6.4 Maximum Flow

- introduction
- Ford–Fulkerson algorithm
- maxflow–mincut theorem
- analysis of running time
- Java implementation (see video)
- applications
6.4 Maximum Flow

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

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

![Graph Representation of the MinCut Problem]

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 $B$. 

---

![Diagram](image.png)

**capacity** = $10 + 8 + 16 = \boxed{34}$
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 \( st \)-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

Efficient Maximum Flow Algorithms by Andrew Goldberg and Bob Tarjan

https://vimeo.com/100774435
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 constraint: $0 \leq \text{edge's flow} \leq \text{edge's capacity}$.
- Local equilibrium: inflow = outflow at every vertex (except $s$ and $t$).
Maxflow problem

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

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

\[
\text{value} = 5 + 10 + 10 = 25
\]
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 $t$.

**Maximum st-flow (maxflow) problem.** Find a flow of maximum value.
Maxflow application (Tolstoi 1930s)

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

[Image: 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 \( t \).

**Min-cut 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.
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**

![Diagram of a network flow problem](image)

bottleneck capacity = 10

**Impact:**
Increases value of flow; maintains flow conservation

$0 + 10 = 10$
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 of a network flow problem showing a 2nd augmenting path with capacities and flows indicated on the edges.]
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).

**3rd augmenting path**

![Diagram showing the 3rd augmenting path with values and annotations indicating the impact of increasing the flow value while maintaining flow conservation.](image-url)
**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

![Diagram of the Ford–Fulkerson algorithm with a 4th augmenting path](image-url)
**Termination.** All paths from $s$ to $t$ are blocked by either a

- Full forward edge.
- Empty backward edge.

**no more augmenting paths**
Maxflow: quiz 2

Which is an augmenting path?

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

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

C. Both A and B.

D. Neither A nor B.
Maxflow: quiz 3

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

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.** 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** = \(5 + 10 + 10 = 25\)

**Value of flow** = 25
**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 \(A\).

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

\[
\text{net flow across cut} = (10 + 10 + 5 + 10 + 0 + 0) - (5 + 5 + 0 + 0) = 25
\]
What is the net flow across the st-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 $(A, B)$ 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 \( f \leq \) the capacity of the cut \((A, B)\).

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

Maxflow–mincut theorem. Value of the maxflow $= \text{capacity of mincut.}$

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. $f$ is a maxflow.

ii. There is no augmenting path with respect to $f$.

iii. There exists a cut whose capacity equals the value of the flow $f$.

[ $i \implies ii$ ] We prove contrapositive: $\neg ii \implies \neg i$.

- 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. •
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. \( f \) is a maxflow.

ii. There is no augmenting path with respect to \( f \).

iii. There exists a cut whose capacity equals the value of the flow \( f \).

\[ \text{[ iii } \Rightarrow \text{i ]} \]

• Let \((A, B)\) be a cut whose capacity equals the value of the flow \( f \).

• Then, the value of any flow \( f' \leq \) capacity of \((A, B)\) = value of \( f \).

• Thus, \( f \) is a maxflow. \( \blacksquare \)

\[ \text{weak duality} \quad \text{by assumption} \]
Maxflow–mincut theorem

\[[ ii \Rightarrow iii \]

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

**Diagram:**
- $G$: Graph
- $A$: Set of vertices reachable from $s$
- $B$: Set of vertices reachable from $t$
- $s$: Source
- $t$: Sink
- Flow from $A$ to $B$
- Flow from $B$ to $A$ (flow = 0)
- Forward edge from $A$ to $B$ (flow = capacity)
- Backward edge from $B$ to $A$
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.
- Capacity of cut \((A, B) = \) value of flow \(f \Rightarrow \) mincut.
Maxflow: quiz 5

Given the following maxflow, which is a mincut?

A. \( S = \{ A, F \} \).
B. \( S = \{ A, B, C, F \} \).
C. \( S = \{ A, B, C, E, F \} \).
D. None of the above.
6.4 Maximum Flow

- introduction
- Ford–Fulkerson algorithm
- maxflow–mincut theorem
- analysis of running time
- Java implementation
- applications
Ford–Fulkerson algorithm analysis (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.**
- Proposition + Augmenting path theorem $\Rightarrow$ FF terminates with a maxflow.
- Invariant $\Rightarrow$ That maxflow is integral. ▪
Bad case for Ford–Fulkerson

**Bad news.** Number of augmenting paths can be very large.

Even when capacities are integral

Diagram:
- Initialize with 0 flow
- Flow: 0
- Capacity: 100
Bad case for Ford–Fulkerson

*Bad news.* Number of augmenting paths can be very large.

1st augmenting path
Bad case for Ford–Fulkerson

**Bad news.** Number of augmenting paths can be very large.
Bad news. Number of augmenting paths can be very large.

3rd augmenting path
Bad case for Ford–Fulkerson

**Bad news.** Number of augmenting paths can be very large.
Bad case for Ford–Fulkerson

**Bad news.** Number of augmenting paths can be very large.
Bad case for Ford–Fulkerson

**Bad news.** Number of augmenting paths can be very large.
Bad case for Ford–Fulkerson

**Bad news.** Number of augmenting paths can be very large.
Bad news. Number of augmenting paths can be very large.

exponential in input size \( (V, E, \log U) \)
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>$\leq E \ U$</td>
<td>stack (DFS)</td>
</tr>
<tr>
<td>random path</td>
<td>$\leq E \ U$</td>
<td>randomized queue</td>
</tr>
<tr>
<td><strong>shortest path (fewest edges)</strong></td>
<td>$\leq \frac{1}{2} E \ V$</td>
<td>queue (BFS)</td>
</tr>
<tr>
<td><strong>fattest path (max bottleneck capacity)</strong></td>
<td>$\leq E \ln(E \ U)$</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.
Bipartite matching problem

Problem. Given a bipartite graph, find a perfect matching (if one exists).

![Bipartite graph and perfect matching](image)

- **Bipartite graph**
  - Nodes represent tasks and people.
  - Edges indicate compatibility between tasks and people.

- **Perfect matching**
  - Pairing tasks to people in a way that no task is unpaired.
  - Example pairings: 1-4', 2-1', 3-3', 4-5', 5-2'.

- **Person 5'** is qualified to perform tasks 4 and 5.
Maxflow formulation of bipartite matching

- Create source $s$, sink $t$, 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 integral flows of value $n$ in flow network.

Integrality theorem + 1–1 correspondence $\Rightarrow$ Maxflow formulation is correct.
How many augmentations does the Ford–Fulkerson algorithms make 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>$O(EVU)$</td>
<td>Ford–Fulkerson</td>
</tr>
<tr>
<td>1970</td>
<td>shortest augmenting paths</td>
<td>$O(EV^2)$</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>$O(EV \log V)$</td>
<td>Sleator–Tarjan</td>
</tr>
<tr>
<td>1988</td>
<td>push–relabel</td>
<td>$O(EV \log (V^2/E))$</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>$O(EV)$</td>
<td>Orlin</td>
</tr>
<tr>
<td>2014</td>
<td>interior–point methods</td>
<td>$\tilde{O}(E^{1/2} \log U)$</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>Madry</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.


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. 22, 117418, Moscow, Russia
cherv@vrii.msk.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 seem 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).

- value of flow = 28
- capacity of cut = 28