, s_1), (s, s_2), \dots, (s, s_k)$ .
- 3 Add edges  $(t_1, t), (t_2, t), \dots, (t_\ell, t)$ .
- 4 Set the capacity of the new edges to be  $\infty$ .

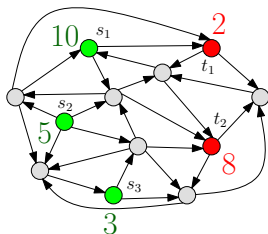


# Supplies and Demands

A further generalization:

- 1 source  $s_i$  has a supply of  $S_i \geq 0$
- 2 since  $t_j$  has a demand of  $D_j \geq 0$  units

**Question:** is there a flow from source to sinks such that supplies are not exceeded and demands are met? Formally we have the additional constraints that  $f^{\text{out}}(s_i) - f^{\text{in}}(s_i) \leq S_i$  for each source  $s_i$  and  $f^{\text{in}}(t_j) - f^{\text{out}}(t_j) \geq D_j$  for each sink  $t_j$ .

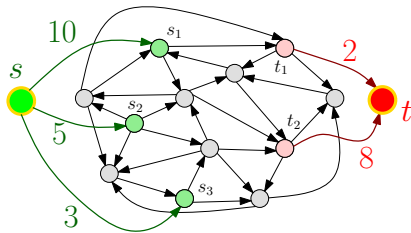
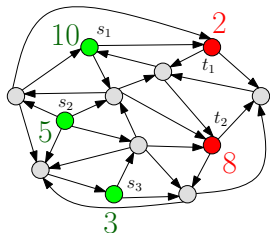


# Supplies and Demands

A further generalization:

- 1 source  $s_i$  has a supply of  $S_i \geq 0$
- 2 since  $t_j$  has a demand of  $D_j \geq 0$  units

**Question:** is there a flow from source to sinks such that supplies are not exceeded and demands are met? Formally we have the additional constraints that  $f^{\text{out}}(s_i) - f^{\text{in}}(s_i) \leq S_i$  for each source  $s_i$  and  $f^{\text{in}}(t_j) - f^{\text{out}}(t_j) \geq D_j$  for each sink  $t_j$ .





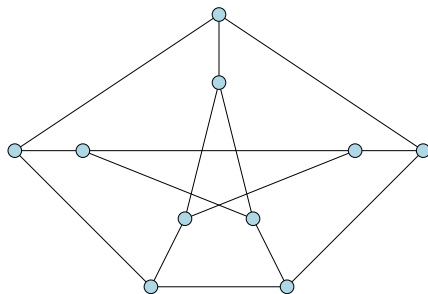
# Matching

## Problem (Matching)

**Input:** Given a (undirected) graph  $G = (V, E)$ .

**Goal:** Find a matching of maximum cardinality.

- 1 A matching is  $M \subseteq E$  such that at most one edge in  $M$  is incident on any vertex



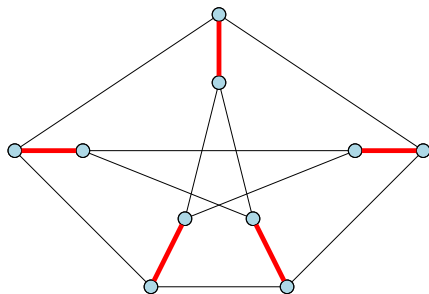
# Matching

## Problem (Matching)

**Input:** Given a (undirected) graph  $G = (V, E)$ .

**Goal:** Find a matching of maximum cardinality.

- 1 A matching is  $M \subseteq E$  such that at most one edge in  $M$  is incident on any vertex

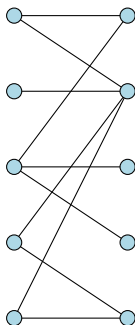


# Bipartite Matching

## Problem (Bipartite matching)

**Input:** Given a bipartite graph  $G = (L \cup R, E)$ .

**Goal:** Find a matching of maximum cardinality



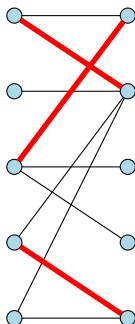
Maximum matching has 4 edges

# Bipartite Matching

## Problem (Bipartite matching)

**Input:** Given a bipartite graph  $G = (L \cup R, E)$ .

**Goal:** Find a matching of maximum cardinality



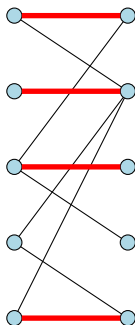
Maximum matching has 4 edges

# Bipartite Matching

## Problem (Bipartite matching)

**Input:** Given a bipartite graph  $G = (L \cup R, E)$ .

**Goal:** Find a matching of maximum cardinality

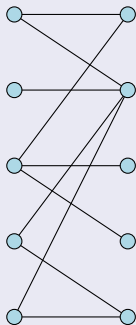


Maximum matching has 4 edges

# Reduction of bipartite matching to max-flow

## Max-Flow Construction

Given graph  $G = (L \cup R, E)$  create flow-network  $G' = (V', E')$  as follows:

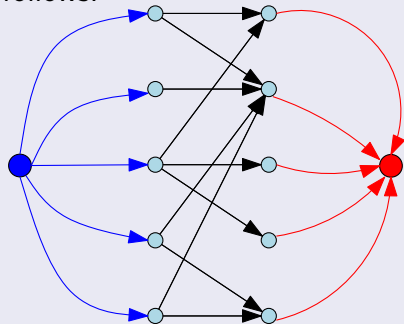


- 1  $V' = L \cup R \cup \{s, t\}$  where  $s$  and  $t$  are the new source and sink.
- 2 Direct all edges in  $E$  from  $L$  to  $R$ , and add edges from  $s$  to all vertices in  $L$  and from each vertex in  $R$  to  $t$ .
- 3 Capacity of every edge is 1.

# Reduction of bipartite matching to max-flow

## Max-Flow Construction

Given graph  $G = (L \cup R, E)$  create flow-network  $G' = (V', E')$  as follows:

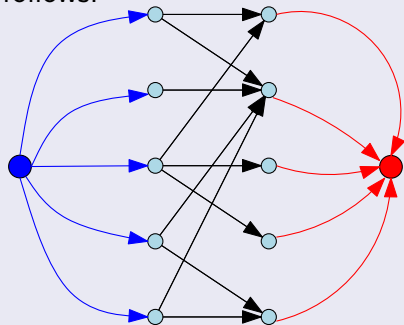


- 1  $V' = L \cup R \cup \{s, t\}$  where  $s$  and  $t$  are the new source and sink.
- 2 Direct all edges in  $E$  from  $L$  to  $R$ , and add edges from  $s$  to all vertices in  $L$  and from each vertex in  $R$  to  $t$ .
- 3 Capacity of every edge is  $1$ .

# Reduction of bipartite matching to max-flow

## Max-Flow Construction

Given graph  $G = (L \cup R, E)$  create flow-network  $G' = (V', E')$  as follows:



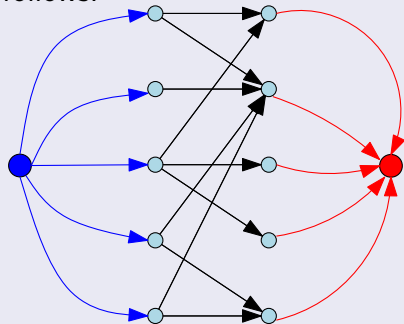
- 1  $V' = L \cup R \cup \{s, t\}$  where  $s$  and  $t$  are the new source and sink.
- 2 Direct all edges in  $E$  from  $L$  to  $R$ , and add edges from  $s$  to all vertices in  $L$  and from each vertex in  $R$  to  $t$ .
- 3 Capacity of every edge is 1.



# Reduction of bipartite matching to max-flow

## Max-Flow Construction

Given graph  $G = (L \cup R, E)$  create flow-network  $G' = (V', E')$  as follows:



- 1  $V' = L \cup R \cup \{s, t\}$  where  $s$  and  $t$  are the new source and sink.
- 2 Direct all edges in  $E$  from  $L$  to  $R$ , and add edges from  $s$  to all vertices in  $L$  and from each vertex in  $R$  to  $t$ .
- 3 Capacity of every edge is  $1$ .

# Correctness: Matching to Flow

## Proposition

If  $G$  has a matching of size  $k$  then  $G'$  has a flow of value  $k$ .

## Proof.

Let  $M$  be matching of size  $k$ . Let  $M = \{(u_1, v_1), \dots, (u_k, v_k)\}$ . Consider following flow  $f$  in  $G'$ :

- 1  $f(s, u_i) = 1$  and  $f(v_i, t) = 1$  for  $1 \leq i \leq k$
- 2  $f(u_i, v_i) = 1$  for  $1 \leq i \leq k$
- 3 for all other edges flow is zero.

Verify that  $f$  is a flow of value  $k$  (because  $M$  is a matching). □

# Correctness: Matching to Flow

## Proposition

If  $G$  has a matching of size  $k$  then  $G'$  has a flow of value  $k$ .

## Proof.

Let  $M$  be matching of size  $k$ . Let  $M = \{(u_1, v_1), \dots, (u_k, v_k)\}$ . Consider following flow  $f$  in  $G'$ :

- 1  $f(s, u_i) = 1$  and  $f(v_i, t) = 1$  for  $1 \leq i \leq k$
- 2  $f(u_i, v_i) = 1$  for  $1 \leq i \leq k$
- 3 for all other edges flow is zero.

Verify that  $f$  is a flow of value  $k$  (because  $M$  is a matching). □

# Correctness: Flow to Matching

## Proposition

If  $G'$  has a flow of value  $k$  then  $G$  has a matching of size  $k$ .

## Proof.

Consider flow  $f$  of value  $k$ .

- 1 Can assume  $f$  is integral. Thus each edge has flow  $1$  or  $0$ .
- 2 Consider the set  $M$  of edges from  $L$  to  $R$  that have flow  $1$ .
  - 1  $M$  has  $k$  edges because value of flow is equal to the number of non-zero flow edges crossing cut  $(L \cup \{s\}, R \cup \{t\})$
  - 2 Each vertex has at most one edge in  $M$  incident upon it. Why?



# Correctness of Reduction

## Theorem

*The maximum flow value in  $G'$  = maximum cardinality of matching in  $G$ .*

## Consequence

Thus, to find maximum cardinality matching in  $G$ , we construct  $G'$  and find the maximum flow in  $G'$ . Note that the matching itself (not just the value) can be found efficiently from the flow.

# Running Time

For graph  $G$  with  $n$  vertices and  $m$  edges  $G'$  has  $O(n + m)$  edges, and  $O(n)$  vertices.

- ① Generic Ford-Fulkerson: Running time is  $O(mC) = O(nm)$  since  $C = n$ .
- ② Capacity scaling: Running time is  $O(m^2 \log C) = O(m^2 \log n)$ .

Better running time is known:  $O(m\sqrt{n})$ .

# Running Time

For graph  $G$  with  $n$  vertices and  $m$  edges  $G'$  has  $O(n + m)$  edges, and  $O(n)$  vertices.

- ① Generic Ford-Fulkerson: Running time is  $O(mC) = O(nm)$  since  $C = n$ .
- ② Capacity scaling: Running time is  $O(m^2 \log C) = O(m^2 \log n)$ .

Better running time is known:  $O(m\sqrt{n})$ .

# 3sided matching

## Clicker question

Given sets **A**, **B**, **C** of size **n**, and a set of triples  $\mathbf{T} = \left\{ (x_i, y_i, z_i) \in \mathbf{A} \times \mathbf{B} \times \mathbf{C} \mid i = 1, \dots, m \right\}$ , one can compute a maximum number of disjoint triplets in **T**, by building the natural **3** layers graph as before (with an additional source and sink), and computing max-flow in this graph. This algorithm is

- (A) incorrect because one can not compute a max flow in such graph.
- (B) incorrect because of some other reason.
- (C) correct.
- (D) correct if you assume no element in **A**, **B** and **C** participates in more than **2** triplets.
- (E) None of the above.



# Perfect Matchings

## Definition

A matching  $M$  is said to be **perfect** if every vertex has one edge in  $M$  incident upon it.

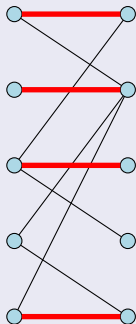


Figure : This graph does not have a perfect matching

# Characterizing Perfect Matchings

## Problem

When does a bipartite graph have a perfect matching?

- 1 Clearly  $|L| = |R|$
- 2 Are there any necessary and sufficient conditions?

# A Necessary Condition

## Lemma

*If  $G = (L \cup R, E)$  has a perfect matching then for any  $X \subseteq L$ ,  $|N(X)| \geq |X|$ , where  $N(X)$  is the set of neighbors of vertices in  $X$ .*

## Proof.

Since  $G$  has a perfect matching, every vertex of  $X$  is matched to a different neighbor, and so  $|N(X)| \geq |X|$ . □

# A Necessary Condition

## Lemma

*If  $G = (L \cup R, E)$  has a perfect matching then for any  $X \subseteq L$ ,  $|N(X)| \geq |X|$ , where  $N(X)$  is the set of neighbors of vertices in  $X$ .*

## Proof.

Since  $G$  has a perfect matching, every vertex of  $X$  is matched to a different neighbor, and so  $|N(X)| \geq |X|$ . □

# Hall's Theorem

## Theorem (Frobenius-Hall)

Let  $\mathbf{G} = (\mathbf{L} \cup \mathbf{R}, \mathbf{E})$  be a bipartite graph with  $|\mathbf{L}| = |\mathbf{R}|$ .  $\mathbf{G}$  has a perfect matching if and only if for every  $\mathbf{X} \subseteq \mathbf{L}$ ,  $|\mathbf{N}(\mathbf{X})| \geq |\mathbf{X}|$ .

One direction is the necessary condition.

For the other direction we will show the following:

- 1 Create flow network  $\mathbf{G}'$  from  $\mathbf{G}$ .
- 2 If  $|\mathbf{N}(\mathbf{X})| \geq |\mathbf{X}|$  for all  $\mathbf{X}$ , show that minimum  $\mathbf{s-t}$  cut in  $\mathbf{G}'$  is of capacity  $\mathbf{n} = |\mathbf{L}| = |\mathbf{R}|$ .
- 3 Implies that  $\mathbf{G}$  has a perfect matching.

# Hall's Theorem

## Theorem (Frobenius-Hall)

Let  $\mathbf{G} = (\mathbf{L} \cup \mathbf{R}, \mathbf{E})$  be a bipartite graph with  $|\mathbf{L}| = |\mathbf{R}|$ .  $\mathbf{G}$  has a perfect matching if and only if for every  $\mathbf{X} \subseteq \mathbf{L}$ ,  $|\mathbf{N}(\mathbf{X})| \geq |\mathbf{X}|$ .

One direction is the necessary condition.

For the other direction we will show the following:

- 1 Create flow network  $\mathbf{G}'$  from  $\mathbf{G}$ .
- 2 If  $|\mathbf{N}(\mathbf{X})| \geq |\mathbf{X}|$  for all  $\mathbf{X}$ , show that minimum  $\mathbf{s-t}$  cut in  $\mathbf{G}'$  is of capacity  $\mathbf{n} = |\mathbf{L}| = |\mathbf{R}|$ .
- 3 Implies that  $\mathbf{G}$  has a perfect matching.

# Proof of Sufficiency

Assume  $|\mathbf{N}(\mathbf{X})| \geq |\mathbf{X}|$  for any  $\mathbf{X} \subseteq \mathbf{L}$ . Then show that min **s-t** cut in  $\mathbf{G}'$  is of capacity at least  $\mathbf{n}$ .

Let  $(\mathbf{A}, \mathbf{B})$  be an *arbitrary* **s-t** cut in  $\mathbf{G}'$

- 1 Let  $\mathbf{X} = \mathbf{A} \cap \mathbf{L}$  and  $\mathbf{Y} = \mathbf{A} \cap \mathbf{R}$ .
- 2 Cut capacity is at least  $(|\mathbf{L}| - |\mathbf{X}|) + |\mathbf{Y}| + |\mathbf{N}(\mathbf{X}) \setminus \mathbf{Y}|$

# Proof of Sufficiency

Assume  $|N(X)| \geq |X|$  for any  $X \subseteq L$ . Then show that min **s-t** cut in  $G'$  is of capacity at least **n**.

Let  $(A, B)$  be an *arbitrary* **s-t** cut in  $G'$

- 1 Let  $X = A \cap L$  and  $Y = A \cap R$ .
- 2 Cut capacity is at least  $(|L| - |X|) + |Y| + |N(X) \setminus Y|$



# Proof of Sufficiency

Assume  $|N(X)| \geq |X|$  for any  $X \subseteq L$ . Then show that min **s-t** cut in  $G'$  is of capacity at least  $n$ .

Let  $(A, B)$  be an *arbitrary* **s-t** cut in  $G'$

① Let  $X = A \cap L$  and  $Y = A \cap R$ .

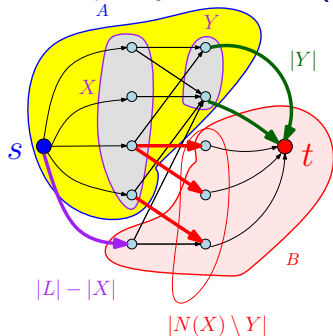
② Cut capacity is at least  $(|L| - |X|) + |Y| + |N(X) \setminus Y|$

# Proof of Sufficiency

Assume  $|\mathbf{N}(\mathbf{X})| \geq |\mathbf{X}|$  for any  $\mathbf{X} \subseteq \mathbf{L}$ . Then show that min  $\mathbf{s-t}$  cut in  $\mathbf{G}'$  is of capacity at least  $\mathbf{n}$ .

Let  $(\mathbf{A}, \mathbf{B})$  be an arbitrary  $\mathbf{s-t}$  cut in  $\mathbf{G}'$

- 1 Let  $\mathbf{X} = \mathbf{A} \cap \mathbf{L}$  and  $\mathbf{Y} = \mathbf{A} \cap \mathbf{R}$ .
- 2 Cut capacity is at least  $(|\mathbf{L}| - |\mathbf{X}|) + |\mathbf{Y}| + |\mathbf{N}(\mathbf{X}) \setminus \mathbf{Y}|$



Because there are...

- 1  $|\mathbf{L}| - |\mathbf{X}|$  edges from  $\mathbf{s}$  to  $\mathbf{L} \cap \mathbf{B}$ .
- 2  $|\mathbf{Y}|$  edges from  $\mathbf{Y}$  to  $\mathbf{t}$ .
- 3 there are at least  $|\mathbf{N}(\mathbf{X}) \setminus \mathbf{Y}|$  edges from  $\mathbf{X}$  to vertices on the right side that are not in  $\mathbf{Y}$ .

# Proof of Sufficiency

Continued...

- ① By the above, cut capacity is at least

$$\alpha = (|L| - |X|) + |Y| + |N(X) \setminus Y|.$$

- ②  $|N(X) \setminus Y| \geq |N(X)| - |Y|$ .

(This holds for any two sets.)

- ③ By assumption  $|N(X)| \geq |X|$  and hence

$$|N(X) \setminus Y| \geq |N(X)| - |Y| \geq |X| - |Y|.$$

- ④ Cut capacity is therefore at least

$$\begin{aligned} \alpha &= (|L| - |X|) + |Y| + |N(X) \setminus Y| \\ &\geq |L| - |X| + |Y| + |X| - |Y| \geq |L| = n. \end{aligned}$$

- ⑤ Any  $s$ - $t$  cut capacity is at least  $n \implies$  max flow at least  $n$  units  $\implies$  perfect matching.

**QED**

# Hall's Theorem: Generalization

## Theorem (Frobenius-Hall)

Let  $\mathbf{G} = (\mathbf{L} \cup \mathbf{R}, \mathbf{E})$  be a bipartite graph with  $|\mathbf{L}| \leq |\mathbf{R}|$ .  $\mathbf{G}$  has a matching that matches all nodes in  $\mathbf{L}$  if and only if for every  $\mathbf{X} \subseteq \mathbf{L}$ ,  $|\mathbf{N}(\mathbf{X})| \geq |\mathbf{X}|$ .

Proof is essentially the same as the previous one.

# Assigning jobs to people

- 1  $n$  jobs,  $n/2$  people
- 2 For each job: a set of people who can do that job.
- 3 Each person  $j$  has to do exactly two jobs.
- 4 **Goal:** find an assignment of 2 jobs to each person, such that all jobs are assigned.

**Solution:** Build bipartite graph, compute maximum matching, remove it, compute another maximum matching. Both matchings together form a valid solution if it exists. This algorithm is

(A) Correct.

(B) Incorrect.

# Application: Assigning jobs to people

- 1  $n$  jobs or tasks
- 2  $m$  people
- 3 for each job a set of people who can do that job
- 4 for each person  $j$  a limit on number of jobs  $k_j$
- 5 **Goal:** find an assignment of jobs to people so that all jobs are assigned and no person is overloaded

Reduce to max-flow similar to matching.

Arises in many settings. Using *minimum-cost flows* can also handle the case when assigning a job  $i$  to person  $j$  costs  $c_{ij}$  and goal is assign all jobs but minimize cost of assignment.

# Application: Assigning jobs to people

- 1  $n$  jobs or tasks
- 2  $m$  people
- 3 for each job a set of people who can do that job
- 4 for each person  $j$  a limit on number of jobs  $k_j$
- 5 **Goal:** find an assignment of jobs to people so that all jobs are assigned and no person is overloaded

Reduce to max-flow similar to matching.

Arises in many settings. Using *minimum-cost flows* can also handle the case when assigning a job  $i$  to person  $j$  costs  $c_{ij}$  and goal is assign all jobs but minimize cost of assignment.

# Reduction to Maximum Flow

- 1 Create directed graph  $G = (V, E)$  as follows
  - 1  $V = \{s, t\} \cup L \cup R$ :  $L$  set of  $n$  jobs,  $R$  set of  $m$  people
  - 2 add edges  $(s, i)$  for each job  $i \in L$ , capacity  $1$
  - 3 add edges  $(j, t)$  for each person  $j \in R$ , capacity  $k_j$
  - 4 if job  $i$  can be done by person  $j$  add an edge  $(i, j)$ , capacity  $1$
- 2 Compute max  $s$ - $t$  flow. There is an assignment if and only if flow value is  $n$ .



# Matchings in General Graphs

Matchings in general graphs more complicated.

There is a polynomial time algorithm to compute a maximum matching in a general graph. Best known running time is  $O(m\sqrt{n})$ .

# Notes

# Notes

# Notes



Menger, K. (1927). Zur allgemeinen Kruventheorie. *Fund. Math.*, 10:96–115.