Lecture 16: Ford-Fulkerson
Announcements

See EdStem Announcement/Email posted/sent yesterday.

Midterm next **Wednesday, November 8, 6:00 – 7:30 pm in this room**

- Exam designed for a regular class time-slot but this includes extra time to finish.
- Coverage:
  - Up to the end of last Thursday’s section on Dynamic Programming

- Sample midterm for practice problems and length coming later today.
  - Will include “reference sheet” available to you on the midterm.

- Tomorrow’s section will focus on review problems.

- Zoom review session for Q&A on Tuesday Nov 7 at 4:30 pm.
Last time: Flow Network

Flow network:

- Abstraction for material \textit{flowing} through the edges.
- \( G = (V, E) \) directed graph, no parallel edges.
- Two distinguished nodes: \( s = \text{source}, \ t = \text{sink} \).
- \( c(e) = \text{capacity of edge } e \geq 0 \).
Minimum s-t cut problem:

**Given:** a flow network

**Find:** an \( s-t \) cut \((A, B)\) of minimum capacity

\[
c(A, B) = \sum_{e \text{ out of } A} c(e)
\]
Last time: Flows

Defn: An \textbf{s-t flow} in a flow network is a function \( f : E \to \mathbb{R} \) that satisfies:

- For each \( e \in E: 0 \leq f(e) \leq c(e) \) \hspace{1cm} \text{[capacity constraints]}

- For each \( v \in V - \{s, t\}: \sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e) \) \hspace{1cm} \text{[flow conservation]}

Defn: The \textbf{value} of flow \( f \),
\[
\nu(f) = \sum_{e \text{ out of } s} f(e)
\]

Only show non-zero values of \( f \)

value = 24
Given: a flow network
Find: an $s$-$t$ flow of maximum value

Last time: Maximum Flow Problem

value = 28
Corollary: Let $f$ be any $s\!-\!t$ flow and $(A, B)$ be any $s\!-\!t$ cut.

If $\nu(f) = c(A, B)$ then $f$ is a max flow and $(A, B)$ is a min cut.

Value of flow = 28

Capacity of cut = 28

Both are optimal!
Last time: Towards a Max Flow Algorithm

What about the following greedy algorithm?

- Start with $f(e) = 0$ for all edges $e \in E$.
- While there is an $s$-$t$ path $P$ where each edge has $f(e) < c(e)$.
  - “Augment” flow along $P$; that is:
    - Let $\alpha = \min_{e \in P} (c(e) - f(e))$
    - Add $\alpha$ to flow on every edge $e$ along path $P$. (Adds $\alpha$ to $v(f)$.)

But this can get stuck...
Flows and cuts so far

Let $f$ be any $s$-$t$ flow and $(A, B)$ be any $s$-$t$ cut:

**Flow Value Lemma:** The net value of the flow sent across $(A, B)$ equals $v(f)$.

$$v(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ into } A} f(e)$$

**Weak Duality:** The value of the flow is at most the capacity of the cut; i.e., $v(f) \leq c(A, B)$. “Maxflow $\leq$ Mincut”

**Corollary:** If $v(f) = c(A, B)$ then $f$ is a maximum flow and $(A, B)$ is a minimum cut.

Augmenting along paths using a greedy algorithm can get stuck.

**Today:** Ford-Fulkerson Algorithm, which applies greedy ideas to a “residual graph” that lets us reverse prior flow decisions from the basic greedy approach.
Suppose that we took this flow $f$ as a baseline, what changes could each edge handle?

- We could add up to 10 units along $sv$ or $ut$ or $uv$
- We could reduce by up to 20 units from $su$ or $uv$ or $vt$

This gives us a residual graph $G_f$ of possible changes where we draw reducing as “sending back”.

The only way we could route more flow from $s$ to $t$ would be to reduce the flow from $u$ to $v$ to make room for that amount of extra flow from $s$ to $v$. But to conserve flow we also would need to increase the flow from $u$ to $t$ by that same amount.
Greed Revisited: Residual Graph & Augmenting Paths

Residual graph $G_f$

Augment flow along path

Path in $G_f$
Greed Revisited: Residual Graph & Augmenting Paths

New residual graph $G_f$

No $s-t$ path

BTW: Flow is optimal
Residual Graphs

Original edge: $e = (u, v) \in E$.
- Flow $f(e)$, capacity $c(e)$.

Residual edges of two kinds:
- Forward: $e = (u, v)$ with capacity $c_f(e) = c(e) - f(e)$
  - Amount of extra flow we can add along $e$
- Backward: $e^R = (v, u)$ with capacity $c_f(e) = f(e)$
  - Amount we can reduce/undo flow along $e$

Residual graph: $G_f = (V, E_f)$.
- Residual edges with residual capacity $c_f(e) > 0$.
- $E_f = \{e : f(e) < c(e)\} \cup \{e^R : f(e) > 0\}$.
Residual Graphs and Augmenting Paths

Residual edges of two kinds:

- **Forward**: $e = (u, v)$ with capacity $c_f(e) = c(e) - f(e)$
  - Amount of extra flow we can add along $e$
- **Backward**: $e^R = (v, u)$ with capacity $c_f(e) = f(e)$
  - Amount we can reduce/undo flow along $e$

Residual graph: $G_f = (V, E_f)$.

- Residual edges with residual capacity $c_f(e) > 0$.
- $E_f = \{e : f(e) < c(e)\} \cup \{e^R : f(e) > 0\}$.

Augmenting Path: Any $s$-$t$ path $P$ in $G_f$. Let $\text{bottleneck}(P) = \min_{e \in P} c_f(e)$.

Ford-Fulkerson idea: Repeat “find an augmenting path $P$ and increase flow by $\text{bottleneck}(P)$” until none left.
Ford-Fulkerson Algorithm

$G:$

![Graph Diagram]
Ford-Fulkerson Algorithm

$G$:

$G_f$:

Flow value = 0

residual capacity

0 flows not shown
Ford-Fulkerson Algorithm

\[ G: \]

\[ G_f: \]

Flow value = 0
+8=8
Ford-Fulkerson Algorithm

$G$:  

$G_f$:  

Flow value = 8
Ford-Fulkerson Algorithm

$G$:  
Flow value = $8 + 2 = 10$

$G_f$:  
Flow value = $8$
Ford-Fulkerson Algorithm

$G$:  

$G_f$:  

Flow value = 10
Ford-Fulkerson Algorithm

$G$:  

$G_f$:  

Flow value = 10 + 6 = 16
Ford-Fulkerson Algorithm

$G$:

$G_f$:

Flow value = 16
Ford-Fulkerson Algorithm

\(G:\)

\(G_f:\)

Flow value = 16
+ 2 = 18
Ford-Fulkerson Algorithm

$G$:

$G_f$:

Flow value = 18
Ford-Fulkerson Algorithm

\[ G: \]

\[ G_f: \]

Flow value = 18
+1=19
Ford-Fulkerson Algorithm

$G$:

$G_f$:

Flow value = 19
Ford-Fulkerson Algorithm

$G$:

$G_f$:

Flow value = 19

Cut capacity = 19
Augmenting Path Algorithm

Ford-Fulkerson($G, s, t, c$) {
    foreach $e \in E$  $f(e) \leftarrow 0$
    $G_f \leftarrow$ residual graph

    while ($G_f$ has an $s$–$t$ path $P$) {
        $f \leftarrow$ Augment($f, c, P$)
        update $G_f$
    }
    return $f$
}

Augment($f, c, P$) {
    $b \leftarrow$ bottleneck($P$)
    foreach $e \in P$ {
        if ($e \in E$)  $f(e) \leftarrow f(e) + b$
        else  $f(e^R) \leftarrow f(e^R) - b$
    }
    return $f$
}
Max-Flow Min-Cut Theorem

Augmenting Path Theorem: Flow $f$ is a max flow $\iff$ there are no augmenting paths wrt $f$


Proof: We prove both together by showing that all of these are equivalent:

(i) There is a cut $(A, B)$ such that $v(f) = c(A, B)$.

(ii) Flow $f$ is a max flow.

(iii) There is no augmenting path w.r.t. $f$.

(i) $\Rightarrow$ (ii): We already know this by the corollary to weak duality lemma.

(ii) $\Rightarrow$ (iii): (by contradiction)

If there is an augmenting path w.r.t. flow $f$ then we can improve $f$. Therefore $f$ is not a max flow.
Proof of Max-Flow Min-Cut Theorem

(iii) ⇒ (i):

Claim: If there is no augmenting path w.r.t. $f$, there is a cut $(A, B)$ s.t. $v(f) = c(A, B)$.

Proof of Claim: Let $f$ be a flow with no augmenting paths.

Let $A$ be the set of vertices reachable from $s$ in residual graph $G_f$.

- By definition of $A$, $s \in A$.
- Since no augmenting path ($s$-$t$ path in $G_f$), $t \not\in A$.

Then $v(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ into } A} f(e)$

original network
Proof of Max-Flow Min-Cut Theorem

(iii) ⇒ (i):

Claim: If there is no augmenting path w.r.t. \( f \), there is a cut \((A, B)\) s.t. \( v(f) = c(A, B) \).

Proof of Claim: Let \( f \) be a flow with no augmenting paths.

Let \( A \) be the set of vertices reachable from \( s \) in residual graph \( G_f \).

- By definition of \( A \), \( s \in A \).
- Since no augmenting path (\( s-t \) path in \( G_f \)), \( t \notin A \).

Then
\[
v(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ into } A} f(e)
= \sum_{e \text{ out of } A} f(e)
\]

No flow
\[0 = c_f(e^R) = f(e)\]
Proof of Max-Flow Min-Cut Theorem

(iii) ⇒ (i):

Claim: If there is no augmenting path w.r.t. \( f \), there is a cut \((A, B)\) s.t. \( v(f) = c(A, B) \).

Proof of Claim: Let \( f \) be a flow with no augmenting paths.

Let \( A \) be the set of vertices reachable from \( s \) in residual graph \( G_f \).

- By definition of \( A \), \( s \in A \).
- Since no augmenting path (\( s-t \) path in \( G_f \), \( t \notin A \).

Then

\[
v(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ into } A} f(e)
\]

\[
= \sum_{e \text{ out of } A} f(e)
\]

\[
= \sum_{e \text{ out of } A} c(e)
\]

\[0 = c_f(e) = c(e) - f(e)\]  

“Saturated”
Proof of Max-Flow Min-Cut Theorem

(iii) ⇒ (i):

Claim: If there is no augmenting path w.r.t. \( f \), there is a cut \((A, B)\) s.t. \( v(f) = c(A, B) \).

Proof of Claim: Let \( f \) be a flow with no augmenting paths.

- Let \( A \) be the set of vertices reachable from \( s \) in residual graph \( G_f \).
  - By definition of \( A \), \( s \in A \).
  - Since no augmenting path (\( s-t \) path in \( G_f \)), \( t \notin A \).

Then
\[
v(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ into } A} f(e)
\]
\[
= \sum_{e \text{ out of } A} f(e)
\]
\[
= \sum_{e \text{ out of } A} c(e) = c(A, B)
\]

original network
Running Time

- Computing first \( G_f \) takes \( O(n + m) \) time. (Can ignore disconnected bits so \( m \geq n - 1 \).)
- Finding each augmenting path (graph search in \( G_f \)) takes \( O(m) \) time.
- Updating \( f \) and \( G_f \) takes \( O(n) \) time.

Total \( O(m) \) per iteration.

**Assumption:** All capacities are integers between \( 1 \) and \( C \).

**Ford-Fulkerson Invariant:** Every flow value \( f(e) \) and every residual capacity \( c_f(e) \) remains an integer throughout the algorithm. So there is a maximum flow with only integer flows.

**Theorem:** The Ford-Fulkerson algorithm terminates in \( \leq \text{Maxflow} < nC \) iterations.

**Proof:** Capacity of cut with \( A = \{s\} \) is \( \leq (n - 1)C \). Each augmentation increases flow value by at least \( 1 \).

**Corollary:** If \( C = 1 \), Ford-Fulkerson runs in \( O(mn) \) time.
Bipartite Matching

A graph $G = (V, E)$ is bipartite iff

- Set $V$ of vertices has two disjoint parts $X$ and $Y$
- Every edge in $E$ joins a vertex from $X$ and a vertex from $Y$

Set $M \subseteq E$ is a matching in $G$ iff no two edges in $M$ share a vertex

Goal: Find a matching $M$ in $G$ of maximum size.

Differences from stable matching

- limited set of possible partners for each vertex
- sides may not be the same size
- no notion of stability; matching everything may be impossible.
Bipartite Matching

• Models assignment problems
  • \( X \) represents customers, \( Y \) represents salespeople
  • \( X \) represents professors, \( Y \) represents courses

• If \(|X| = |Y| = n\)
  • \( G \) has perfect matching iff maximum matching has size \( n \)
Bipartite Matching

**Input:** Bipartite graph

**Goal:** Find *maximum size* matching.
Bipartite Matching as a special case of Flow

**Input:** Bipartite graph
Bipartite Matching as a special case of Flow

Add new source $s$ pointing to left set, new sink $t$ pointed to by right set.
Direct all edges from left to right with capacity 1. Compute MaxFlow.
Bipartite Matching as a special case of Flow

Add new source $s$ pointing to left set, new sink $t$ pointed to by right set.
Direct all edges from left to right with capacity 1. Compute MaxFlow.

**Correctness:**
Integer flow just gives a subset of edges.
Source and sink edges imply it is a matching

Time $O(mn)$
Bipartite Matching

**Input:** Bipartite graph

**Goal:** Find **maximum size** matching.
Bipartite Matching as a special case of Flow

Add new source \( s \) pointing to left set, new sink \( t \) pointed to by right set. Direct all edges from left to right with capacity 1. Compute MaxFlow.

**Correctness:**
Integer flow just gives a subset of edges.
Source and sink edges imply it is a matching

**Optimality**

Time \( O(mn) \)
Ford-Fulkerson Efficiency

Worst case runtime $O(mnC)$ with integer capacities $\leq C$.

- $O(m)$ time per iteration.
- At most $nC$ iterations.
- This is “pseudo-polynomial” running time.

- May take exponential time, even with integer capacities:

$c = 10^9$, say

$G_f = G$ etc.
Choosing Good Augmenting Paths
Choosing Good Augmenting Paths

Use care when selecting augmenting paths.

- Some choices lead to exponential algorithms.
- Clever choices lead to polynomial algorithms.
- If capacities are irrational, algorithm not guaranteed to terminate!

**Goal:** Choose augmenting paths so that:

- Can find augmenting paths efficiently.
- Few iterations.
- Choose augmenting paths with: [Edmonds-Karp 1972, Dinitz 1970]
  - Max bottleneck capacity.
  - Sufficiently large bottleneck capacity.
  - Fewest number of edges.
Capacity Scaling

General idea:

• Choose augmenting paths $P$ with ‘large’ capacity.

• Can augment flows along a path $P$ by any amount $\leq \text{bottleneck}(P)$
  • Ford-Fulkerson still works

• Get a flow that is maximum for the high-order bits first and then add more bits later
Capacity Scaling
Capacity Scaling Bit 1

Solve flow problem with capacities with just the high-order bit:
Capacity Scaling Bit 1

Solve flow problem with capacities with just the high-order bit:
- Each edge has “capacity” $\leq 1$ (equivalent to 4 here)
- Time $O(mn)$
Capacity Scaling Bit 1
Solve flow problem with capacities with the 2 high-order bits:

- Capacity of old min cut goes up by \( \leq 1 \) per edge (equivalent to 2 here) for a total residual capacity \( \leq m \).
Solve flow problem with capacities with the 2 high-order bits:

• Capacity of old min cut goes up by $\leq 1$ per edge (equivalent to 2 here) for a total residual capacity $\leq m$.

• Time $O(m^2)$ for $\leq m$ iterations.
Capacity Scaling Bits 1 and 2
Solve flow problem with capacities with all 3 bits:

- Capacity of old min cut goes up by \( \leq 1 \) per edge for a total residual capacity \( \leq m \).
Solve flow problem with capacities with all 3 bits:

- Capacity of old min cut goes up by \( \leq 1 \) per edge for a total residual capacity \( \leq m \).
- Time \( O(m^2) \) for \( \leq m \) iterations.
Capacity Scaling All Bits

Flow = 15
Capacity Scaling All Bits

Flow = 15
Cut Value = 15
Flow is a MaxFlow
Total time for capacity scaling

- Number of rounds = \([\log_2 C]\) where \(C\) is the largest capacity
- Time per round \(O(m^2)\)
  - At most \(m\) augmentations per round
  - \(O(m)\) time per augmentation

Total time \(O(m^2 \log C)\)