Chapter 7
Network Flow
Soviet Rail Network, 1955

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 . . .
Flow network.

- Abstraction for material flowing through the edges.
- $G = (V, E) =$ directed graph, no parallel edges.
- Two distinguished nodes: $s =$ source, $t =$ sink.
- $c(e) =$ capacity of edge $e$, a non-negative integer.

Minimum Cut Problem
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: 

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

![Graph with labels and capacities](image)
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: \[\text{cap}(A, B) = \sum_{e \text{ out of } A} c(e)\]

Capacity = 9 + 15 + 8 + 30 = 62
Min s-t cut problem. Find an s-t cut of minimum capacity.

Minimum Cut Problem

Capacity = 10 + 8 + 10 = 28
Def. An s-t flow is a function that satisfies:

1. For each $e \in E$: \[0 \leq f(e) \leq c(e)\] (capacity)
2. For each $v \in V - \{s, t\}$: \[
\sum_{e \text{ in } v} f(e) = \sum_{e \text{ out of } v} f(e)\] (conservation)

Def. The value of a flow $f$ is: \[
v(f) = \sum_{e \text{ out of } s} f(e)\]
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 $v \in V - \{s, t\}$: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ (conservation)

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

Value = 3
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 \( v \in V - \{s, t\} \):
  \[ \sum_{e \text{ in } v} f(e) = \sum_{e \text{ out of } v} f(e) \] (conservation)

Def. The value of a flow \( f \) is:
\[ v(f) = \sum_{e \text{ out of } s} f(e). \]
Max flow problem. Find $s$-$t$ flow of maximum value.

Value $= 28$
Flows and Cuts

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

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

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

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

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

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

Diagram:

```
\begin{verbatim}
\text{Value} = 10 - 4 + 8 - 0 + 10 = 24
\end{verbatim}
```
**Flows and Cuts**

**Flow value lemma.** Let $f$ be any flow, and let $(A, B)$ be any $s$-$t$ cut. Then

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

**Pf.**

$$v(f) = \sum_{e \text{ out of } s} f(e)$$

by flow conservation, all terms except $v = s$ are 0

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

all contributions due to internal edges cancel

$$\rightarrow = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e).$$
**Flows and Cuts**

**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.

\[
\text{Cut capacity} = 30 \quad \Rightarrow \quad \text{Flow value} \leq 30
\]
Weak duality. Let $f$ be any flow. Then, for any $s$-$t$ cut $(A, B)$ we have $v(f) \leq \text{cap}(A, B)$.

\[ v(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) \leq \sum_{e \text{ out of } A} f(e) \leq \sum_{e \text{ out of } A} c(e) = \text{cap}(A, B). \]
Certificate of Optimality

**Corollary.** Let $f$ be any flow, and let $(A, B)$ be any cut. If $v(f) = \text{cap}(A, B)$, then $f$ is a max flow and $(A, B)$ is a min cut.

Value of flow = 28
Cut capacity = 28 $\Rightarrow$ Flow value $\leq$ 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 Network Diagram](image)

Flow value = 0
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
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.

$greedy = 20$

$opt = 30$

locally optimality \(\neq\) global optimality
**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.
- \( E_f = \{ e : f(e) < c(e) \} \cup \{ e : f(e) > 0 \} \).
Ford-Fulkerson Algorithm

G:

capacity
Ford-Fulkerson Algorithm

$G$:

$G_f$:
Ford-Fulkerson Algorithm

G:

s
10

2
10

3
9

4
10

5
10

t

Gf:

s
10

2
10

3
9

4
6

5
10

t

FIND PATH
Ford-Fulkerson Algorithm

\(G:\)

\(G_f:\)

UPDATE FLOW

capacity
Ford-Fulkerson Algorithm

\[ G: \]

\[ G_f: \]

\[ \text{UPDATE RESIDUAL GRAPH} \]
Ford-Fulkerson Algorithm

\[ G: \]

\[ G_f: \]

FIND PATH
Ford-Fulkerson Algorithm

\[ G: \]

\[ G_f: \]

UPDATE FLOW

capacity
Ford-Fulkerson Algorithm

$G$:  

$G_f$: UPDATE RESIDUAL GRAPH

Edge capacities:
- $s$ to $3$: 10
- $3$ to $2$: 2
- $2$ to $4$: 4
- $4$ to $5$: 6
- $5$ to $t$: 10
- $s$ to $t$: 10
- $3$ to $t$: 2
- $2$ to $s$: 6
- $4$ to $2$: 8
- $5$ to $3$: 7
- $t$ to $5$: 8
- $t$ to $s$: 2

Example residual graph:

1. $s$ to $3$: 0
2. $3$ to $2$: 2
3. $2$ to $4$: 4
4. $4$ to $5$: 6
5. $5$ to $t$: 10
6. $s$ to $t$: 10
7. $3$ to $t$: 2
8. $2$ to $s$: 6
9. $4$ to $2$: 8
10. $5$ to $3$: 7
11. $t$ to $5$: 8
12. $t$ to $s$: 2
Ford-Fulkerson Algorithm

\[ G: \]

\[ G_f: \]

FIND PATH
Ford-Fulkerson Algorithm

$G$: capacity

$G_f$: UPDATE FLOW
Ford-Fulkerson Algorithm

$G$:  

$G_f$:  

UPDATE RESIDUAL GRAPH
Ford-Fulkerson Algorithm

\[ G: \]

\[ G_f: \]
Ford-Fulkerson Algorithm

\[ G: \]

\[ G_f: \]

UPDATE FLOW

capacity
Ford-Fulkerson Algorithm

$G$: 

$s$ → $2$: 10
$2$ → $3$: 2
$2$ → $4$: 4
$3$ → $5$: 9
$4$ → $5$: 2
$5$ → $t$: 6

$G_f$: 

$s$ → $2$: 10
$2$ → $3$: 2
$3$ → $s$: 8
$2$ → $4$: 2
$4$ → $2$: 4
$4$ → $5$: 4
$5$ → $s$: 7
$5$ → $t$: 4
$t$ → $5$: 6

UPDATE RESIDUAL GRAPH
Ford-Fulkerson Algorithm

$G$:  

$G_f$:  

FIND PATH
Ford-Fulkerson Algorithm

\[ G: \]

\[ G_f: \]
Ford-Fulkerson Algorithm

\[ G: \]

\[ G_f: \]

UPDATE RESIDUAL GRAPH
Ford-Fulkerson Algorithm

\[ G: \]

G: capacity

\[ G_f: \]

Gf: FIND PATH
Ford-Fulkerson Algorithm

\[ G: \]

\[ G_f: \]

UPDATE FLOW
Ford-Fulkerson Algorithm

$G$: [Graph with nodes and edges labeled with capacities.]

$G_f$: [Graph showing the residual graph with updated capacities.]
Ford-Fulkerson Algorithm

G:

G_f:
Augmenting Path Algorithm

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

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

(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 $f$, $t \in A$.

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

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

\[
= \text{cap}(A, B)
\]
Running Time

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

**Invariant.** Every flow value $f(e)$ and every residual capacities $c_f(e)$ remains an integer throughout the algorithm.

**Theorem.** The algorithm terminates in at most $v(f^*) \leq nC$ iterations, if $f^*$ is optimal flow.
**Pf.** Each augmentation increase value by at least 1. \( \blacksquare \)

**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. \( \blacksquare \)
7.3 Choosing Good Augmenting Paths
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.

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.
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 $\delta$. 

\[ G_f \]

\[ G_f(100) \]
Capacity Scaling

Scaling-Max-Flow(G, s, t, c) {
    foreach e ∈ E  f(e) ← 0
    δ ← smallest power of 2 greater than or equal to C
    G_f ← residual graph

    while (δ ≥ 1) {
        G_f(δ) ← δ-residual graph
        while (there exists augmenting path P in G_f(δ)) {
            f ← augment(f, c, P)
            update G_f(δ)
        }
        δ ← δ / 2
    }
    return f
}
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 \( \delta = 1 \), \( G_f(\delta) = G_f \).
- Upon termination of \( \delta = 1 \) phase, there are no augmenting paths. \( \square \)
Lemma 1. The outer while loop repeats $1 + \log_2 Cn$ times.

**Pf.** Initially $\delta < 2Cn$. $\delta$ 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 $v(f) + m \delta$. ← proof on next slide

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 implies $v(f^*) \leq v(f) + m (2\delta)$.
- 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, when $m > n$. •
Lemma 2. Let $f$ be the flow at the end of a $\delta$-scaling phase. Then value of the maximum flow is at most $v(f) + m \delta$.

Proof. (almost identical to proof of max-flow min-cut theorem)

- We show that at the end of a $\delta$-phase, there exists a cut $(A, B)$ such that $\text{cap}(A, B) \leq v(f) + m \delta$.
- Choose $A$ to be the set of nodes reachable from $s$ in $G_f(\delta)$.
- By definition of $A$, $s \in A$.
- By definition of $f$, $t$ not in $A$.

\[
v(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e)
\]
\[
\geq \sum_{e \text{ out of } A} (c(e) - \delta) - \sum_{e \text{ in to } A} \delta
\]
\[
= \sum_{e \text{ out of } A} c(e) - \sum_{e \text{ in to } A} \delta - \sum_{e \text{ in to } A} \delta
\]
\[
\geq \text{cap}(A, B) - m\delta
\]