6.4 Maximum Flow

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

Mincut problem

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

Each edge has a positive 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 $= 10 + 5 + 15 = 30$
**MinCut Problem**

**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 \( B \).

---

**MinCut Application (RAND 1950s)**

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

---

**Potential MinCut Application (2010s)**

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

---

**Figure 2** From Harris and Ross [1955]: Schematic diagram of the rail network connecting Soviet Union with Eastern European countries (map declassified by Pentagon in 1999)
Maxflow problem

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

- Each edge has a positive capacity.

![Diagram of a maxflow problem with a source vertex $s$, a target vertex $t$, and edges with capacities labeled.]

**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 $t$).

**Def.** The value of a flow is the inflow at $t$.

- We assume no edges point to $s$ or from $t$.

![Diagram showing a flow with values at each vertex and capacities on the edges.]

Maximum $st$-flow (maxflow) problem. Find a flow of maximum value.

- Value = $5 + 10 + 10 = 25$

![Diagram illustrating the maximum flow with values at each vertex and capacities on the edges.]

- Value = $8 + 10 + 10 = 28$
Maxflow application (Tolstoi 1930s)

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

![Map of rail network connecting Soviet Union with Eastern European countries](image)

Potential maxflow application (2010s)

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

figure 2

facebook graph

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.

![Diagram of digraph](image)

value of flow = 28

![Diagram of digraph](image)

capacity of cut = 28

Remarkable fact. These two problems are dual!

6.4 Maximum Flow

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

http://algs4.cs.princeton.edu

ROBERT SEDGEWICK | KEVIN WAYNE

Algorithms
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).

2nd augmenting path

3rd augmenting path

backward edge (not empty)
**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

![Augmenting path diagram](image)

**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

**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?

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

no more augmenting paths

![Termination diagram](image)

**6.4 Maximum Flow**
**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 \(A\).

\[
\text{net flow across cut} = 5 + 10 + 0 = 15
\]

\[
\text{value of flow} = 25
\]

**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 \(s\) = inflow to \(t\) = 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.

**Proof.** Value of flow \( f = \) net flow across cut \((A, B)\) \( \leq \) 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.

**Proof.** The following three conditions are equivalent for any flow \( f \):

i. There exists a cut whose capacity equals the value of the flow \( f \).
ii. \( f \) is a maxflow.
iii. There is no augmenting path with respect to \( f \).

\[ i \Rightarrow ii \]

- Suppose that \((A, B)\) is a cut with capacity equal to the value of \( f \).
- Then, the value of any flow \( f' \) \( \leq \) capacity of \((A, B)\) = value of \( f \).
- Thus, \( f \) is a maxflow.

\[ ii \Rightarrow iii \]

- We prove contrapositive: \( \neg iii \Rightarrow \neg ii \).
- Suppose that there is an augmenting path with respect to \( f \).
- Can improve flow \( f \) by sending flow along this path.
- Thus, \( f \) is not a maxflow.

\[ iii \Rightarrow i \]

- Suppose that there is no augmenting path with respect to \( f \).
- Let \((A, B)\) be a cut where \( A \) is the set of vertices connected to \( s \) by an undirected path with no full forward or empty backward edges.
- By definition of cut, \( s \) is in \( A \).
- Since no augmenting path, \( t \) is in \( B \).
- Capacity of cut = net flow across cut \( = value \ of \ flow \ f \).
Computing a mincut from a maxflow

To compute mincut \((A, B)\) from maxflow \(f\):

- By augmenting path theorem, no augmenting paths with respect to \(f\).
- Compute \(A\) = set of vertices connected to \(s\) by an undirected path with no full forward or empty backward edges.

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

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?

Important special case. Edge capacities are integers between 1 and \(U\).

Invariant. The flow is integer-valued 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 integer-valued 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.

1. **Initialize with 0 flow**

2. **1st iteration**

3. **2nd iteration**

4. **3rd 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.

![4th iteration diagram](image)

![199th iteration diagram](image)

![200th iteration diagram](image)
**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.  
*can be exponential in input size*

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

---

**How to choose augmenting paths?**

Use care when selecting augmenting paths.  
- Some choices lead to exponential algorithms.  
- Clever choices lead to polynomial algorithms.

### Augmenting Path Choices

<table>
<thead>
<tr>
<th>Augmenting Path</th>
<th>Number of Paths</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>random path</td>
<td>(\leq E U)</td>
<td>randomized queue</td>
</tr>
<tr>
<td>DFS path</td>
<td>(\leq E U)</td>
<td>stack (DFS)</td>
</tr>
<tr>
<td>shortest path</td>
<td>(\leq \frac{1}{2} E V)</td>
<td>queue (BFS)</td>
</tr>
<tr>
<td>fattest path</td>
<td>(\leq E \ln(E U))</td>
<td>priority queue</td>
</tr>
</tbody>
</table>

- **digraph with** \(V\) **vertices,** \(E\) **edges,** and integer capacities between 1 and \(U\)**

---

**6.4 Maximum Flow**

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

**Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems**

**Jaeck Edmonds**  
University of California, Berkeley, California

**Richard M. Karp**  
University of California, Berkeley, California

**Algorithm for Solution of a Problem of Maximum Flow in a Network with Fixed Capacity**

**EDSAC**

**E. A. Rice**

Different measures of the speed of an algorithm for the problem of minimal cost flow in a network and the requirements it needs are given in [1].  These rates of growth on algorithms solving the problem in the case when the initial flow are integers are, when in equilibrium, nonnegative.  In the general case, these algorithm requires polynomial running at the initial flow, i.e., only non-negative values of the problem is possible.  In this respect the optimality of convergence of the algorithm is inversely proportional to the solution precision.

---

**Edmonds–Karp 1972 (USA)**  
**Dinic 1970 (Soviet Union)**
Flow network representation

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

```
flow f_e capacity c_e
```

**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 (sare) 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 edge API**

```java
public class FlowEdge {
    // ... (code snippet)
}
```

**Flow edge: Java implementation**

```java
public class FlowEdge {
    private final int from = ...;
    private final int to = ...;
    private final double capacity = ...;
    private double flow = ...;
    private double residualCapacityTo(int v) { ...; }
    void addResidualFlowTo(int v, double delta) { ...; }
    ... (code snippet)
}
```

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

- includes all edges with positive residual capacity
- residual edge (not empty)
- forward edge (not full)

**Key point.** Augmenting paths in original network are in 1-1 correspondence with directed paths in residual network.
Flow edge: Java implementation (continued)

```java
public double residualCapacityTo(int vertex) {
    if (vertex == v) return flow;
    else if (vertex == w) return capacity - flow;
    else throw new IllegalArgumentException();
}
```

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

Flow network API

```java
public class FlowNetwork
    FlowNetwork(int V)
    FlowNetwork(InputStream in)
    void addEdge(FlowEdge e)
    Iterable<FlowEdge> adj(int v)
    Iterable<FlowEdge> edges()
    int V()
    int E()
    String toString()
```

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];
    }
}
```

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

Flow network: adjacency-lists representation

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) {
    edgeTo = new FlowEdge[G.V()];
    marked = new boolean[G.V()];
    Queue<Integer> queue = new Queue<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);
            }
        }
    }
    return marked[t];
}
```

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) {
        // compute edgeTo[]
        double bottle = Double.POSITIVE_INFINITY;
        for (int v = t; v != s; v = edgeTo[v].other(v))
            bottle = Math.min(bottle, edgeTo[v].residualCapacityTo(v));

        // compute bottleneck capacity
        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. */
    }

    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?

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).

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

Goal. When no perfect matching, explain why.

\[
S = \{2, 4, 5\} \\
T = \{7, 10\} \\
\text{student in } S \text{ can be matched only to companies in } T \\
|S| > |T| \\
\text{no perfect matching exists}
\]

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>MON</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Atlanta</td>
<td>83</td>
<td>71</td>
<td>8</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td></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>Montreal</td>
<td>77</td>
<td>82</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Montreal is mathematically eliminated.
- Montreal finishes with \(\leq 80\) wins.
- Atlanta already has 83 wins.

Philadelphia is mathematically eliminated.
- Philadelphia finishes with \(\leq 83\) wins.
- 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>3</td>
<td>8</td>
<td>7</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Baltimore</td>
<td>71</td>
<td>63</td>
<td>28</td>
<td>3</td>
<td>7</td>
<td>7</td>
<td>4</td>
<td></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>0</td>
<td>0</td>
<td></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>0</td>
<td></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.

- Detroit finishes with ≤ 76 wins.
- Wins for \( R = \{ \text{NYY, BAL, BOS, TOR} \} = 278. \)
- Remaining games among \( \{ \text{NYY, BAL, BOS, TOR} \} = 3 + 8 + 7 + 2 + 7 = 27. \)
- Average team in \( R \) wins \( 305/4 = 76.25 \) games.

**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>Dinitz, Edmonds-Karp</td>
</tr>
<tr>
<td>1970</td>
<td>fattest augmenting path</td>
<td>( E^2 \log E \log (E U) )</td>
<td>Dinitz, Edmonds-Karp</td>
</tr>
<tr>
<td>1977</td>
<td>blocking flow</td>
<td>( E^{5/2} )</td>
<td>Cherkesky</td>
</tr>
<tr>
<td>1978</td>
<td>blocking flow</td>
<td>( E^{3/3} )</td>
<td>Calil</td>
</tr>
<tr>
<td>1983</td>
<td>dynamic trees</td>
<td>( E^3 \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^{5/2} \log E \log U )</td>
<td>Goldberg-Rao</td>
</tr>
<tr>
<td>2012</td>
<td>compact network</td>
<td>( E^3 / \log E )</td>
<td>Orlin</td>
</tr>
</tbody>
</table>

**Baseball elimination problem: maxflow formulation**

*Intuition.* Remaining games flow from \( s \) to \( t \).

- Games left between 1 and 2.
- Team 2 can still win this many more games.
- Team vertices (each team other than 4).
- Game vertices (each pair of teams other than 4).

*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: \( E^{3/2} \).

---

**On Implementing Push-Relabel Method for the Maximum Flow Problem**

Boris V. Cherkassky and Andrew V. Goldberg

*Abstract.* We study efficient implementations of the push-relabel method for the maximum flow problem. The running times are better than the previous codes, and much faster in some problem families. The running time is due to the combination of heuristics used in our implementation. We also exhibit a family of problems for which the running time of all known methods seems to have a roughly quadratic growth rate.
Summary

Min-cut problem. Find an $st$-cut of minimum capacity.
Max-flow problem. Find an $st$-flow of maximum value.
Duality. Value of the max-flow = capacity of min-cut.

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!