CS 473: Algorithms

Chandra Chekuri
chekuri@cs.illinois.edu
3228 Siebel Center
University of Illinois, Urbana-Champaign

Fall 2009
Network Flow: Facts to Remember

Flow network: directed graph $G$, capacities $c$, source $s$, sink $t$
- maximum $s$-$t$ flow can be computed
Network Flow: Facts to Remember

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

- maximum $s$-$t$ flow can be computed
  - using Ford-Fulkerson algorithm in $O(mC)$ time when capacities are integral and $C$ is an upper bound on the flow
Network Flow: Facts to Remember

Flow network: directed graph $G$, capacities $c$, source $s$, sink $t$
- maximum $s$-$t$ flow can be computed
  - using Ford-Fulkerson algorithm in $O(mC)$ time when capacities are integral and $C$ is an upper bound on the flow
  - using capacity scaling algorithm in $O(m^2 \log C)$ time when capacities are integral

If capacities are integral then there is a maximum flow that is integral and above algorithms give an integral max flow

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

Maximum flow is equal to the minimum cut and minimum cut can be found in $O(m + n)$ time given any maximum flow
Network Flow: Facts to Remember

Flow network: directed graph $G$, capacities $c$, source $s$, sink $t$
- maximum $s$-$t$ flow can be computed
  - using Ford-Fulkerson algorithm in $O(mC)$ time when capacities are integral and $C$ is an upper bound on the flow
  - using capacity scaling algorithm in $O(m^2 \log C)$ time when capacities are integral
  - using Edmonds-Karp algorithm in $O(m^2 n)$ time when capacities are rational (strongly polynomial time algorithm)
Network Flow: Facts to Remember

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

- maximum $s$-$t$ flow can be computed
  - using Ford-Fulkerson algorithm in $O(mC)$ time when capacities are integral and $C$ is an upper bound on the flow
  - using capacity scaling algorithm in $O(m^2 \log C)$ time when capacities are integral
  - using Edmonds-Karp algorithm in $O(m^2 n)$ time when capacities are rational (strongly polynomial time algorithm)

- if capacities are integral then there is a maximum flow that is integral and above algorithms give an integral max flow

if capacities are integral then there is a maximum flow that is integral and above algorithms give an integral max flow
Network Flow: Facts to Remember

Flow network: directed graph $G$, capacities $c$, source $s$, sink $t$
- maximum $s$-$t$ flow can be computed
  - using Ford-Fulkerson algorithm in $O(mC)$ time when capacities are integral and $C$ is an upper bound on the flow
  - using capacity scaling algorithm in $O(m^2 \log C)$ time when capacities are integral
  - using Edmonds-Karp algorithm in $O(m^2 n)$ time when capacities are rational (strongly polynomial time algorithm)
- if capacities are integral then there is a maximum flow that is integral and above algorithms give an integral max flow
- given a flow of value $\nu$, can decompose into $O(m + n)$ flow paths of same total value $\nu$. integral flow implies integral flow on paths
Network Flow: Facts to Remember

Flow network: directed graph $G$, capacities $c$, source $s$, sink $t$
- maximum $s$-$t$ flow can be computed
  - using Ford-Fulkerson algorithm in $O(mC)$ time when capacities are integral and $C$ is an upper bound on the flow
  - using capacity scaling algorithm in $O(m^2 \log C)$ time when capacities are integral
  - using Edmonds-Karp algorithm in $O(m^2 n)$ time when capacities are rational (strongly polynomial time algorithm)
- if capacities are integral then there is a maximum flow that is integral and above algorithms give an integral max flow
- 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
- maximum flow is equal to the minimum cut and minimum cut can be found in $O(m + n)$ time given any maximum flow
Definition

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

Question: Given flow $f$, can there be cycles in its support?
Paths, Cycles and Acyclicity of Flows

Definition

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

Question: Given flow $f$, can there be cycles in its support?
Proposition

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

Proof.

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

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

- \( |\mathcal{P} \cup \mathcal{C}| \leq m \)
- for each \( e \in E \), \( \sum_{P \in \mathcal{P} : e \in P} f'(P) + \sum_{C \in \mathcal{C} : e \in C} f'(C) = f(e) \)
- \( v(f) = \sum_{P \in \mathcal{P}} f'(P) \)
- if \( f \) is integral then so are \( f'(P) \) and \( f'(C) \) for all \( P \) and \( C \)
Flow Decomposition

**Lemma**

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

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

**Proof Idea.**

- First remove all cycles as in previous proposition.
- Then decompose into paths as in previous lecture.
- Exercise: verify claims.
Example

\[ C_1 = u \rightarrow v \rightarrow u \quad \text{and} \quad f(C_1) = 10 \]
\[ C_2 = u \rightarrow v \rightarrow w \rightarrow u \quad \text{and} \quad f(C_2) = 5 \]
\[ \phi_1 : S \rightarrow w \rightarrow x \rightarrow t \quad \text{and} \quad f(\phi_1) = 10 \]
\[ \phi_2 : S \rightarrow u \rightarrow v \rightarrow x \rightarrow t \quad \text{and} \quad f(\phi_2) = 5 \]
\[ \phi_3 : S \rightarrow u \rightarrow v \rightarrow w \rightarrow x \rightarrow t \quad \text{and} \quad f(\phi_3) = 5 \]
\[ \phi_4 : S \rightarrow u \rightarrow v \rightarrow t \quad \text{and} \quad f(\phi_4) = 5 \]
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.
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 find max-flow.
Correctness of Reduction

Lemma

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

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

Chekuri CS473ug
Correctness of Reduction

**Lemma**

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

Proof.

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

Chekuri CS473ug
Correctness of Reduction

**Lemma**

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

**Proof.**

Set $f(e) = 1$ if $e$ belongs to one of the paths $P_1, P_2, \ldots, P_k$; 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 $t$. 
Correctness of Reduction

**Lemma**

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

**Proof.**

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

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

Run Ford-Fulkerson algorithm. Maximum possible flow is $n$ and hence run-time is $O(nm)$. 
Theorem (Menger)

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 $t$. 

Theorem (Menger)

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.
Theorem (Menger)

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 \( G \)
Problem

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

Reduction:

- create directed graph $H$ by adding directed edges $(u, v)$ and $(v, u)$ for each edge $uv$ in $G$.
- compute maximum $s$-$t$ flow in $H$
Edge Disjoint Paths in Undirected Graphs

Problem

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

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

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

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

**Reduction:**

- create directed graph $H$ by adding directed edges $(u, v)$ and $(v, u)$ for each edge $uv$ in $G$.
- 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

- Directed graph $G$ with edge capacities $c(e)$
- Source nodes $s_1, s_2, \ldots, s_k$
- Sink nodes $t_1, t_2, \ldots, t_\ell$
- Sources and sinks are disjoint
Multiple Sources and Sinks

- Directed graph $G$ with edge capacities $c(e)$
- source nodes $s_1, s_2, \ldots, s_k$
- sink nodes $t_1, t_2, \ldots, t_\ell$
- 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

- Directed graph $G$ with edge capacities $c(e)$
- source nodes $s_1, s_2, \ldots, s_k$
- sink nodes $t_1, t_2, \ldots, t_\ell$
- sources and sinks are *disjoint*

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

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

**Goal:** $\max \sum_{i=1}^{k} (f^{\text{out}}(s_i) - f^{\text{in}}(s_i))$, that is, flow out of sources
Add a source node $s$ and a sink node $t$
Add edges $(s, s_1), (s, s_2), \ldots, (s, s_k)$
Add edges $(t_1, t), (t_2, t), \ldots, (t_\ell, t)$
Set the capacity of the new edges to be $\infty$
Supplies and Demands

A further generalization:

- source \( s_i \) has a supply of \( S_i \geq 0 \)
- sink \( t_j \) 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 \( f^\text{in}(t_j) - f^\text{out}(t_j) \leq D_j \) for each sink \( t_j \).
Matching

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

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

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

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

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

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

**Goal**  Find a matching of maximum cardinality

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

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

**Goal**  Find a matching of maximum cardinality

![Diagram of a bipartite graph with matching edges]
**Bipartite Matching**

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

**Goal**  Find a matching of maximum cardinality

![Figure: Maximum matching has 4 edges]
**Bipartite Matching**

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

**Goal** Find a matching of maximum cardinality

*Figure: Maximum matching has 4 edges*
Reduction to Max-Flow

Max-Flow Construction

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

- $V' = L \cup R \cup \{s, t\}$ where $s$ and $t$ are the new source and sink.
- 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 $t$.
- Capacity of every edge is 1.
Given graph $G = (L \cup R, E)$ create flow-network $G' = (V', E')$ as follows:

- $V' = L \cup R \cup \{s, t\}$ where $s$ and $t$ are the new source and sink
Max-Flow Construction

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

- $V' = L \cup R \cup \{s, t\}$ where $s$ and $t$ are the new source and sink
- 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 $t$
Given graph $G = (L \cup R, E)$ create flow-network $G' = (V', E')$ as follows:

- $V' = L \cup R \cup \{s, t\}$ where $s$ and $t$ are the new source and sink
- 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 $t$
- 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 a matching of size $k$. Let $M = \{(u_1, v_1), \ldots, (u_k, v_k)\}$.

Consider the following flow $f$ in $G'$:

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

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

Chekuri CS473ug
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 $M = \{(u_1, v_1), \ldots, (u_k, v_k)\}$. Consider following flow $f$ in $G'$:

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

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

- Can assume $f$ is integral. Thus each edge has flow 1 or 0.
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 $k$.

- Can assume $f$ is integral. Thus each edge has flow 1 or 0.
- Consider the set $M$ of edges from $L$ to $R$ that have flow 1.
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 $k$.

- Can assume $f$ is integral. Thus each edge has flow 1 or 0
- Consider the set $M$ of edges from $L$ to $R$ that have flow 1
  - $M$ has $k$ edges because value of flow is equal to the number of non-zero flow edges crossing cut $(L \cup \{s\}, R \cup \{t\})$
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 \( k \).

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

**Theorem**

*The maximum flow value in $G' = \text{maximum cardinality of matching in } G*
Correctness of Reduction

**Theorem**

The maximum flow value in $G' = \text{maximum cardinality of matching in } G$

**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.

- Generic Ford-Fulkerson: Running time is $O(mC) = O(nm)$ since $C = n$
Running Time

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

- Generic Ford-Fulkerson: Running time is $O(mC) = O(nm)$ since $C = n$
- Capacity scaling: Running time is $O(m^2 \log C) = O(m^2 \log n)$
For graph $G$ with $n$ vertices and $m$ edges $G'$ has $O(n + m)$ edges, and $O(n)$ vertices.

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

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

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

Better known running time: $O(m\sqrt{n})$
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?
Problem

When does a bipartite graph have a perfect matching?

- Clearly $|L| = |R|$
Characterizing Perfect Matchings

**Problem**

When does a bipartite graph have a perfect matching?

- Clearly $|L| = |R|$.
- Are there any necessary and sufficient conditions?
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 $X$. 
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 $G = (L \cup R, E)$ be a bipartite graph with $|L| = |R|$. $G$ has a perfect matching if and only if for every $X \subseteq L$, $|N(X)| \geq |X|$

One direction is the necessary condition.
Hall’s Theorem

**Theorem (Frobenius-Hall)**

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

One direction is the necessary condition. For the other direction we will show the following:

- create flow network $G'$ from $G$
- if $|N(X)| \geq |X|$ for all $X$, show that minimum $s$-$t$ cut in $G'$ is of capacity $n = |L| = |R|$.
- implies that $G$ has a perfect matching.
Proof of Sufficiency

Assume $|N(X)| \geq |X|$ for each $X \in L$. Then show that min $s$-$t$ cut in $G'$ is of capacity $n$. 
Proof of Sufficiency

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

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

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

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

- let $X = A \cap L$ and $Y = A \cap R$
Proof of Sufficiency

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

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

- let $X = A \cap L$ and $Y = A \cap R$
- cut capacity is at least $(|L| - |X|) + |Y| + |N(X) - Y|$
Proof of Sufficiency

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

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

- let $X = A \cap L$ and $Y = A \cap R$
- cut capacity is at least $(|L| - |X|) + |Y| + |N(X) - Y|$
- $|N(X) - Y| \geq |N(X)| - |Y|$ and by assumption $|N(X)| \geq |X|$ and hence $|N(X) - Y| \geq |X| - |Y|$
- cut capacity is therefore at least $|L| - |X| + |Y| + |X| - |Y| \geq |L| = n$. 
\[ X = ANL, \quad Y = ANR \]

\[ c(A, B) \geq (|L| - |X|) + |Y| + \left( |N(X)| - |Y| \right) \]

\[ \geq |L| + \left| |N(X)| - |X| \right| \]

\[ \geq |L| \]
Application: assigning jobs to people

- \( n \) jobs or tasks
- \( m \) people
- for each job a set of people who can do that job
- for each person \( j \) a limit on number of jobs \( k_j \)
- **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 to assign all jobs but minimize cost of assignment.
Application: assigning jobs to people

- $n$ jobs or tasks
- $m$ people
- for each job a set of people who can do that job
- for each person $j$ a limit on number of jobs $k_j$
- **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

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

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