6.4 Maximum Flow

- introduction
- Ford–Fulkerson algorithm
- maxflow–mincut theorem
- analysis of running time
- Java implementation (see textbook)
- applications

https://algs4.cs.princeton.edu
6.4 Maximum Flow

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

**Input.** A digraph with positive edge weights, source vertex $s$, and target vertex $t$. 

![Diagram of a digraph with edge weights and capacities](image-url)
**Def.** A \textit{st-cut (cut)} is a partition of the vertices into two disjoint sets, with \textit{s} in one set \textit{A} and \textit{t} in the other set \textit{B}.

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

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{mincut_diagram.png}
\caption{Example of a \textit{min-cut} problem with capacities 10, 5, and 15.}
\end{figure}

\textit{capacity} = 10 + 5 + 15 = 30
**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*.

![Diagram showing the mincut problem with capacities 10, 8, and 16, and the mincut capacity calculated as 34.](image-url)

capacity = 10 + 8 + 16 = 34
**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 $B$.

**Minimum st-cut (mincut) problem.** Find a cut of minimum capacity.

capacity $= 10 + 8 + 10 = 28$
What is the capacity of the cut \{A, E, F, G\}?

A. 11 \((20 + 25 - 8 - 11 - 9 - 6)\)

B. 34 \((8 + 11 + 9 + 6)\)

C. 45 \((20 + 25)\)

D. 79 \((20 + 25 + 8 + 11 + 9 + 6)\)
“Free world” goal. Disrupt rail network (if Cold War turns into real war).

rail network connecting Soviet Union with Eastern European countries
(map declassified by Pentagon in 1999)
Though maximum flow algorithms have a long history, revolutionary progress is still being made.

BY ANDREW V. GOLDBERG AND ROBERT E. TARJAN

Efficient Maximum Flow Algorithms

gorithms in more detail. We restrict ourselves to basic maximum flow algorithms and do not cover interesting special cases (such as undirected graphs, planar graphs, and bipartite matchings) or generalizations (such as minimum-cost and multi-commodity flow problems).

Before formally defining the maximum flow and the minimum cut problems, we give a simple example of each problem: For the maximum flow example, suppose we have a graph that represents an oil pipeline network from an oil well to an oil depot. Each arc has a capacity, or maximum number of liters per second that can flow through the corresponding pipe. The goal is to find the maximum number of liters per second (maximum flow) that can be shipped from well to depot. For the minimum cut problem, we want to find the set of pipes of the smallest total capacity such that removing the pipes disconnects the oil well from the oil depot (minimum cut).

The maximum flow, minimum cut
Maxflow problem

**Input.** A digraph with positive edge weights, source vertex $s$, and target vertex $t$. 
Maxflow problem

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

- Capacity constraints: $0 \leq$ edge’s flow $\leq$ edge’s capacity.
- Flow conservation constraints: 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 constraints: $0 \leq$ edge’s flow $\leq$ edge’s capacity.
- Flow conservation constraints: inflow = outflow at every vertex (except $s$ and $t$).

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

we assume no edges incident to $s$ or from $t$

value = 5 + 10 + 10 = 25
Maxflow problem

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

Def. The \textbf{value} of a flow is the inflow at \(t\).

\textbf{Maximum st-flow (maxflow) problem.} Find a flow of maximum value.
Maxflow application (Tolstoï 1930s)

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

rail network connecting Soviet Union with Eastern European countries
(map declassified by Pentagon in 1999)
Summary

**Input.** A digraph with positive edge weights, 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! [stay tuned]
6.4 Maximum Flow

- introduction
- Ford–Fulkerson algorithm
- maxflow–mincut theorem
- analysis of running time
- Java implementation
- applications
**Initialization.** Start with 0 flow.
Ford–Fulkerson algorithm demo

**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).

**1st augmenting path**

![Graph diagram showing the first augmenting path and bottleneck capacity]

**Impact:** Increases value of flow (maintains capacity and flow conservation constraints)

bottleneck capacity = 10

$0 + 10 = 10$
Ford–Fulkerson algorithm demo

**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**

![Diagram showing the 2nd augmenting path]
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).

**3rd augmenting path**

\[ 20 + 5 = 25 \]
Ford–Fulkerson algorithm demo

**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**

\[ 25 + 3 = 28 \]
Termination. All paths from $s$ to $t$ are blocked by either a

- Full forward edge.
- Empty backward edge.

Ford–Fulkerson algorithm demo

no more augmenting paths
Which is an augmenting path?

A. \( A \to F \to G \to D \to H \)

B. \( A \to F \to B \to G \to C \to D \to H \)

C. Both A and B.

D. Neither A nor B.
What is the bottleneck capacity of the augmenting path $A \rightarrow F \rightarrow B \rightarrow G \rightarrow C \rightarrow D \rightarrow H$?

A. 4
B. 5
C. 6
D. 7
Ford–Fulkerson algorithm

Start with 0 flow.
While there exists an augmenting path:
  – find an augmenting path $P$
  – compute bottleneck capacity of $P$
  – update flow on $P$ by bottleneck capacity

Fundamental questions.

• How to find an augmenting path?
• How many augmenting paths?
• Guaranteed to compute a maxflow?
• Given a maxflow, how to compute a mincut?
6.4 Maximum Flow

- introduction
- Ford-Fulkerson algorithm
- maxflow-mincut theorem
- analysis of running time
- Java implementation
- applications
Relationship between flows and cuts

**Def.** Given a flow \( f \), 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
Relationship between flows and cuts

**Def.** Given a flow $f$, 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 $A$.

**Diagram:**

```
net flow across cut = 10 + 5 + 10 = 25
```

```
value of flow = 25
```
**Def.** Given a flow $f$, 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 $A$.

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

**value of flow** = 25
Maxflow: quiz 4

Given the flow f below, what is the net flow across the cut \( \{ A, E, F, G \} \)?

A. 11 \((20 + 25 - 8 - 11 - 9 - 6)\)

B. 26 \((20 + 22 - 8 - 4 - 4 - 0)\)

C. 42 \((20 + 22)\)

D. 45 \((20 + 25)\)
Relationship between flows and cuts

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

Intuition. Conservation of flow.

Pf. By induction on the number of vertices in $B$.

- Base case: $B = \{ t \}$.
- Induction step: remains true when moving any vertex $v$ from $A$ to $B$ (because of flow conservation constraint for vertex $v$)

Corollary. Outflow from $s =$ inflow to $t =$ value of flow. 

we assume no edges incident to $s$ or from $t$
**Weak duality.** Let $f$ be any flow and let $(A, B)$ be any cut. Then, the value of flow $f \leq$ the capacity of cut $(A, B)$.

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

**Equivalent.** Value of maxflow $\leq$ capacity of mincut.
Maxflow–mincut theorem


Augmenting path theorem. A flow $f$ is a maxflow if and only if no augmenting paths.

Pf. For any flow $f$, the following three conditions are equivalent:

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

[i $\Rightarrow$ ii] We prove contrapositive: $\sim$ii $\Rightarrow$ $\sim$i.

• Suppose that there is an augmenting path with respect to flow $f$.
• Can improve $f$ by sending flow along this path.
• Thus, $f$ is not a maxflow. □
Maxflow–mincut theorem


Augmenting path theorem. A flow \( f \) is a maxflow if and only if no augmenting paths.

Pf. For any flow \( f \), the following three conditions are equivalent:

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

[ iii \Rightarrow i ]

• Let \((A, B)\) be a cut whose capacity equals the value of flow \( f \).
• Then, the value of any flow \( f' \) \( \leq \) capacity of \((A, B)\) = value of \( f \).
• Thus, \( f \) is a maxflow.
Let $f$ be a flow with no augmenting paths.

Let $A$ be set of vertices reachable from $s$ via a path with no full forward or empty backward edges.

By definition of cut $(A, B)$, $s$ is in $A$.

By definition of cut $(A, B)$ and flow $f$, $t$ is in $B$.

Capacity of cut $(A, B) = \text{net flow across cut} = \text{value of flow } f$.

Maxflow–mincut theorem
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 \(A = \text{set of vertices connected to } s\) by an undirected path with no full forward or empty backward edges.
- Capacity of cut \((A, B) = \text{value of flow } f \Rightarrow \text{mincut.}\)
Maxflow: quiz 5

Given the following maxflow, which is a mincut?

A. \(A = \{A, F\}\).

B. \(A = \{A, B, C, F\}\).

C. \(A = \{A, B, C, E, F\}\).

D. None of the above.
6.4 Maximum Flow

- introduction
- Ford–Fulkerson algorithm
- maxflow–mincut theorem
- analysis of running time
- Java implementation
- applications
Important special case. Edge capacities are integers between 1 and $U$.

**Invariant.** The flow is integral throughout Ford–Fulkerson.

**Pf.**
- 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 of the flow by at least one. □

**Integrality theorem.** There exists an integral maxflow.

**Pf.**
- Proposition + Augmenting path theorem $\Rightarrow$ Ford–Fulkerson terminates with a maxflow.
- Invariant $\Rightarrow$ That maxflow is integral. □
Bad news. Number of augmenting paths can be very large. 

even when capacities are integral

initialize with 0 flow

initialize with 0 flow
How to choose augmenting paths?

Bad news. Some choices lead to exponential-time algorithms.

Good news. Clever choices lead to polynomial-time algorithms.

<table>
<thead>
<tr>
<th>augmenting path</th>
<th>number of paths</th>
<th>implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFS path</td>
<td>$\leq EU$</td>
<td>stack</td>
</tr>
<tr>
<td>random path</td>
<td>$\leq EU$</td>
<td>randomized queue</td>
</tr>
<tr>
<td>shortest path (fewest edges)</td>
<td>$\leq \frac{1}{2} EV$</td>
<td>queue</td>
</tr>
<tr>
<td>fattest path (max bottleneck capacity)</td>
<td>$\leq E \ln(EU)$</td>
<td>priority queue</td>
</tr>
</tbody>
</table>

Flow network 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
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

**Problem.** Given $n$ people and $n$ tasks, assign the tasks to people so that:

- Every task is assigned to a **qualified** person.
- Every person is assigned to exactly one task.
Problem. Given a bipartite graph, find a perfect matching (if one exists).

Bipartite graph

Person 5’ is qualified to perform tasks 4 and 5

perfect matching

1-4’
2-1’
3-3’
4-5’
5-2’
Maxflow formulation of bipartite matching

- Create source $s$, target $t$, one vertex $i$ for each task, and one vertex $j'$ for each person.
- Add edge from $s$ to each task $i$ (of capacity 1).
- Add edge from each person $j'$ to $t$ (of capacity 1).
- Add edge from task $i$ to qualified person $j'$ (of capacity 1 or $\infty$).

**flow network**

interpretation: flow on edge $4 \rightarrow 5' = 1$ means assign task 4 to person 5'
Maxflow formulation of bipartite matching

1–1 correspondence between perfect matchings in bipartite graph and \textit{integral flows} of value $n$ in flow network.

Integrality theorem + 1–1 correspondence $\Rightarrow$ Maxflow formulation is correct.
In the worst case, how many augmenting paths does the Ford-Fulkerson algorithm consider in order to find a perfect matching in a bipartite graph with $n$ vertices per side?

A. $\Theta(n)$
B. $\Theta(n^2)$
C. $\Theta(n^3)$
D. $\Theta(n^4)$
# Maximum flow algorithms: theory highlights

<table>
<thead>
<tr>
<th>year</th>
<th>method</th>
<th>worst case</th>
<th>discovered by</th>
</tr>
</thead>
<tbody>
<tr>
<td>1955</td>
<td>augmenting paths</td>
<td>$O(E V U)$</td>
<td>Ford–Fulkerson</td>
</tr>
<tr>
<td>1970</td>
<td>shortest augmenting paths</td>
<td>$O(E V^2)$</td>
<td>Edmonds–Karp, Dinitz</td>
</tr>
<tr>
<td>1974</td>
<td>blocking flows</td>
<td>$O(V^3)$</td>
<td>Karzanov</td>
</tr>
<tr>
<td>1983</td>
<td>dynamic trees</td>
<td>$O(E V \log V)$</td>
<td>Sleator–Tarjan</td>
</tr>
<tr>
<td>1988</td>
<td>push–relabel</td>
<td>$O(E V \log (V^2 / E))$</td>
<td>Goldberg–Tarjan</td>
</tr>
<tr>
<td>1998</td>
<td>binary blocking flows</td>
<td>$O(E^{3/2} \log (V^2 / E) \log U)$</td>
<td>Goldberg–Rao</td>
</tr>
<tr>
<td>2013</td>
<td>compact networks</td>
<td>$O(E V)$</td>
<td>Orlin</td>
</tr>
<tr>
<td>2014</td>
<td>interior–point methods</td>
<td>$\tilde{O}(E V^{1/2} \log U)$</td>
<td>Lee–Sidford</td>
</tr>
<tr>
<td>2016</td>
<td>electrical flows</td>
<td>$\tilde{O}(E^{10/7} U^{1/7})$</td>
<td>Mądry</td>
</tr>
<tr>
<td>2022</td>
<td>min ratio cycles</td>
<td>$O(E^{1+\epsilon} \log^2 U)$</td>
<td>CKLPGS</td>
</tr>
<tr>
<td>20xx</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Max–flow algorithms with $E$ edges, $V$ vertices, and integer capacities between 1 and $U$
Maximum flow algorithms: practice

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

**Often best in practice.** Push-relabel method with gap relabeling.

**Computer vision.** Specialized algorithms for problems with special structure.

---

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

Boris V. Cherkassky¹ and Andrew V. Goldberg²

¹ Central Institute for Economics and Mathematics, Krasikova St. 32, 117418, Moscow, Russia
dcher@emi.ru

² Computer Science Department, Stanford University, Stanford, CA 94305, USA
goldberg@cs.stanford.edu

**Abstract.** We study efficient implementations of the push-relabel method for the maximum flow problem. The resulting codes are faster than the previous codes, and much faster on some problem families. The speedup is due to the combination of heuristics used in our implementations. 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

**Mincut problem.** Find a cut of minimum capacity.

**Maxflow problem.** Find a 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).