## CS 473: Fundamental Algorithms, Spring 2013

# Applications of Network Flows

Lecture 18

March 28, 2013



## 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 $\mathbf{O(mC)}$ time when capacities are integral and **C** is an upper bound on the flow.
   2. Using variant of algorithm, in $\mathbf{O(m^2 \log C)}$ time, when capacities are integral. (Polynomial time.)
   3. Using Edmonds-Karp algorithm, in $\mathbf{O(m^2 n)}$ 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 $\mathbf{O(m+n)}$ 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 $\mathbf{O(m+n)}$ time given any maximum flow.

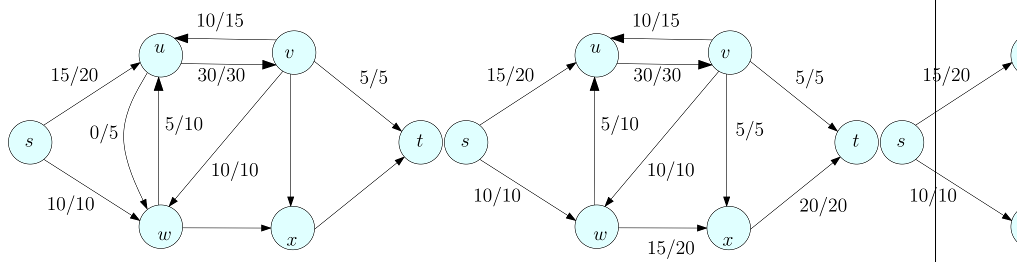


## Paths, Cycles and Acyclicity of Flows

### Definition

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

Question: Given a flow **f**, can there by cycles in its support?



# Acyclicity of Flows

## Proposition

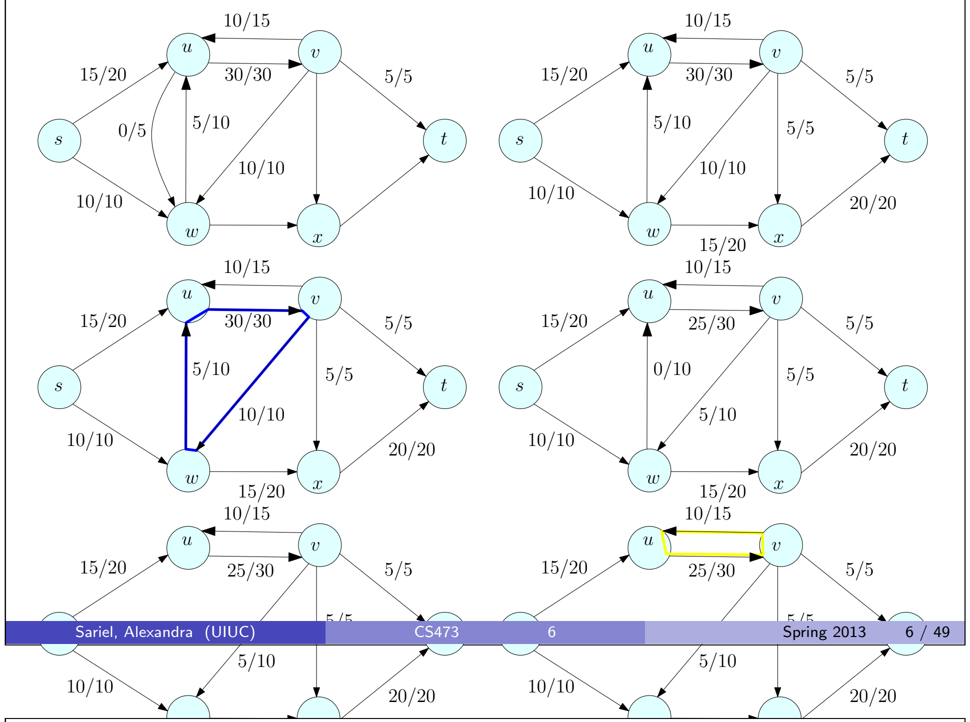
*In any flow network, if* $\mathbf{f}$ *is a flow then there is another flow* $\mathbf{f'}$ *such that the support of* $\mathbf{f'}$ *is an acyclic graph and* $\mathbf{v(f') = v(f)}$. *Further if* $\mathbf{f}$ *is an integral flow then so is* $\mathbf{f'}$.

## Proof.

1. $\mathbf{E' = \{e \in E \mid f(e) > 0\}}$, 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 $\mathbf{e \in C}$, reduce flow by $\mathbf{f(e')}$. 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






# Flow Decomposition

## Lemma

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

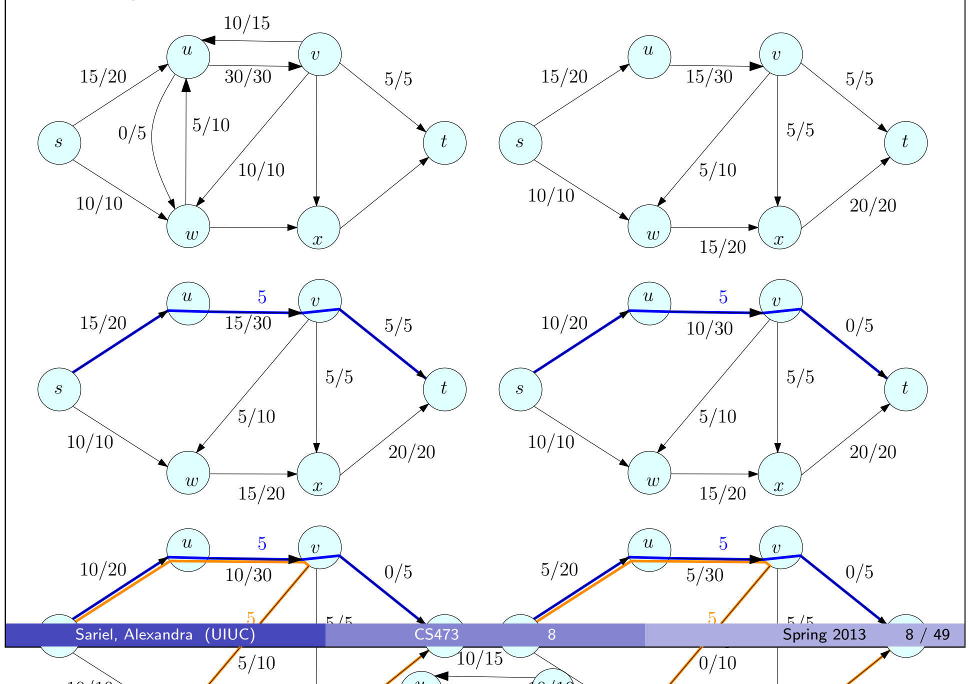
1. $|\mathcal{P} \cup \mathcal{C}| \leq \mathbf{m}$
2. *for each* $\mathbf{e \in E}$, $\sum_{\mathbf{P} \in \mathcal{P}: \mathbf{e} \in \mathbf{P}} \mathbf{f'(P)} + \sum_{\mathbf{C} \in \mathcal{C}: \mathbf{e} \in \mathbf{C}} \mathbf{f'(C)} = \mathbf{f(e)}$
3. $\mathbf{v(f)} = \sum_{\mathbf{P} \in \mathcal{P}}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



## Flow Decomposition

### Lemma

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

1. $|\mathcal{P} \cup \mathcal{C}| \leq \mathbf{m}$
2. *for each* $\mathbf{e} \in \mathbf{E}$, $\sum_{P \in \mathcal{P}: e \in P} \mathbf{f}'(\mathbf{P}) + \sum_{C \in \mathcal{C}: e \in C} \mathbf{f}'(\mathbf{C}) = \mathbf{f}(\mathbf{e})$
3. $\mathbf{v}(\mathbf{f}) = \sum_{P \in \mathcal{P}} \mathbf{f}'(\mathbf{P})$.
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* $\mathbf{O}(\mathbf{m}^2)$ *time.*



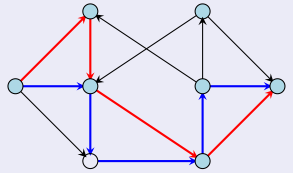
# Part I

## 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 $\mathbf{s}$ and $\mathbf{t}$, find the *maximum* number of edge disjoint paths from $\mathbf{s}$ to $\mathbf{t}$.

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 $\mathbf{G}$ with two special vertices $\mathbf{s}$ and $\mathbf{t}$, find the maximum number of edge disjoint paths from $\mathbf{s}$ to $\mathbf{t}$.

### Reduction

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

## Correctness of Reduction

### Lemma

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

### Proof.

Set $\mathbf{f(e) = 1}$ if $\mathbf{e}$ belongs to one of the paths $\mathbf{P_1, P_2, \ldots, P_k}$; other-wise set $\mathbf{f(e) = 0}$. This defines a flow of value $\mathbf{k}$. □



## Correctness of Reduction

### Lemma

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

### Proof.

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



## Running Time

### Theorem

*The number of edge disjoint paths in $\mathbf{G}$ can be found in $\mathbf{O(mn)}$ time.*

### Proof.

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

### Remark

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



## Menger's Theorem

### Theorem (Menger [1927])

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

### 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!

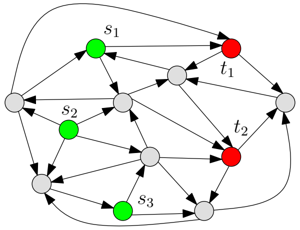
Not a Problem! Can assume maximum flow in **H** is acyclic and hence cannot have non-zero flow on both $(u, v)$ and $(v, u)$. Reduction works. See book for more details.



# Multiple Sources and Sinks

Input:

1. A directed graph **G** with edge capacities $c(e)$.
2. Source nodes $s_1, s_2, \ldots, s_k$.
3. Sink nodes $t_1, t_2, \ldots, t_\ell$.
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 $E'$ such that removing $E'$ disconnects every source from every sink.



# Multiple Sources and Sinks: Formal Definition

Input:

1. A directed graph **G** with edge capacities $c(e)$.
2. Source nodes $s_1, s_2, \ldots, s_k$.
3. Sink nodes $t_1, t_2, \ldots, t_\ell$.
4. Sources and sinks are *disjoint*.

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

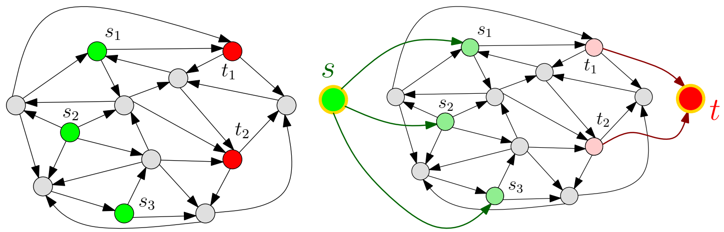
1. For each $e \in E$, $f(e) \leq c(e)$, and
2. for each $v$ which is not a source or a sink $f^{in}(v) = f^{out}(v)$.

Goal: max $\sum_{i=1}^{k} (f^{out}(s_i) - f^{in}(s_i))$, 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 $(s, s_1), (s, s_2), \ldots, (s, s_k)$.
3. Add edges $(t_1, t), (t_2, t), \ldots, (t_\ell, t)$.
4. Set the capacity of the new edges to be $\infty$.

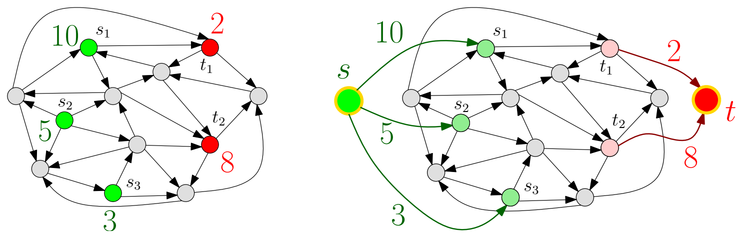


## Supplies and Demands

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 $f^{out}(s_i) - f^{in}(s_i) \leq S_i$ for each source $s_i$ and $f^{in}(t_j) - f^{out}(t_j) \geq D_j$ for each sink $t_j$.



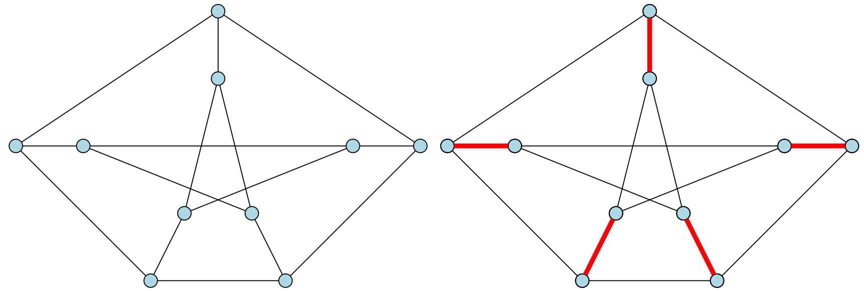



## Matching

### 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*

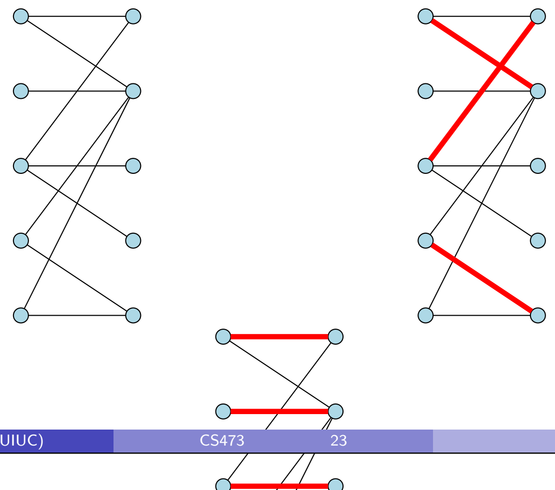




## Bipartite Matching

### Problem (Bipartite matching)

**Input:** *Given a bipartite graph* $G = (L \cup R, E)$.
**Goal:** *Find a matching of maximum cardinality*

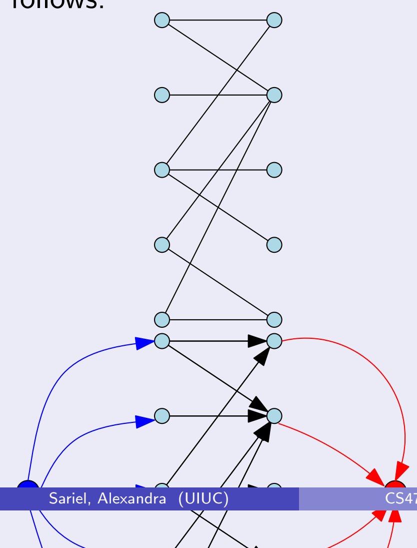




## Reduction of bipartite matching to max-flow

### 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 $\mathbf{G}$ has a matching of size $\mathbf{k}$ then $\mathbf{G}'$ has a flow of value $\mathbf{k}$.*

**Proof.**

Let $\mathbf{M}$ be matching of size $\mathbf{k}$. Let $\mathbf{M} = \{(\mathbf{u}_1, \mathbf{v}_1), \ldots, (\mathbf{u}_k, \mathbf{v}_k)\}$. Consider following flow $\mathbf{f}$ in $\mathbf{G}'$:

1. $\mathbf{f}(\mathbf{s}, \mathbf{u}_i) = 1$ and $\mathbf{f}(\mathbf{v}_i, \mathbf{t}) = 1$ for $1 \leq i \leq k$
2. $\mathbf{f}(\mathbf{u}_i, \mathbf{v}_i) = 1$ for $1 \leq i \leq k$
3. for all other edges flow is zero.

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



## Correctness: Flow to Matching

**Proposition**

*If $\mathbf{G}'$ has a flow of value $\mathbf{k}$ then $\mathbf{G}$ has a matching of size $\mathbf{k}$.*

**Proof.**

Consider flow $\mathbf{f}$ of value $\mathbf{k}$.

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

□



## Correctness of Reduction

**Theorem**

*The maximum flow value in $\mathbf{G}'$ = maximum cardinality of matching in $\mathbf{G}$.*

**Consequence**

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



## Running Time

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

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

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

## Perfect Matchings

### Definition

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

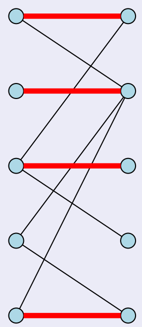


Figure: This graph does not have a perfect matching



## Characterizing Perfect Matchings

### Problem

When does a bipartite graph have a perfect matching?

1. Clearly $|\mathbf{L}| = |\mathbf{R}|$
2. Are there any necessary and sufficient conditions?



## A Necessary Condition

### Lemma

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

### Proof.

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



## 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:

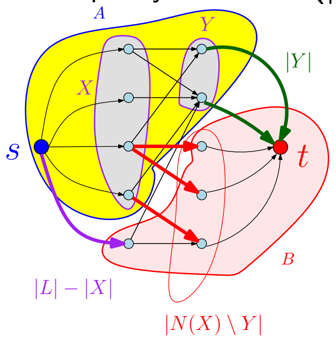
1. Create flow network $\mathbf{G'}$ from $\mathbf{G}$.
2. If $|\mathbf{N}(\mathbf{X})| \geq |\mathbf{X}|$ for all $\mathbf{X}$, show that minimum $\mathbf{s}$-$\mathbf{t}$ cut in $\mathbf{G'}$ is of capacity $\mathbf{n} = |\mathbf{L}| = |\mathbf{R}|$.
3. Implies that $\mathbf{G}$ has a perfect matching.

## 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 **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 $(|\mathbf{L}| - |\mathbf{X}|) + |\mathbf{Y}| + |\mathbf{N}(\mathbf{X}) \setminus \mathbf{Y}|$




**Because there are...**

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



## Proof of Sufficiency

Continued...

1. By the above, cut capacity is at least
$$\alpha = (|\mathbf{L}| - |\mathbf{X}|) + |\mathbf{Y}| + |\mathbf{N}(\mathbf{X}) \setminus \mathbf{Y}|.$$
2. $|\mathbf{N}(\mathbf{X}) \setminus \mathbf{Y}| \geq |\mathbf{N}(\mathbf{X})| - |\mathbf{Y}|$.
(This holds for any two sets.)
3. By assumption $|\mathbf{N}(\mathbf{X})| \geq |\mathbf{X}|$ and hence
$$|\mathbf{N}(\mathbf{X}) \setminus \mathbf{Y}| \geq |\mathbf{N}(\mathbf{X})| - |\mathbf{Y}| \geq |\mathbf{X}| - |\mathbf{Y}|.$$
4. Cut capacity is therefore at least
$$\alpha = (|\mathbf{L}| - |\mathbf{X}|) + |\mathbf{Y}| + |\mathbf{N}(\mathbf{X}) \setminus \mathbf{Y}|$$
$$\geq |\mathbf{L}| - |\mathbf{X}| + |\mathbf{Y}| + |\mathbf{X}| - |\mathbf{Y}| \geq |\mathbf{L}| = \mathbf{n}.$$
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. $\mathbf{n}$ jobs or tasks
2. $\mathbf{m}$ people
3. for each job a set of people who can do that job
4. for each person $\mathbf{j}$ a limit on number of jobs $\mathbf{k_j}$
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 $\mathbf{i}$ to person $\mathbf{j}$ costs $\mathbf{c_{ij}}$ and goal is assign all jobs but minimize cost of assignment.



## Reduction to Maximum Flow

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



## 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 $\mathbf{O(m\sqrt{n})}$.



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