Applications of Network Flows

Lecture 15
March 23, 2021

Most slides are courtesy Prof. Chekuri
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.

Max flow as computed by \texttt{algEdmondsKarp} or \texttt{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 $O(m^2 n)$ time, when capacities are rational (strongly polynomial time algorithm).
   4. There is an $O(mn)$ time algorithm due to Orlin which is the currently fastest strongly polynomial-time algorithm.
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**.
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.
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.
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 a flow $f$, can there by cycles in its support?
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 a flow \( f \), can there be cycles in its support?
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 a flow $f$, can there by 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) $O(mn^2)$. 

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.

Homework.
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:

1. $|\mathcal{P} \cup \mathcal{C}| \leq m$

2. for each $e \in E$, $\sum_{P \in \mathcal{P} : e \in P} f'(P) + \sum_{C \in \mathcal{C} : e \in C} f'(C) = f(e)$

3. $v(f) = \sum_{P \in \mathcal{P}} f'(P)$.

4. if $f$ is integral then so are $f'(P)$ and $f'(C)$ for all $P$ and $C$.

Proof Idea.

1. Find cyclic flows and remove them one by one – gives $f'(C)$'s.

2. Next, decompose into paths as in previous lecture.

3. Exercise: verify claims.
**Flow Decomposition**

**Lemma**

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

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

**Proof Idea.**

1. Find cyclic flows and remove them one by one – gives $f'(C)$'s.
2. Next, decompose into paths as in previous lecture.
3. Exercise: verify claims.
Find cycles one-by-one and remove.
Example

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

Compute remaining flow
Find a source to sink path, and push max flow along it (5 unites). 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
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
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 \( f : E \rightarrow \mathbb{R}_{\geq 0} \), 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:

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

Above flow decomposition can be computed in \( O(mn) \) time.

Exercise: Naive implementation of flow-decomposition takes \( O(m^2) \) time. Show how to implement in \( O(mn) \) time.
Flow decomposition into paths and cycles

Consider an integral flow network $G$, and two maximum flows $f$ and $g$ in $G$. Assume both $f$ and $g$ are acyclic. Let $P_f$ and $P_g$ be the decomposition of the two flows into paths. Then:

(A) $P_f = P_g$ (paths are the same).
(B) $|P_f| = |P_g|$ (i.e., number of paths is the same).
(C) $|P_f| + |P_g| = m$.
(D) $|P_f| \times |P_g| = nm$.
(E) None of the above.
Let \( f \) be an \( s-t \) flow in a directed network \( G = (V, E) \), and let \( A \subset V \) with \( s \in A \). The value of the flow going across cut \((A, V \setminus A)\) is

\[
\sum_{e \in \delta_{out}(A)} f(e) - \sum_{e \in \delta_{in}(A)} f(e)
\]

This is same as:

1. 0
2. \( \nu(f)/|A| \)
3. \( \nu(f) \)
4. None of the above.
Part I

Network Flow Applications I
A set of paths is **edge disjoint** if no two paths share an edge.
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.
Menger’s Theorem

Theorem

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.

(Homework) 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**

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.**

(Homework) Maxflow-mincut theorem and integrality of flow.
Theorem

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.

(Homework) 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.
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:

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

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!
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

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.*
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
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 \( \infty \).
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 $t_j$. 
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 $t_j$. 
Input: Given a (undirected) graph $G = (V, E)$.
Goal: Find a matching of maximum cardinality.
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
Bipartite Matching

Problem (Bipartite matching)

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

![Diagram of the flow-network construction](image)
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.

![Diagram of flow-network]
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 $t$. 
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 $k$. 
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 $G'$:

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 $1 \leq i \leq k$
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 $G'$:

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.
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 \( G' \):

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).
Proposition

If $G'$ has a flow of value $k$ then $G$ has a matching of size $k$. 
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.
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. \( |M| \) is \( k \) edges because \( \text{val}(f) \) 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. Paths with largest bottleneck + Ford-Fulkerson: Running time is $O(m^2 \log C) \leq O(m^2 \log n)$.  

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. Paths with largest bottleneck + Ford-Fulkerson: Running time is $O(m^2 \log C) \leq O(m^2 \log n)$.

Better running time is known: $O(m \sqrt{n})$. 
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 $X \subseteq L$, $\vert N(X) \vert \geq \vert X \vert$, 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 $\vert N(X) \vert \geq \vert X \vert$. 

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

We proved one direction (the necessary condition). For the other direction we will show the following:

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

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 $X \subseteq L$, $|N(X)| \geq |X|$.

We proved one direction (the necessary condition). For the other direction we will show the following:

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

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 $Y = A \cap R$. 

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

Because there are...

1. $|L| - |X|$ edges from $s$ to $L \cap B$.
2. $|Y|$ edges from $Y$ to $t$.
3. there are at least $|N(X) \setminus Y|$ edges from $X$ to vertices on the right side that are not in $Y$. 
By the above, cut capacity is at least
\[
\alpha = (|L| - |X|) + |Y| + |N(X) \setminus Y|.
\]
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.)
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
   \[ |N(X) \setminus Y| \geq |N(X)| - |Y| \geq |X| - |Y|. \]

By the above, cut capacity is at least
\[ \alpha = (|L| - |X|) + |Y| + |N(X) \setminus Y|. \]
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
\[ \alpha = (|L| - |X|) + |Y| + |N(X) \setminus Y| \\geq |L| - |X| + |Y| + |X| - |Y| \geq |L| = n. \]

5. Any \(s-t\) cut capacity is at least \(n \iff \) max flow at least \(n\) units \(\iff\) perfect matching. QED
Hall’s Theorem: Generalization

**Theorem (Frobenius-Hall)**

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

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
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 was until very recently $O(m\sqrt{n})$ due to Micali and Vazirani (1980). Now there is another algorithm that runs in $\tilde{O}(m^{10/7})$-time due to Madry (2015).