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:
   - Using Ford-Fulkerson algorithm in $O(mC)$ time when capacities are integral and $C$ is an upper bound on the flow.
   - Using variant of algorithm, in $O(m^2 \log C)$ time, when capacities are integral. (Polynomial time.)
   - Using Edmonds-Karp algorithm, in $O(m^2 n)$ 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 $O(m + n)$ 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.
**Exercise:** verify claims.

**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.

---

**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:

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

**Proof Idea.**

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

---

**Example**

![Example Diagram]

---

**Example**

![Example Diagram]
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. \( v(f) = \sum_{P \in P} f'(P) \)

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

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 $P_1, P_2, \ldots, P_k$ 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 $P_1, P_2, \ldots, P_k$; otherwise set $f(e) = 0$. This defines a flow of value $k$.

**Running Time**

**Theorem**

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

**Proof.**

- Set capacities of edges in $G$ to 1.
- Run Ford-Fulkerson algorithm.
- Maximum value of flow is $n$ and hence run-time is $O(nm)$.
- 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.
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 $f^{\text{in}}(v) = f^{\text{out}}(v)$.

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

Reduction to Single-Source Single-Sink

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:

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

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 \( G \) 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).

---

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.
   - \( M \) 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 \( G' \) = 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: \( O(m\sqrt{n}) \).
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?

- Clearly $|L| = |R|
- 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 $|N(X)| \geq |X|$.

---

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 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 $Y$.

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.

Sariel, Alexandra (UIUC)  CS473  33  Spring 2013  33 / 49

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.
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**

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** and goal is assign all jobs but minimize cost of assignment.

---

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√n)**.

---

Reduction to Maximum Flow

- Create directed graph **G = (V, E)** as follows
  - **V** = **{s, t}** ∪ **L** ∪ **R**: **L** set of **n** jobs, **R** set of **m** people
  - add edges (**s, i**) for each job **i** ∈ **L**, capacity **1**
  - add edges (**j, t**) for each person **j** ∈ **R**, capacity **k**
  - if job **i** can be done by person **j** ∈ **R** add an edge (**i, j**), capacity **1**

Compute max **s-t** flow. There is an assignment if and only if flow value is **n**.

---