Chapter 18

Applications of Network Flows

CS 473: Fundamental Algorithms, Fall 2011
November 1, 2011

18.1 Important Properties of Flows

18.1.0.1 Network Flow: Facts to Remember

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

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

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

(C) Given a flow of value $v$, can decompose into $O(m+n)$ flow paths of same total value $v$.

(D) Integral flow implies integral flow on paths.

(D) Maximum flow is equal to the minimum cut and minimum cut can be found in $O(m+n)$ time given any maximum flow.

18.1.0.2 Paths, Cycles and Acyclicity of Flows

Definition 18.1.1 Given a flow network $G = (V,E)$ and a flow $f : E \to \mathbb{R}^\geq$ 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 be cycles in its support?
18.1.0.3 Acyclicity of Flows

**Proposition 18.1.2** 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:**
(A) $E' = \{e \in E \mid f(e) > 0\}$, support of $f$.
(B) Suppose there is a directed cycle $C$ in $E'$
(C) Let $e'$ be the edge in $C$ with least amount of flow
(D) For each $e \in C$, reduce flow by $f(e')$. Remains a flow. Why?
(E) flow on $e'$ is reduced to 0
(F) Claim: Flow value from $s$ to $t$ does not change. Why?
(G) Iterate until no cycles

18.1.0.4 Example
18.1.0.5 Flow Decomposition

**Lemma 18.1.3** 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:

(A) $|\mathcal{P} \cup \mathcal{C}| \leq m$

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

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

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

**Proof:** [Proof Idea]

(A) Remove all cycles as in previous proposition.

(B) Next, decompose into paths as in previous lecture.

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

---

Throw away edge with no flow on it
Find a cycle in the support/flow
Reduce flow on cycle as much as possible
Throw away edge with no flow on it
Viola!!!
(C) Exercise: verify claims.

18.1.0.6 Example

Find cycles as shown before
Find a source to sink path, and push max flow along it (5 units)
Compute remaining flow
Find a source to sink path, and push max flow along it (5 units)
Definition 18.3.1

unites). Edges with 0 flow on them can not be used as they are no longer in the support of the flow. Compute remaining flow: Find a source to sink path, and push max flow along it (10 unites). Compute remaining flow: Find a source to sink path, and push max flow along it (5 unites). 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.

18.1.0.7 Flow Decomposition

Lemma 18.1.4 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:

(A) $|\mathcal{P} \cup \mathcal{C}| \leq m$

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

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

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

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

18.2 Network Flow Applications I

18.3 Edge Disjoint Paths

18.3.1 Directed Graphs

18.3.1.1 Edge-Disjoint Paths in Directed Graphs

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.

18.3.2 Reduction to Max-Flow

18.3.2.1 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.

### 18.3.2.2 Correctness of Reduction

**Lemma 18.3.2** 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 $k$.

### 18.3.2.3 Correctness of Reduction

**Lemma 18.3.3** If $G$ has a flow of value $k$ then there are $k$ edge disjoint paths between $s$ and $t$.

**Proof**: 
(A) Capacities are all 1 and hence there is integer flow of value $k$, that is $f(e) = 0$ or $f(e) = 1$ for each $e$.

(B) Decompose flow into paths of same value

(C) Flow on each path is either 1 or 0

(D) Hence there are $k$ paths $P_1, P_2, \ldots, P_k$ with flow of 1 each

(E) Paths are edge-disjoint since capacities are 1.

### 18.3.2.4 Running Time

**Theorem 18.3.4** 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 $O(nm)$.

### 18.3.3 Menger’s Theorem

#### 18.3.3.1 Menger’s Theorem

**Theorem 18.3.5 (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.
18.3.4 Undirected Graphs

18.3.4.1 Edge Disjoint Paths in Undirected Graphs

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

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

18.4 Multiple Sources and Sinks

18.4.0.2 Multiple Sources and Sinks

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

18.4.0.3 Multiple Sources and Sinks

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

18.4.0.4 Multiple Sources and Sinks: Formal Definition

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

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

(A) for each $e \in E$, $f(e) \leq c(e)$ and
(B) 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

### 18.4.0.5 Reduction to Single-Source Single-Sink

(A) Add a source node $s$ and a sink node $t$.
(B) Add edges $(s, s_1), (s, s_2), \ldots, (s, s_k)$.
(C) Add edges $(t_1, t), (t_2, t), \ldots, (t_\ell, t)$.
(D) Set the capacity of the new edges to be $\infty$.

### 18.4.0.6 Supplies and Demands

A further generalization:

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

### 18.5 Bipartite Matching

#### 18.5.1 Definitions

18.5.1.1 Matching

**Input** Given a (undirected) graph $G = (V, E)$
**Goal** Find a matching of maximum cardinality

(A) A matching is \( M \subseteq E \) such that at most one edge in \( M \) is incident on any vertex

### 18.5.1.2 Bipartite Matching

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

**Goal** Find a matching of maximum cardinality

Maximum matching has 4 edges

### 18.5.2 Reduction to Max-Flow

18.5.2.1 Reduction to Max-Flow

**Max-Flow Construction**

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

(A) \( V' = L \cup R \cup \{s, t\} \) where \( s \) and \( t \) are the new source and sink

(B) 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 \( t \)

(C) Capacity of every edge is 1
18.5.2.2 Correctness: Matching to Flow

Proposition 18.5.1 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 $G'$:

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

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

18.5.2.3 Correctness: Flow to Matching

Proposition 18.5.2 If $G'$ has a flow of value $k$ then $G$ has a matching of size $k$.

Proof: Consider flow $f$ of value $k$.

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

(A) $M$ has $k$ edges because value of flow is equal to the number of non-zero flow edges crossing cut $(L \cup \{s\}, R \cup \{t\})$
(B) Each vertex has at most one edge in $M$ incident upon it. Why?

18.5.2.4 Correctness of Reduction

Theorem 18.5.3 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.
18.5.2.5 Running Time

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

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

(B) Capacity scaling: Running time is $O(m^2 \log C) = O(m^2 \log n)$

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

18.5.3 Perfect Matchings

18.5.3.1 Perfect Matchings

Definition 18.5.4  A matching $M$ is said to be perfect if every vertex has one edge in $M$ incident upon it.

18.5.3.2 Characterizing Perfect Matchings

Problem
When does a bipartite graph have a perfect matching?

(A) Clearly $|L| = |R|$  

(B) Are there any necessary and sufficient conditions?

18.5.3.3 A Necessary Condition

Lemma 18.5.5 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 $|N(X)| \geq |X|$
18.5.3.4 Hall’s Theorem

Theorem 18.5.6 (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:

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

18.5.3.5 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 at least \( n \).

Let \((A, B)\) be an arbitrary \( s-t \) cut in \( G' \)
(A) let \( X = A \cap L \) and \( Y = A \cap R \)
(B) cut capacity is at least \( (|L| - |X|) + |Y| + |N(X) \setminus Y| \)

Because there are...
(A) \( |L| - |X| \) edges from \( s \) to \( L \cap B \).
(B) \( |Y| \) edges from \( Y \) to \( t \).
(C) there are at least \( |N(X) \setminus Y| \) edges from \( X \) to vertices on the right side that are not in \( Y \).

18.5.4 Proof of Sufficiency

18.5.4.1 Continued...

(A) By the above, cut capacity is at least \( \alpha = (|L| - |X|) + |Y| + |N(X) \setminus Y| \).
(B) \( |N(X) \setminus Y| \geq |N(X)| - |Y| \).
(This holds for any two sets.)
(C) By assumption \( |N(X)| \geq |X| \) and hence \( |N(X) \setminus Y| \geq |N(X)| - |Y| \geq |X| - |Y| \).
(D) Cut capacity is therefore at least
\[
\alpha = (|L| - |X|) + |Y| + |N(X) \setminus Y| \\
\geq |L| - |X| + |Y| + |X| - |Y| \geq |L| = n.
\]

(E) Any \( s-t \) cut capacity is at least \( n \) \( \implies \) max flow at least \( n \) units \( \implies \) perfect matching.

QED
18.5.4.2 Application: assigning jobs to people

(A) $n$ jobs or tasks
(B) $m$ people
(C) for each job a set of people who can do that job
(D) for each person $j$ a limit on number of jobs $k_j$
(E) **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.

18.5.4.3 Reduction to Maximum Flow

(A) Create directed graph $G = (V, E)$ as follows
   (A) $V = \{s, t\} \cup L \cup R$: $L$ set of $n$ jobs, $R$ set of $m$ people
   (B) add edges $(s, i)$ for each job $i \in L$, capacity 1
   (C) add edges $(j, t)$ for each person $j \in R$, capacity $k_j$
   (D) if job $i$ can be done by person $j$ add an edge $(i, j)$, capacity 1

(B) Compute max $s$-$t$ flow. There is an assignment if and only if flow value is $n$.

18.5.4.4 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 $O(m\sqrt{n})$. 