Overview

- Paths, Flows & Cuts
  - Paths
  - Flows
  - Cuts

- Ford-Fulkerson Algorithm
  - Residual Graph
  - Main Algorithm

- Acknowledgements
Paths: Measure of “Resilience”

- Given (directed) graph $G(V, E)$, we would like to know how “resilient” it may be
  - Is $G$ (strongly) connected?
  - How many edges does one need to remove to disconnect it?
  - How many vertices does one need to remove to disconnect it?

Using BFS/DFS can determine if there is $s \rightarrow t$ path
Does it have 2 edge-disjoint $s \rightarrow t$ paths?
How many edge-disjoint paths does it have?

Edge-Disjoint Paths problem:
Input: (directed) graph $G(V, E)$, $s, t \in V$
Output: Maximum number of edge-disjoint $s \rightarrow t$ paths

More generally, can consider weighted directed graphs/networks: edge weights are how much data can go through traffic system: edge weights are how much traffic can go through
Weighted version is (almost) the maximum flow problem.

How to generalize notion of edge-disjoint in weighted version?
Paths: Measure of “Resilience”

- Given (directed) graph $G(V, E)$, we would like to know how “resilient” it may be
  - Is $G$ (strongly) connected?
  - How many edges does one need to remove to disconnect it?
  - How many vertices does one need to remove to disconnect it?
- Using BFS/DFS can determine if there is $s \rightarrow t$ path
  - Does it have 2 edge-disjoint $s \rightarrow t$ paths?
  - How many edge-disjoint paths does it have?

Edge-Disjoint Paths problem:

- Input: (directed) graph $G(V, E)$, $s, t \in V$
- Output: Maximum number of edge-disjoint $s \rightarrow t$ paths

More generally, can consider weighted directed graphs

networks: edge weights are how much data can go through
traffic system: edge weights are how much traffic can go through
Weighted version is (almost) the maximum flow problem.

How to generalize notion of edge-disjoint in weighted version?
Paths: Measure of “Resilience”

- Given (directed) graph $G(V, E)$, we would like to know how “resilient” it may be
  - Is $G$ (strongly) connected?
  - How many edges does one need to remove to disconnect it?
  - How many vertices does one need to remove to disconnect it?

- Using BFS/DFS can determine if there is $s \to t$ path
  - Does it have 2 edge-disjoint $s \to t$ paths?
    - How many edge-disjoint paths does it have?

- Edge-Disjoint Paths problem:
  - **Input:** (directed) graph $G(V, E)$, $s, t \in V$
  - **Output:** Maximum number of edge-disjoint $s \to t$ paths
Paths: Measure of “Resilience”

- Given (directed) graph $G(V, E)$, we would like to know how “resilient” it may be
  - Is $G$ (strongly) connected?
  - How many edges does one need to remove to disconnect it?
  - How many vertices does one need to remove to disconnect it?

- Using BFS/DFS can determine if there is $s \to t$ path
  - Does it have 2 edge-disjoint $s \to t$ paths?
  - How many edge-disjoint paths does it have?

- Edge-Disjoint Paths problem:
  - **Input:** (directed) graph $G(V, E)$, $s, t \in V$
  - **Output:** Maximum number of edge-disjoint $s \to t$ paths

- More generally, can consider weighted directed graphs
  - networks: edge weights are how much data can go through
  - traffic system: edge weights are how much traffic can go through
  
  Weighted version is (almost) the *maximum flow* problem.
Paths: Measure of “Resilience”

- Given (directed) graph $G(V, E)$, we would like to know how “resilient” it may be
  - Is $G$ (strongly) connected?
  - How many edges does one need to remove to disconnect it?
  - How many vertices does one need to remove to disconnect it?

- Using BFS/DFS can determine if there is $s \rightarrow t$ path
  - Does it have 2 edge-disjoint $s \rightarrow t$ paths?
  - How many edge-disjoint paths does it have?

- Edge-Disjoint Paths problem:
  - **Input:** (directed) graph $G(V, E)$, $s, t \in V$
  - **Output:** Maximum number of edge-disjoint $s \rightarrow t$ paths

- More generally, can consider weighted directed graphs
  - networks: edge weights are how much data can go through
  - traffic system: edge weights are how much traffic can go through
  - Weighted version is (almost) the *maximum flow* problem.

- How to generalize notion of edge-disjoint in weighted version?
Flows

- We will now think of a weighted graph $G(V, E, c)$, where $c : E \rightarrow \mathbb{R}_{>0}$ (the weight function) is giving the capacity of an edge.

If we have $c : E \rightarrow \mathbb{N}$ then
  - Think of capacity as number of lanes in a street/highway
  - Or think of $G(V, E, c)$ as unweighted graph with $c((u, v))$ being the number of distinct $u \rightarrow v$ edges
Flows

- We will now think of a weighted graph $G(V, E, c)$, where $c : E \rightarrow \mathbb{R}_{>0}$ (the weight function) is giving the capacity of an edge.
- An $s \rightarrow t$ flow is a function $f : E \rightarrow \mathbb{R}_{\geq 0}$ with the properties:
  1. **Capacity constraints:** $0 \leq f(e) \leq c(e)$ for all $e \in E$
  2. **Flow conservation:** $f_{in}(u) = f_{out}(u)$ for each $u \in V \setminus \{s, t\}$, where

\[
  f_{in}(u) := \sum_{w \in N_{in}(u)} f(w, u), \quad \text{and} \quad f_{out}(u) := \sum_{w \in N_{out}(u)} f(u, w)
\]
Flows

- We will now think of a weighted graph $G(V, E, c)$, where $c : E \rightarrow \mathbb{R}_{>0}$ (the weight function) is giving the capacity of an edge.

- An $s \rightarrow t$ flow is a function $f : E \rightarrow \mathbb{R}_{\geq 0}$ with the properties:
  1. Capacity constraints: $0 \leq f(e) \leq c(e)$ for all $e \in E$
  2. Flow conservation: $f_{\text{in}}(u) = f_{\text{out}}(u)$ for each $u \in V \setminus \{s, t\}$, where

\[
 f_{\text{in}}(u) := \sum_{w \in N_{\text{in}}(u)} f(w, u), \quad \text{and} \quad f_{\text{out}}(u) := \sum_{w \in N_{\text{out}}(u)} f(u, w)
\]

- The value of a flow $f$ is $\text{value}(f) := f_{\text{out}}(s) - f_{\text{in}}(s)$

  In this course, we will generally have $f_{\text{in}}(s) = 0$, so $\text{value}(f) = f_{\text{out}}(s)$
Flows

- We will now think of a weighted graph $G(V, E, c)$, where $c : E \rightarrow \mathbb{R}_{>0}$ (the weight function) is giving the capacity of an edge.
- An $s \rightarrow t$ flow is a function $f : E \rightarrow \mathbb{R}_{\geq 0}$ with the properties:
  1. **Capacity constraints**: $0 \leq f(e) \leq c(e)$ for all $e \in E$
  2. **Flow conservation**: $f_{\text{in}}(u) = f_{\text{out}}(u)$ for each $u \in V \setminus \{s, t\}$, where

$$
    f_{\text{in}}(u) := \sum_{w \in N_{\text{in}}(u)} f(w, u), \quad \text{and} \quad f_{\text{out}}(u) := \sum_{w \in N_{\text{out}}(u)} f(u, w)
$$

- The value of a flow $f$ is $\text{value}(f) := f_{\text{out}}(s) - f_{\text{in}}(s)$

  In this course, we will generally have $f_{\text{in}}(s) = 0$, so $\text{value}(f) = f_{\text{out}}(s)$

- **Max-flow problem**:
  - Input: directed graph $G(V, E, c)$, with $c : E \rightarrow \mathbb{R}_{>0}$, vertices $s, t \in V$
  - Output: an $s \rightarrow t$ flow with maximum value.
Figure 10.1. Harris and Ross’s map of the Warsaw Pact rail network. (See Image Credits at the end of the book.)
How does the idea of flows generalize edge-disjoint paths?

- Think of capacity as number of lanes in a street/highway
- Or think of $G(V, E, c)$ as unweighted graph with $c((u, v))$ being the number of distinct $u \rightarrow v$ edges
Flows & Paths

- How does the idea of flows generalize edge-disjoint paths?
- If $G(V, E, c)$, where $c : E \to \mathbb{N}$
  - Think of capacity as number of lanes in a street/highway
  - Or think of $G(V, E, c)$ as unweighted graph with $c((u, v))$ being the number of distinct $u \to v$ edges
Flows & Paths

- How does the idea of flows generalize edge-disjoint paths?
- If $G(V, E, c)$, where $c : E \rightarrow \mathbb{N}$
  - Think of capacity as number of lanes in a street/highway
  - Or think of $G(V, E, c)$ as unweighted graph with $c((u, v))$ being the number of distinct $u \rightarrow v$ edges

Integral flow $f$ (i.e. $f : E \rightarrow \mathbb{N}$) with value($f$) = $k$ corresponds to $k$ edge-disjoint paths in the unweighted graph $G(V, E, c)$ above
  - Think of edge $e$ with $f(e) = h$ as the collections of paths using $h$ lanes in highway
  - flow conservation $\leftrightarrow$ # cars entering vertex $u = $ # cars leaving vertex $u$
  - capacity constraints $\leftrightarrow$ each car gets one lane in highway
Example
(also in Jeff's book)
Path decomposition lemma

Lemma (Path Decomposition Lemma)

Let $G$ be a weighted DAG with integral weights. Let $f$ be an integral $s \to t$ flow, with $f_{\text{in}}(s) = 0$ and $\text{value}(f) = k$. Then, there are $s \to t$ paths $P_1, \ldots, P_k$ such that each edge $e$ appears in $f(e)$ of these paths.

Remark: for full “flow decomposition theorem” see Jeff Erickson’s book, chapter 10.
Flows and Cuts

- How can we upper bound the maximum possible value of a flow?
- How do we know a given flow is the maximum flow?

Trivial upper bound: total capacity of all edges

Better upper bound: total capacity of edges leaving $s$

Can we do better? YES! Let’s look at all $s-t$ cuts!

$(s-t)$ cut is a cut $(S, V \setminus S)$ such that $s \in S$ and $t \not\in S$.

Capacity of cut: $C_{\text{out}}(S) := \sum_{e \in \delta_{\text{out}}(S)} c(e)$

where $\delta_{\text{out}}(S) = \{(u, v) \in E | u \in S, v \not\in S\}$.

By path decomposition lemma or flow conservation, can prove that $\text{value}(f) \leq C_{\text{out}}(S)$ for any flow $f$ and cut $S$. 
Flows and Cuts

- How can we upper bound the maximum possible value of a flow?
- How do we know a given flow is the maximum flow?
- **Trivial** upper bound: total capacity of all edges

By path decomposition lemma or flow conservation, we can prove that value of flow $f$ is less than or equal to the capacity of the cut $S$, i.e., $\text{value}(f) \leq C_{out}(S)$ for any flow $f$ and cut $S$. 

A cut $(S, V \setminus S)$ is a cut such that $s \in S$ and $t \not\in S$. The capacity of the cut is defined as $C_{out}(S) := \sum_{e \in \delta_{out}(S)} c(e)$, where $\delta_{out}(S) = \{(u, v) \in E | u \in S, v \not\in S\}$. 

Both the trivial upper bound and the cut capacity upper bound generalize better than previous bounds.
Flows and Cuts

- How can we upper bound the maximum possible value of a flow?
- How do we know a given flow is the maximum flow?
- **Trivial** upper bound: total capacity of all edges
- **Better** upper bound: total capacity of edges leaving \( s \)

**Definition:** A \( s - t \) cut is a cut \((S, V \setminus S)\) such that \( s \in S \) and \( t \notin S \).

**Capacity of cut:**
\[
C_{\text{out}}(S) := \sum_{e \in \delta_{\text{out}}(S)} c(e)
\]

By path decomposition lemma or flow conservation, can prove that
\[
\text{value}(f) \leq C_{\text{out}}(S)
\]
for any flow \( f \) and cut \( S \).
Flows and Cuts

- How can we upper bound the maximum possible value of a flow?
- How do we know a given flow is the maximum flow?
- **Trivial** upper bound: total capacity of all edges
- **Better** upper bound: total capacity of edges *leaving* $s$
- Can we do better?

**YES!** Let’s look at all $s - t$ cuts!
(generalizes better upper bound)
Flows and Cuts

- How can we upper bound the maximum possible value of a flow?
- How do we know a given flow is the maximum flow?
- **Trivial** upper bound: total capacity of all edges
- **Better** upper bound: total capacity of edges leaving $s$
- Can we do better?

YES! Let’s look at all $s - t$ cuts!

(generalizes better upper bound)

- an $s - t$ cut is a cut $(S, V \setminus S)$ such that $s \in S$ and $t \not\in S$.
- Capacity of cut:
  \[
  C_{out}(S) := \sum_{e \in \delta_{out}(S)} c(e)
  \]
  where $\delta_{out}(S) = \{(u, v) \in E \mid u \in S, v \not\in S\}$
Flows and Cuts

- How can we upper bound the maximum possible value of a flow?
- How do we know a given flow is the maximum flow?
- **Trivial** upper bound: total capacity of all edges
- **Better** upper bound: total capacity of edges *leaving* $s$
- Can we do better?

  YES! Let’s look at all $s - t$ cuts!
  (generalizes better upper bound)

- an $s - t$ cut is a cut $(S, V \setminus S)$ such that $s \in S$ and $t \notin S$.
  - Capacity of cut:
    \[
    C_{\text{out}}(S) := \sum_{e \in \delta_{\text{out}}(S)} c(e)
    \]
    where $\delta_{\text{out}}(S) = \{(u, v) \in E \mid u \in S, v \notin S\}$
  - By path decomposition lemma or flow conservation, can prove that
    \[
    \text{value}(f) \leq C_{\text{out}}(S)
    \]
    for any flow $f$ and cut $S$. 

Example

Note that s-t cut \{s, a, b, t\} gives better upper bound than \{s\}.
Max-Flow Min-Cut Theorem

- Capacity of cuts are an upper bound for flows.
  Is this a tight upper bound?

**Theorem (Max-Flow Min-Cut Theorem)**

The value of the maximum $s \rightarrow t$ flow equals the minimum capacity among all cuts.

$$\max_{f \text{ s-t flow}} \text{value}(f) = \min_{S \text{ is s-t cut}} C_{out}(S)$$
Max-Flow Min-Cut Theorem

- Capacity of cuts are an upper bound for flows.
  Is this a tight upper bound?

Theorem (Max-Flow Min-Cut Theorem)

The value of the maximum $s - t$ flow equals the minimum capacity among all cuts.

$$\max_{f\text{ }s-t\text{ flow}} \text{value}(f) = \min_{S\text{ }is\text{ }s-t\text{ cut}} C_{\text{out}}(S)$$

- We will give an algorithmic proof of this theorem, that solves the max-flow and the min-cut problem at the same time.
Paths, Flows & Cuts
- Paths
- Flows
- Cuts

Ford-Fulkerson Algorithm
- Residual Graph
- Main Algorithm

Acknowledgements
Ford-Fulkerson Algorithm: Intuition

- Natural (greedy) strategy: by path decomposition lemma, we could just keep finding $s \rightarrow t$ paths in the graph (updating the capacities of the graph)
Ford-Fulkerson Algorithm: Intuition

- Natural (greedy) strategy: by path decomposition lemma, we could just keep finding $s \rightarrow t$ paths in the graph (updating the capacities of the graph)
- No bueno: greedy approach may force us to “commit to a path” which may “block others”
Ford-Fulkerson Algorithm: Intuition

- Natural (greedy) strategy: by path decomposition lemma, we could just keep finding $s \rightarrow t$ paths in the graph (updating the capacities of the graph)
- No bueno: greedy approach may force us to “commit to a path” which may “block others”
- Would be nice to “push back/undo” bad paths, but only if this improves our current solution
  
  Main idea behind Ford-Fulkerson.
Ford-Fulkerson Algorithm: Intuition

- Natural (greedy) strategy: by path decomposition lemma, we could just keep finding $s \to t$ paths in the graph
  
  (updating the capacities of the graph)

- No bueno: greedy approach may force us to “commit to a path” which may “block others”

- Would be nice to “push back/undo” bad paths, but only if this improves our current solution

  Main idea behind Ford-Fulkerson.

- Augment the flow by finding “augmenting path” which increases total amount of flow
Example
(bad greedy example)

all capacities 1.

Shortest path in red would block max flow in green
The residual graph is the object we will study to find augmenting paths.
Residual Graph

- The residual graph is the object we will study to find augmenting paths.

- Given $G(V, E, c)$ and $s \rightarrow t$ flow $f$ on $G$, define the residual graph $G_f$ as follows:
  - $V(G_f) = V(G)$
  - For each $(u, v) =: e \in E$ add edges
    - $(u, v)$ to $G_f$ with capacity $c(e) - f(e)$ (forward edges)
    - $(v, u)$ to $G_f$ with capacity $f(e)$ (backward edges)
Example

Graph & flow $f$

Residual graph
Augmenting Path

- An *augmenting path* with respect to a flow \( f \) is simply an \( s \to t \) path\(^1\) in \( G_f \)

---

\(^1\)By path here we mean a simple path, and not a walk.
Augmenting Path

- An *augmenting path* with respect to a flow $f$ is simply an $s \rightarrow t$ path\(^1\) in $G_f$
- Given augmenting path $P$ in $G_f$, want to push *as much flow as possible* through it:

\[
\text{bottleneck}(P, f) := \text{minimum capacity of edge of } P \text{ in } G_f
\]

---

\(^1\)By path here we mean a simple path, and not a walk.
Example

In previous residual graph, have following augmenting path.

\[ \text{bottleneck}(P_f) = 1 \]
Improving the Flow

- **Input:** flow $f$ and an augmenting path $P$ in $G_f$
- **Output:** improved flow $f'$
Improving the Flow

- **Input:** flow \(f\) and an augmenting path \(P\) in \(G_f\)
- **Output:** improved flow \(f'\)

\[
\text{augment}(f, P) :
\]
- Let \(b := \text{bottleneck}(P, f)\) and \(f'(e) = f(e)\) for all \(e \in E\)
- for each \(e := (u, v) \in P\):
  - If \(e\) forward edge:
    \[
    f'(e) = f'(e) + b
    \]
  - If \(e\) backward edge:
    \[
    f'(v, u) = f'(v, u) - b \quad \text{(decrease reversed edge)}
    \]
- **return** \(f'\)
Lemma (Flow Improvement)

Let \( f \) be a flow in \( G \) with \( f_{in}(s) = 0 \) and \( P \) an augmenting path with respect to \( f \). If \( f' \) is the output from \( \text{augment}(f, P) \), then \( f' \) is a flow with

\[
\text{value}(f') = \text{value}(f) + \text{bottleneck}(P, f)
\]

and \( f'_{in}(s) = 0 \).
Improving Flow

Lemma (Flow Improvement)

Let $f$ be a flow in $G$ with $f_{\text{in}}(s) = 0$ and $P$ an augmenting path with respect to $f$. If $f'$ is the output from $\text{augment}(f, P)$, then $f'$ is a flow with

$$\text{value}(f') = \text{value}(f) + \text{bottleneck}(P, f)$$

and $f'_{\text{in}}(s) = 0$.

- To check that $f'$ is a flow, need to check capacity constraint and flow conservation constraint.
Lemma (Flow Improvement)

Let $f$ be a flow in $G$ with $f_{\text{in}}(s) = 0$ and $P$ an augmenting path with respect to $f$. If $f'$ is the output from $\text{augment}(f, P)$, then $f'$ is a flow with

$$\text{value}(f') = \text{value}(f) + \text{bottleneck}(P, f)$$

and $f'_{\text{in}}(s) = 0$.

- Let $b := \text{bottleneck}(P, f)$.
- **Capacity constraint:** given $e \in E(G_f)$, we have
  - $e$ forward edge in $G_f$, then
    $$f'(e) = f(e) + b \leq f(e) + (c(e) - f(e)) = c(e)$$
  - $e := (u, v)$ backward edge in $G_f$, then
    $$f'(v, u) = f(v, u) - b \leq f(v, u) \leq c(v, u)$$

and

$$f'(v, u) = f(v, u) - b \geq f(v, u) - f(v, u) \geq 0$$
Lemma (Flow Improvement)

Let $f$ be a flow in $G$ with $f_{\text{in}}(s) = 0$ and $P$ an augmenting path with respect to $f$. If $f'$ is the output from $\text{augment}(f, P)$, then $f'$ is a flow with

$$\text{value}(f') = \text{value}(f) + \text{bottleneck}(P, f)$$

and $f'_{\text{in}}(s) = 0$.

- Let $b := \text{bottleneck}(P, f)$.
- **Flow Conservation:** let $u \in V$ be a vertex.
  - if $u \notin P$ then flow in and out of $u$ doesn’t change.
Improving Flow

Lemma (Flow Improvement)

Let $f$ be a flow in $G$ with $f_{\text{in}}(s) = 0$ and $P$ an augmenting path with respect to $f$. If $f'$ is the output from $\text{augment}(f, P)$, then $f'$ is a flow with

$$\text{value}(f') = \text{value}(f) + \text{bottleneck}(P, f)$$

and $f'_{\text{in}}(s) = 0$.

- Let $b := \text{bottleneck}(P, f)$.
- Flow Conservation: let $u \in V$ be a vertex.
  - if $u \in P$, have 4 cases to analyze. Let $e_1 := (w, u)$ and $e_2 := (u, z)$ be the edges in $P$ passing through $u$ in $G_f$.
    - $e_1, e_2$ forward edges: both incoming and outgoing flow $\text{increase}$ by $b$
    - $e_1, e_2$ backward edges: both incoming and outgoing flow $\text{decrease}$ by $b$
    - $e_1$ forward, $e_2$ backward: both incoming and outgoing flow $\text{unchanged}$
    - $e_1$ backward, $e_2$ forward: both incoming and outgoing flow $\text{unchanged}$
Improving Flow

**Lemma (Flow Improvement)**

Let $f$ be a flow in $G$ with $f_{\text{in}}(s) = 0$ and $P$ an augmenting path with respect to $f$. If $f'$ is the output from $\text{augment}(f, P)$, then $f'$ is a flow with

$$\text{value}(f') = \text{value}(f) + \text{bottleneck}(P, f)$$

and $f'_{\text{in}}(s) = 0$.

- Let $b := \text{bottleneck}(P, f)$.
- Value of flow $f'$ and $f'_{\text{in}}(s)$:
  - $f_{\text{in}}(s) = 0 \Rightarrow$ no backward edges incident to $s$ in $G_f$
  - $f'_{\text{in}}(s) = f_{\text{in}}(s) + 0 = f_{\text{in}}(s) = 0$
Lemma (Flow Improvement)

Let $f$ be a flow in $G$ with $f_{\text{in}}(s) = 0$ and $P$ an augmenting path with respect to $f$. If $f'$ is the output from augment($f$, $P$), then $f'$ is a flow with

$$\text{value}(f') = \text{value}(f) + \text{bottleneck}(P, f)$$

and $f'_{\text{in}}(s) = 0$.

- Let $b := \text{bottleneck}(P, f)$.
- Value of flow $f'$ and $f'_{\text{in}}(s)$:
  - Value of $f'$: by previous bullet, only forward edges out of $s$, thus:
    $$\text{value}(f') = f'_\text{out}(s) = f_\text{out}(s) + b = \text{value}(f) + b$$
Ford-Fulkerson Algorithm

Now that we know that augmenting paths can only improve our flow, we can describe Ford-Fulkerson, which simply applies the augmenting operation until we can no longer do it.

- Ford-Fulkerson($G$):
  1. Initialize $f(e) = 0$ for all $e \in E$, and initialize $G_f$ accordingly
  2. While there is $s \rightarrow t$ path $P \in G_f$:
     - $f \leftarrow \text{augment}(f, P)$
     - update $G_f$
  3. return $f$
Ford-Fulkerson Algorithm

Now that we know that augmenting paths can only improve our flow, we can describe Ford-Fulkerson, which simply applies the augmenting operation until we can no longer do it.

- **Ford-Fulkerson**($G$):
  1. Initialize $f(e) = 0$ for all $e \in E$, and initialize $G_f$ accordingly
  2. While there is $s \rightarrow t$ path $P \in G_f$:
     - $f \leftarrow \text{augment}(f, P)$
     - update $G_f$
  3. **return** $f$

Next lecture: runtime analysis and proof of correctness.
Acknowledgement

Based on

- Prof. Lau’s Lecture 15
  https://cs.uwaterloo.ca/~lapchi/cs341/notes/L15.pdf
- Jeff Erickson’s book, Chapter 10
References I

