7. Network Flow

Maximum Flow and Minimum Cut

Max flow and min cut.
- Two very rich algorithmic problems.
- Cornerstone problems in combinatorial optimization.
- Beautiful mathematical duality.

Nontrivial applications / reductions.
- Data mining.
- Open-pit mining.
- Project selection.
- Airline scheduling.
- Bipartite matching.
- Baseball elimination.
- Image segmentation.
- Network connectivity.
- Network reliability.
- Distributed computing.
- Egalitarian stable matching.
- Security of statistical data.
- Network intrusion detection.
- Multi-camera scene reconstruction.
- Many many more . . .

Minimum Cut Problem

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

Def. An $s$-$t$ cut is a partition $(A, B)$ of $V$ with $s \in A$ and $t \in B$.

Def. The capacity of a cut $(A, B)$ is: $$\text{cap}(A, B) = \sum_{e \text{ out of } A} c(e)$$

\[\begin{array}{c c c c c c c c c}
2 & 5 & 4 & 5 & 5 & 10 & 9 & 15 & 10 \\
3 & 5 & 6 & 4 & 4 & 6 & 8 & 8 & 10 \\
4 & 3 & 8 & 15 & 15 & 10 & 10 & 10 & 15 \\
5 & 9 & 4 & 10 & 10 & 10 & 30 & 15 & 4 \\
6 & 10 & 10 & 4 & 4 & 6 & 15 & 15 & 10 \\
7 & 30 & 10 & 15 & 15 & 10 & 10 & 10 & 10 \\
8 & 10 & 10 & 10 & 10 & 30 & 30 & 30 & 30 \\
9 & 15 & 10 & 15 & 15 & 10 & 10 & 10 & 10 \\
10 & 15 & 15 & 15 & 15 & 15 & 15 & 15 & 15 \\
\end{array}\]

Capacity $= 10 + 5 + 15 = 30$

Def. An $s$-$t$ cut is a partition $(A, B)$ of $V$ with $s \in A$ and $t \in B$. Def. The capacity of a cut $(A, B)$ is: $$\text{cap}(A, B) = \sum_{e \text{ out of } A} c(e)$$

\[\begin{array}{c c c c c c c c c}
2 & 5 & 4 & 5 & 5 & 10 & 9 & 15 & 10 \\
3 & 5 & 6 & 4 & 4 & 6 & 8 & 8 & 10 \\
4 & 3 & 8 & 15 & 15 & 10 & 10 & 10 & 15 \\
5 & 9 & 4 & 10 & 10 & 10 & 30 & 15 & 4 \\
6 & 10 & 10 & 4 & 4 & 6 & 15 & 15 & 10 \\
7 & 30 & 10 & 15 & 15 & 10 & 10 & 10 & 10 \\
8 & 10 & 10 & 10 & 10 & 30 & 30 & 30 & 30 \\
9 & 15 & 10 & 15 & 15 & 10 & 10 & 10 & 10 \\
10 & 15 & 15 & 15 & 15 & 15 & 15 & 15 & 15 \\
\end{array}\]

Capacity $= 9 + 15 + 8 + 30 = 62$

Min $s$-$t$ cut: find an $s$-$t$ cut of minimum capacity. Minimum Cut Problem

Minimum Cut Problem

Def. An $s$-$t$ flow is a function that satisfies:
- For each $e \in E$: $0 \leq f(e) \leq c(e)$ (capacity)
- For each $v \in V - \{s, t\}$: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ (conservation)

Def. The value of a flow $f$ is: $\nu(f) = \sum_{e \text{ out of } s} f(e)$.

Flows

Def. An $s$-$t$ flow is a function that satisfies:
- For each $e \in E$: $0 \leq f(e) \leq c(e)$ (capacity)
- For each $v \in V - \{s, t\}$: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ (conservation)

Def. The value of a flow $f$ is: $\nu(f) = \sum_{e \text{ out of } s} f(e)$.
Def. An s-t flow is a function that satisfies:
- For each $e \in E$: \( 0 \leq f(e) \leq c(e) \) (capacity)
- For each $v \in V - \{s, t\}$: \( \sum_{e \in \text{in } v} f(e) = \sum_{e \in \text{out } v} f(e) \) (conservation)

Def. The value of a flow $f$ is: \( v(f) = \sum_{e \in \text{out } s} f(e) \).

**Flow value lemma.** Let $f$ be any flow, and let $(A, B)$ be any s-t cut. Then, the net flow, $f(A, B)$ sent across the cut is equal to the amount leaving $s$.

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

**Flow value lemma.** Let $f$ be any flow, and let $(A, B)$ be any s-t cut. Then, the net flow, $f(A, B)$ sent across the cut is equal to the amount leaving $s$.

\[
\sum_{e \in \text{out } A} f(e) - \sum_{e \in \text{in } A} f(e) = v(f)
\]
Flow value lemma. Let $f$ be any flow, and let $(A, B)$ be any s-t cut. Then, the net flow, $f(A, B)$, sent across the cut is equal to the amount leaving $s$.

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

Weak duality. Let $f$ be any flow, and let $(A, B)$ be any s-t cut. Then the value of the flow is at most the capacity of the cut.

\[ v(f) \leq \text{cap}(A, B) \]

\[ v(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) \leq \sum_{e \text{ out of } A} c(e) = \text{cap}(A, B) \]
Corollary. Let f be any flow, and let (A, B) be any cut. If v(f) = cap(A, B), then f is a max flow and (A, B) is a min cut.

**Certificate of Optimality**

Value of flow = 28
Cut capacity = 28 \(\Rightarrow\) Flow value = 28

**Towards a Max Flow Algorithm**

Greedy algorithm.
- Start with \(f(e) = 0\) for all edge \(e \in E\).
- Find an s-t path \(P\) where each edge has \(f(e) < c(e)\).
- Augment flow along path \(P\).
- Repeat until you get stuck.

Flow value = 0

Greedy algorithm.
- Start with \(f(e) = 0\) for all edge \(e \in E\).
- Find an s-t path \(P\) where each edge has \(f(e) < c(e)\).
- Augment flow along path \(P\).
- Repeat until you get stuck.

Flow value = 20

```plaintext
Towards a Max Flow Algorithm

Greedy algorithm.
- Start with \(f(e) = 0\) for all edge \(e \in E\).
- Find an s-t path \(P\) where each edge has \(f(e) < c(e)\).
- Augment flow along path \(P\).
- Repeat until you get stuck.

Flow value = 20

greedy = 20

Towards a Max Flow Algorithm

Greedy algorithm.
- Start with \(f(e) = 0\) for all edge \(e \in E\).
- Find an s-t path \(P\) where each edge has \(f(e) < c(e)\).
- Augment flow along path \(P\).
- Repeat until you get stuck.

Flow value = 0

locally optimality \(\Rightarrow\) global optimality
```

**Towards a Max Flow Algorithm**

Flow value = 0

Flow value = 20

locally optimality \(\Rightarrow\) global optimality

Flow value = 30

opt = 30
Residual Graph

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

Residual edge.
- "Undo" flow sent.
- $e = (u, v)$ and $e^R = (v, u)$.
- Residual capacity: $c_f(e) = \begin{cases} c(e) - f(e) & \text{if } e \in E \\ f(e) & \text{if } e^R \in E \end{cases}$

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

Max-Flow Min-Cut Theorem

Augmenting path theorem. Flow $f$ is a max flow iff there are no augmenting paths.

Max-flow min-cut theorem. [Ford-Fulkerson, 1956] The value of the max flow is equal to the value of the min cut.

Proof strategy. We prove both simultaneously by showing that the following are equivalent:
- (i) There exists a cut $(A, B)$ such that $v(f) = \text{cap}(A, B)$.
- (ii) Flow $f$ is a max flow.
- (iii) There is no augmenting path relative to $f$.

(i) $\Rightarrow$ (ii) This was the corollary to weak duality lemma.

(ii) $\Rightarrow$ (iii) We show contrapositive.
- Let $f$ be a flow. If there exists an augmenting path, then we can improve $f$ by sending flow along path.

Proof of Max-Flow Min-Cut Theorem

(iii) $\Rightarrow$ (i)
- Let $f$ be a flow with no augmenting paths.
- Let $A$ be set of vertices reachable from $s$ in residual graph.
- By definition of $A$, $s \in A$.
- By definition of $f$, $t \not\in A$.

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

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

$$= \text{cap}(A, B) \quad \blacksquare$$
Augmenting Path Algorithm

Augment\((f, c, P)\) {
  \(b \leftarrow \text{bottleneck}(P)\)
  foreach \(e \in P\) {
    if \((e \in E)\) \(f(e) \leftarrow f(e) + b\)
    else \(f(e) \leftarrow f(e) - b\)
  }
  return \(f\)
}

Ford-Fulkerson\((G, s, t, c)\) {
  foreach \(e \in E\) \(f(e) \leftarrow 0\)
  \(G_r \leftarrow \text{residual graph}\)
  while (there exists augmenting path \(P\)) {
    \(f \leftarrow \text{Augment}(f, c, P)\)
    update \(G_r\)
  }
  return \(f\)
}

7.3 Choosing Good Augmenting Paths

Running Time

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

Invariant.  Every flow value \(f(e)\) and every residual capacities \(c_r(e)\) remains an integer throughout the algorithm.

Theorem.  The algorithm terminates in at most \(v(f^*) = nC\) iterations.
  Pf.  Each augmentation increase value by at least 1.

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

Integrality theorem.  If all capacities are integers, then there exists a max flow \(f\) for which every flow value \(f(e)\) is an integer.
  Pf.  Since algorithm terminates, theorem follows from invariant.

Ford-Fulkerson:  Exponential Number of Augmentations

Q.  Is generic Ford-Fulkerson algorithm polynomial in input size?

A.  No.  If max capacity is \(C\), then algorithm can take \(C\) iterations.
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]
- Max bottleneck capacity.
- Sufficiently large bottleneck capacity.
- Fewest number of edges.

Capacity Scaling

Intuition. Choosing path with highest bottleneck capacity increases flow by max possible amount.
- Don’t worry about finding exact highest bottleneck path.
- Maintain scaling parameter $\Delta$.
- Let $G_f(\Delta)$ be the subgraph of the residual graph consisting of only arcs with capacity at least $\Delta$.

Capacity Scaling: Correctness

Assumption. All edge capacities are integers between 1 and $C$.

Integrality invariant. All flow and residual capacity values are integral.

Correctness. If the algorithm terminates, then $f$ is a max flow.

Pf:
- By integrality invariant, when $\Delta = 1 \Rightarrow G_f(\Delta) = G_f$.
- Upon termination of $\Delta = 1$ phase, there are no augmenting paths.
Lemma 1. The outer while loop repeats $1 + \lceil \log_2 C \rceil$ times.

*Proof.* Initially $C/2 < \Delta \leq C$. $\Delta$ decreases by a factor of 2 each iteration. 

Lemma 2. Let $f$ be the flow at the end of a $\Delta$-scaling phase. Then the value of the maximum flow is at most $v(f) + m \Delta$.

Adding back arcs with capacity less than $\Delta$ can increase capacity of cut by at most $m\Delta$.

Lemma 3. There are at most $2m$ augmentations per scaling phase.

- Let $f$ be the flow at the end of the previous scaling phase.
- $L2 \Rightarrow v(f^*) \leq v(f) + m (2\Delta)$
- Each augmentation in a $\Delta$-phase increases $v(f)$ by at least $\Delta$.

Theorem. The scaling max-flow algorithm finds a max flow in $O(m \log C)$ augmentations. It can be implemented to run in $O(m^2 \log C)$ time.