6.4 Maximum Flow

- introduction
- Ford-Fulkerson algorithm
- maxflow-mincut theorem
- analysis of running time
- Java implementation
- applications

Min-cut problem

Input. An edge-weighted digraph, source vertex $s$, and target vertex $t$.

each edge has a positive capacity

capacity

Def. A $st$-cut (cut) is a partition of the vertices into two disjoint sets, with $s$ in one set $A$ and $t$ in the other set $B$.

Def. Its capacity is the sum of the capacities of the edges from $A$ to $B$. 

capacity = $10 + 5 + 15 = 30$
Min-cut problem

Def. A \textit{st-cut (cut)} is a partition of the vertices into two disjoint sets, with \( s \) in one set \( A \) and \( t \) in the other set \( B \).

Def. Its \textit{capacity} is the sum of the capacities of the edges from \( A \) to \( B \).

Minimum \textit{st-cut (mincut) problem}. Find a cut of minimum capacity.

Maxflow: quiz 1

What is the capacity of the \textit{st-cut} \( \{ A, E, F, G \} \)?

A. 34 (8 + 11 + 9 + 6)

B. 44 (20 + 24)

C. 78 (20 + 8 + 11 + 9 + 6 + 24)

D. I don’t know.

Min-cut application (RAND 1950s)

"Free world" goal. Cut supplies (if cold war turns into real war).

rail network connecting Soviet Union with Eastern European countries
(map declassified by Pentagon in 1999)
Potential mincut application (2010s)

Government-in-power’s goal. Cut off communication to set of people.

Maxflow problem

**Input.** An edge-weighted digraph, source vertex $s$, and target vertex $t$.

Each edge has a positive capacity

**Def.** An $st$-flow (flow) is an assignment of values to the edges such that:

- Capacity constraint: $0 \leq$ edge's flow $\leq$ edge's capacity.
- Local equilibrium: inflow = outflow at every vertex (except $s$ and $t$).
Maxflow problem

**Def.** An \( st \)-flow (flow) is an assignment of values to the edges such that:
- Capacity constraint: \( 0 \leq \text{edge's flow} \leq \text{edge's capacity} \).
- Local equilibrium: inflow = outflow at every vertex (except \( s \) and \( t \)).

**Def.** The value of a flow is the inflow at \( t \).

---

**Soviet Union goal.** Maximize flow of supplies to Eastern Europe.

---

**Potential maxflow application (2010s)**

"Free world" goal. Maximize flow of information to specified set of people.
Summary

Input. A weighted digraph, source vertex $s$, and target vertex $t$.
Mincut problem. Find a cut of minimum capacity.
Maxflow problem. Find a flow of maximum value.

Remarkable fact. These two problems are dual!

Ford-Fulkerson algorithm

Initialization. Start with 0 flow.

Idea: increase flow along augmenting paths

Augmenting path. Find an undirected path from $s$ to $t$ such that:
- Can increase flow on forward edges (not full).
- Can decrease flow on backward edge (not empty).

6.4 Maximum Flow

• introduction
• Ford-Fulkerson algorithm
• maxflow-mincut theorem
• analysis of running time
• Java implementation
• applications
Idea: increase flow along augmenting paths

Augmenting path. Find an undirected path from $s$ to $t$ such that:
- Can increase flow on forward edges (not full).
- Can decrease flow on backward edge (not empty).

2nd augmenting path

3rd augmenting path

Idea: increase flow along augmenting paths

Augmenting path. Find an undirected path from $s$ to $t$ such that:
- Can increase flow on forward edges (not full).
- Can decrease flow on backward edge (not empty).

4th augmenting path

Termination. All paths from $s$ to $t$ are blocked by either a
- Full forward edge.
- Empty backward edge.

no more augmenting paths

Termination. All paths from $s$ to $t$ are blocked by either a
- Full forward edge.
- Empty backward edge.
Maxflow: quiz 2

Which is an augmenting path of highest bottleneck capacity?

A. $A \rightarrow F \rightarrow G \rightarrow H$
B. $A \rightarrow F \rightarrow B \rightarrow G \rightarrow H$
C. $A \rightarrow F \rightarrow B \rightarrow G \rightarrow D \rightarrow H$
D. I don’t know.

Ford-Fulkerson algorithm

Ford–Fulkerson algorithm

Start with 0 flow.
While there exists an augmenting path:
- find an augmenting path
- compute bottleneck capacity
- increase flow on that path by bottleneck capacity

Fundamental questions.
- How to compute a mincut?
- How to find an augmenting path?
- If FF terminates, does it always compute a maxflow?
- Does FF always terminate? If so, after how many augmentations?

Relationship between flows and cuts

Def. The net flow across a cut $(A, B)$ is the sum of the flows on its edges from $A$ to $B$ minus the sum of the flows on its edges from $B$ to $A$. 

$\text{net flow across cut} = 5 + 10 + 10 = 25$

value of flow = 25
**Def.** The **net flow across** a cut \((A, B)\) is the sum of the flows on its edges from \(A\) to \(B\) minus the sum of the flows on its edges from \(B\) to \(A\).

\[
\text{net flow across cut} = (10 + 10 + 5 + 10 + 0 + 0) - (5 + 5 + 0 + 0) = 25
\]

**Flow-value lemma.** Let \(f\) be any flow and let \((A, B)\) be any cut. Then, the net flow across \((A, B)\) equals the value of \(f\).

**Intuition.** Conservation of flow.

**Pf.** By induction on the size of \(B\).
- Base case: \(B = \{t\}\).
- Induction step: remains true by local equilibrium when moving any vertex from \(A\) to \(B\).

**Corollary.** Outflow from \(s = \text{inflow to } t = \text{value of flow.}\)

**Weak duality.** Let \(f\) be any flow and let \((A, B)\) be any cut. Then, the value of the flow \(\leq\) the capacity of the cut.

**Pf.** Value of flow \(f = \text{net flow across cut } (A, B) \leq \text{capacity of cut } (A, B)\).
Maxflow-mincut theorem

Augmenting path theorem. A flow $f$ is a maxflow iff no augmenting paths.
Maxflow-mincut theorem. Value of the maxflow = capacity of mincut.

Pf. The following three conditions are equivalent for any flow $f$:
\begin{enumerate}[i.]
  \item There exists a cut whose capacity equals the value of the flow $f$.
  \item $f$ is a maxflow.
  \item There is no augmenting path with respect to $f$.
\end{enumerate}

\begin{itemize}
  \item $[i \Rightarrow ii]$ \\
    Suppose that $(A, B)$ is a cut with capacity equal to the value of $f$.
    Then, the value of any flow $f' \leq \text{capacity of } (A, B) = \text{value of } f$.
    Thus, $f$ is a maxflow.
  \item $[ii \Rightarrow iii]$ We prove contrapositive: $\neg iii \Rightarrow \neg ii$.
    \begin{itemize}
      \item Suppose that there is an augmenting path with respect to $f$.
      \item Can improve flow $f$ by sending flow along this path.
      \item Thus, $f$ is not a maxflow.
    \end{itemize}
\end{itemize}

Maxflow-mincut theorem

Augmenting path theorem. A flow $f$ is a maxflow iff no augmenting paths.
Maxflow-mincut theorem. Value of the maxflow = capacity of mincut.

Pf. The following three conditions are equivalent for any flow $f$:
\begin{enumerate}[i.]
  \item There exists a cut whose capacity equals the value of the flow $f$.
  \item $f$ is a maxflow.
  \item There is no augmenting path with respect to $f$.
\end{enumerate}

\begin{itemize}
  \item $[i \Rightarrow ii]$ \\
    Let $f'$ be a flow with no augmenting paths.
    Let $A$ be set of vertices connected to $s$ by an undirected path with no full forward or empty backward edges
    By definition of cut $A$, $s$ is in $A$.
    By definition of cut $A$ and flow $f$, $t$ is in $B$.
    Capacity of cut $= \text{net flow across cut} = \text{value of flow } f$.
\end{itemize}

Computing a mincut from a maxflow

To compute mincut $(A, B)$ from maxflow $f$:
\begin{itemize}
  \item By augmenting path theorem, no augmenting paths with respect to $f$.
  \item Compute $A = \text{set of vertices connected to } s \text{ by an undirected path with no full forward or empty backward edges}$.
\end{itemize}
6.4 Maximum Flow

- Introduction
- Ford-Fulkerson algorithm
- Maxflow-mincut theorem
- Analysis of running time
- Java implementation
- Applications

Ford-Fulkerson Algorithm

Start with 0 flow.
While there exists an augmenting path:
- find an augmenting path
- compute bottleneck capacity
- increase flow on that path by bottleneck capacity

Fundamental questions.
- How to compute a mincut? Easy. ✔
- How to find an augmenting path? BFS works well.
- If FF terminates, does it always compute a maxflow? Yes. ✔
- Does FF always terminate? If so, after how many augmentations?

Yes, provided edge capacities are integers (or augmenting paths are chosen carefully).
Requires clever analysis.

Ford-Fulkerson Algorithm with Integer Capacities

Important special case. Edge capacities are integers between 1 and $U$.

Invariant. The flow is integral throughout Ford-Fulkerson.

Pf. [by induction]
- Bottleneck capacity is an integer.
- Flow on an edge increases/decreases by bottleneck capacity.

Proposition. Number of augmentations $\leq$ the value of the maxflow.

Pf. Each augmentation increases the value by at least 1.

Integrality theorem. There exists an integral maxflow.

Pf. Ford-Fulkerson terminates and maxflow that it finds is integer-valued.

Bad Case for Ford-Fulkerson

Bad News. Even when edge capacities are integers, number of augmenting paths could be equal to the value of the maxflow.
Bad case for Ford-Fulkerson

**Bad news.** Even when edge capacities are integers, number of augmenting paths could be equal to the value of the maxflow.

1st iteration

2nd iteration

3rd iteration

4th iteration
Bad case for Ford-Fulkerson

**Bad news.** Even when edge capacities are integers, number of augmenting paths could be equal to the value of the maxflow.

\[ \ldots \]

Good news. This case is easily avoided. [use shortest/fattest path]

\[ \ldots \]
How to choose augmenting paths?

Choose augmenting paths with:

- Shortest path: fewest number of edges.
- Fattest path: max bottleneck capacity.

Flow network representation

Flow edge data type. Associate flow \( f_e \) and capacity \( c_e \) with edge \( e = v \rightarrow w \).

Flow network data type. Must be able to process edge \( e = v \rightarrow w \) in either direction: include \( e \) in adjacency lists of both \( v \) and \( w \).

Residual (spare) capacity.

- Forward edge: residual capacity = \( c_e - f_e \).
- Backward edge: residual capacity = \( f_e \).

Augment flow.

- Forward edge: add \( \Delta \).
- Backward edge: subtract \( \Delta \).
Flow network representation

**Residual network.** A useful view of a flow network.

Key point. Augmenting paths in original network are in one-to-one correspondence with directed paths in residual network.

Flow edge API

```java
public class FlowEdge {
    private final int v, w;  // from and to
    private final double capacity;  // capacity
    private double flow;  // flow

    public FlowEdge(int v, int w, double capacity) {
        this.v = v;
        this.w = w;
        this.capacity = capacity;
    }

    public int from() { return v; }
    public int to() { return w; }
    public double capacity() { return capacity; }
    public double flow() { return flow; }

    public int other(int vertex) {
        if (vertex == v) return w;
        else if (vertex == w) return v;
        else throw new IllegalArgumentException();
    }

    public double residualCapacityTo(int vertex) {
        return flow;
    }

    public void addResidualFlowTo(int vertex, double delta) {
        if (vertex == v) flow += delta;
        else if (vertex == w) flow -= delta;
        else throw new IllegalArgumentException();
    }
}
```

Flow edge: Java implementation (continued)
Flow network API

```java
public class FlowNetwork {
    public FlowNetwork(int V) { /* create an empty flow network with V vertices */ }
    public FlowNetwork(In in) { /* construct flow network input stream */ }
    void addEdge(FlowEdge e) { /* add flow edge e to this flow network */ }
    Iterable<FlowEdge> adj(int v) { /* forward and backward edges incident to/from v */ }
    Iterable<FlowEdge> edges() { /* all edges in this flow network */ }
    int V() { /* number of vertices */ }
    int E() { /* number of edges */ }
    String toString() { /* string representation */ }
}
```

**Conventions.** Allow self-loops and parallel edges.

Flow network: Java implementation

```java
public class FlowNetwork {
    private final int V;
    private Bag<FlowEdge>[] adj;
    public FlowNetwork(int V) {
        this.V = V;
        adj = (Bag<FlowEdge>[]) new Bag[V];
        for (int v = 0; v < V; v++)
            adj[v] = new Bag<FlowEdge>();
    }
    public void addEdge(FlowEdge e) {
        int v = e.from();
        int w = e.to();
        adj[v].add(e);
        adj[w].add(e);
    }
    public Iterable<FlowEdge> adj(int v) {
        return adj[v];
    }
}
```

same as EdgeWeightedGraph, but adjacency lists of FlowEdges instead of Edges

add forward edge
add backward edge

Flow network: adjacency-lists representation

Maintain vertex-indexed array of FlowEdge lists (use Bag abstraction).

```
<table>
<thead>
<tr>
<th>V</th>
<th>E</th>
<th>adj[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>3.0</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>1.0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3.0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2.0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2.0</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>3.0</td>
</tr>
</tbody>
</table>
```

Note. Adjacency list includes edges with 0 residual capacity. (residual network is represented implicitly)

Finding a shortest augmenting path (cf. breadth-first search)

```java
private boolean hasAugmentingPath(FlowNetwork G, int s, int t) {
    FlowEdge[] edgeTo = new FlowEdge[G.V()];
    boolean[] marked = new boolean[G.V()];
    Bag<Integer> queue = new Bag<Integer>();
    marked[s] = true;
    while (!queue.isEmpty()) {
        int v = queue.dequeue();
        for (FlowEdge e : G.adj(v)) {
            int w = e.other(v);
            if (!marked[w] && e.residualCapacityTo(w) > 0) {
                edgeTo[w] = e;
                marked[w] = true;
                queue.enqueue(w);
                if (w == t) {
                    System.out.println("Found path from s to w in the residual network!");
                    return true;
                }
            }
        }
    }
    return false;
}
```

is t reachable from s in residual network?
Ford-Fulkerson: Java implementation

```java
public class FordFulkerson {
    private boolean[] marked; // true if s->v path in residual network
    private FlowEdge[] edgeTo; // last edge on s->v path
    private double value; // value of flow

    public FordFulkerson(FlowNetwork G, int s, int t) {
        value = 0.0;
        while (hasAugmentingPath(G, s, t)) {
            double bottle = Double.POSITIVE_INFINITY;
            for (int v = t; v != s; v = edgeTo[v].other(v))
                bottle = Math.min(bottle, edgeTo[v].residualCapacityTo(v));
            for (int v = t; v != s; v = edgeTo[v].other(v))
                edgeTo[v].addResidualFlowTo(v, bottle);
            value += bottle;
        }
    }

    private boolean hasAugmentingPath(FlowNetwork G, int s, int t) {
        /* See previous slide. */
        return value; }

    public double value() {
        return value; }

    public boolean inCut(int v) {
        return marked[v]; }
}
```

Maxflow and mincut applications

Maxflow/mincut is a widely applicable problem-solving model.

- Data mining.
- Open-pit mining.
- Bipartite matching.
- Network reliability.
- Baseball elimination.
- Image segmentation.
- Network connectivity.
- Distributed computing.
- Security of statistical data.
- Egalitarian stable matching.
- Multi-camera scene reconstruction.
- Sensor placement for homeland security.
- Many, many, more.

Bipartite matching problem

N students apply for N jobs.

Each gets several offers.

Is there a way to match all students to jobs?

<table>
<thead>
<tr>
<th>Student</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Adobe</td>
</tr>
<tr>
<td>Bob</td>
<td>Amazon</td>
</tr>
<tr>
<td>Carol</td>
<td>Google</td>
</tr>
<tr>
<td>Dave</td>
<td>Facebook</td>
</tr>
<tr>
<td>Eliza</td>
<td>Yahoo</td>
</tr>
</tbody>
</table>

bipartite matching problem

1 Alice Adobe Alice
2 Bob Adobe Alice
3 Carol Adobe Bob
4 Dave Amazon Alice
5 Eliza Amazon Bob
6 Adobe Alice Carol
7 Amazon Alice Bob
8 Facebook Alice
9 Google Alice Carol
10 Yahoo Alice Eliza
Bipartite matching problem

Given a bipartite graph, find a perfect matching.

**Network flow formulation of bipartite matching**

- Create $s$, $t$, one vertex for each student, and one vertex for each job.
- Add edge from $s$ to each student (capacity 1).
- Add edge from each job to $t$ (capacity 1).
- Add edge from student to each job offered (infinite capacity).

**What the mincut tells us**

**Goal.** When no perfect matching, explain why.

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$.
What the mincut tells us

Mincut. Consider mincut \((A, B)\).
\( \cdot \) Let \(S\) = students on \(x\) side of cut.
\( \cdot \) Let \(T\) = companies on \(x\) side of cut.
\( \cdot \) Fact: \(|S| > |T|\); students in \(S\) can be matched only to companies in \(T\).

\[
S = \{2, 4, 5\} \quad T = \{7, 10\}
\]

Let \(S\) = students on \(x\) side of cut.
Let \(T\) = companies on \(x\) side of cut.

\[
|S| > |T|
\]

student in \(S\) can be matched only to companies in \(T\)

Bottom line. When no perfect matching, mincut explains why.

Baseball elimination problem

Q. Which teams have a chance of finishing the season with the most wins?

<table>
<thead>
<tr>
<th>i</th>
<th>team</th>
<th>wins</th>
<th>losses</th>
<th>to play</th>
<th>ATL</th>
<th>PHI</th>
<th>NYM</th>
<th>WAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Atlanta</td>
<td>83</td>
<td>71</td>
<td>8</td>
<td>–</td>
<td>1</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>Philly</td>
<td>80</td>
<td>79</td>
<td>3</td>
<td>1</td>
<td>–</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>New York</td>
<td>78</td>
<td>78</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>Washington</td>
<td>77</td>
<td>82</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>–</td>
</tr>
</tbody>
</table>

Philadelphia is mathematically eliminated.
\( \cdot \) Philadelphia finishes with \(\leq 83\) wins.
\( \cdot \) Either New York or Atlanta will finish with \(\geq 84\) wins.

Observation. Answer depends not only on how many games already won and left to play, but on whom they’re against.

Baseball elimination problem

Q. Which teams have a chance of finishing the season with the most wins?

<table>
<thead>
<tr>
<th>i</th>
<th>team</th>
<th>wins</th>
<th>losses</th>
<th>to play</th>
<th>NYY</th>
<th>BAL</th>
<th>BOS</th>
<th>TOR</th>
<th>DET</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>New York</td>
<td>75</td>
<td>59</td>
<td>28</td>
<td>–</td>
<td>3</td>
<td>8</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>Baltimore</td>
<td>71</td>
<td>63</td>
<td>28</td>
<td>3</td>
<td>–</td>
<td>2</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>Boston</td>
<td>69</td>
<td>66</td>
<td>27</td>
<td>8</td>
<td>2</td>
<td>–</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>Toronto</td>
<td>63</td>
<td>72</td>
<td>27</td>
<td>7</td>
<td>7</td>
<td>0</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>Detroit</td>
<td>49</td>
<td>86</td>
<td>27</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>–</td>
</tr>
</tbody>
</table>

Detroit is mathematically eliminated.
\( \cdot \) Detroit finishes with \(\leq 76\) wins.
\( \cdot \) Wins for \(R = \{NYY, BAL, BOS, TOR\}\) = 278.
\( \cdot \) Remaining games among \(\{NYY, BAL, BOS, TOR\}\) = \(3 + 8 + 7 + 2 + 7 = 27\).
\( \cdot \) Average team in \(R\) wins \(305/4 = 76.25\) games.
Baseball elimination problem: maxflow formulation

**Intuition.** Remaining games flow from $s$ to $t$.

![Game graph](image)

**Fact.** Team 4 not eliminated iff all edges pointing from $s$ are full in maxflow.

Maximum flow algorithms: practice

**Warning.** Worst-case order-of-growth is generally not useful for predicting or comparing maxflow algorithm performance in practice.

**Best in practice.** Push-relabel method with gap relabeling: $E^{3/2}$.

Maximum flow algorithms: theory

(Yet another) holy grail for theoretical computer scientists.

<table>
<thead>
<tr>
<th>year</th>
<th>method</th>
<th>worst case</th>
<th>discovered by</th>
</tr>
</thead>
<tbody>
<tr>
<td>1951</td>
<td>simplex</td>
<td>$E^3 U$</td>
<td>Dantzig</td>
</tr>
<tr>
<td>1955</td>
<td>augmenting path</td>
<td>$E^2 U$</td>
<td>Ford-Fulkerson</td>
</tr>
<tr>
<td>1970</td>
<td>shortest augmenting path</td>
<td>$E^3$</td>
<td>Dinit, Edmonds-Karp</td>
</tr>
<tr>
<td>1970</td>
<td>fastest augmenting path</td>
<td>$E^2 \log E \log(U)$</td>
<td>Dinit, Edmonds-Karp</td>
</tr>
<tr>
<td>1977</td>
<td>blocking flow</td>
<td>$E^{3/2}$</td>
<td>Cherkasy</td>
</tr>
<tr>
<td>1978</td>
<td>blocking flow</td>
<td>$E^{3/3}$</td>
<td>Galil</td>
</tr>
<tr>
<td>1983</td>
<td>dynamic trees</td>
<td>$E^2 \log E$</td>
<td>Sleator-Tarjan</td>
</tr>
<tr>
<td>1985</td>
<td>capacity scaling</td>
<td>$E^2 \log U$</td>
<td>Gabow</td>
</tr>
<tr>
<td>1997</td>
<td>length function</td>
<td>$E^{3/2} \log E \log U$</td>
<td>Goldberg-Rao</td>
</tr>
<tr>
<td>2012</td>
<td>compact network</td>
<td>$E^2 \log E$</td>
<td>Orlin</td>
</tr>
</tbody>
</table>

maxflow algorithms for sparse networks with $E$ edges, integer capacities between 1 and $U$

Summary

**Mincut problem.** Find an $st$-cut of minimum capacity.

**Maxflow problem.** Find an $st$-flow of maximum value.

**Duality.** Value of the maxflow $=$ capacity of mincut.

Proven successful approaches.
- Ford-Fulkerson (various augmenting-path strategies).
- Preflow-push (various versions).

Open research challenges.
- Practice: solve real-world maxflow/mincut problems in linear time.
- Theory: prove it for worst-case inputs.
- Still much to be learned!