6.4 Maximum Flow

- introduction
- Ford-Fulkerson algorithm
- maxflow-mincut theorem
- analysis of running time
- Java implementation
- applications
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
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 \textbf{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 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 \( B \).

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

capacity = \(10 + 8 + 10 = 28\)
Mincut 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

capacity

![Graph diagram](attachment:graph.png)
**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 $t$).

![Diagram](image-url)
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 $t$.

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

\[
\text{value} = 5 + 10 + 10 = 25
\]
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 Example Diagram]

The value is $8 + 10 + 10 = 28$. 

---

12
Maxflow application (Tolstoi 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)
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 $t$.

**Min-cut problem.** Find a cut of minimum capacity.

**Maxflow problem.** Find a flow of maximum value.

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
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).

**1st augmenting path**

![Graph diagram showing the first augmenting path with bottleneck capacity = 10]
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).

2\textsuperscript{nd} 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).

[Diagram of network flow with annotations for 3rd augmenting path and backward edge (not empty).]

Flow values are indicated on each edge, showing how the flow is increased along the 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*
Idea: increase flow along augmenting paths

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

no more augmenting paths

full forward edge
empty backward edge
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?
6.4 Maximum Flow

- introduction
- Ford-Fulkerson algorithm
- maxflow-mincut theorem
- analysis of running time
- Java implementation
- applications
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
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](https://via.placeholder.com/150)

Net flow across cut = 10 + 5 + 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 \(A\).

\[
\text{net flow across cut} = (10 + 10 + 5 + 10 + 0 + 0) - (5 + 5 + 0 + 0) = 25
\]
**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 $s =$ inflow to $t =$ value of flow.
Relationship between flows and cuts

**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 $f$:
  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 $f$.

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

- 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 $(A, B) = \text{ value of } f$.
- Thus, $f$ is a maxflow.  

  \[\text{weak duality}  \quad \text{by assumption}\]
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 $f$:

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 $f$.

\[
\text{[ ii } \Rightarrow \text{ iii } ] \quad \text{We prove contrapositive: } \sim \text{iii } \Rightarrow \sim \text{ii}.
\]

- 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.
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 $f$:

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 $f$.

[ iii ⇒ 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 $f$.  

*forward edges full; backward edges empty*

*flow-value lemma*
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.
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

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
  - requires clever analysis (or augmenting paths are chosen carefully)
Ford-Fulkerson algorithm with integer capacities

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

flow on each edge is an integer

**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.

critical for some applications (stay tuned)

**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.
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.
**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.
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.

199th 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.

200\textsuperscript{th} 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.

*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.

<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
How to choose augmenting paths?

Choose augmenting paths with:

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

Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems

JACK EDMONDS
University of Waterloo, Waterloo, Ontario, Canada

AND

RICHARD M. KARP
University of California, Berkeley, California

Abstract. This paper presents new algorithms for the maximum flow problem, the Hitchcock transportation problem, and the general minimum-cost flow problem. Upper bounds on the numbers of steps in these algorithms are derived, and are shown to compare favorably with upper bounds on the numbers of steps required by earlier algorithms.

Edmonds-Karp 1972 (USA)

Dinic 1970 (Soviet Union)
6.4 Maximum Flow

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

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

Original network

Residual network

Key point. Augmenting paths in original network are in 1-1 correspondence with directed paths in residual network.
Flow edge API

public class FlowEdge

FlowEdge(int v, int w, double capacity)  // create a flow edge v→w
    int from()  // vertex this edge points from
    int to()  // vertex this edge points to
    int other(int v)  // other endpoint
    double capacity()  // capacity of this edge
    double flow()  // flow in this edge
    double residualCapacityTo(int v)  // residual capacity toward v
    void addResidualFlowTo(int v, double delta)  // add delta flow toward v

flow $f_e$  capacity $c_e$

residual capacity
forward edge
backward edge
Flow edge: Java implementation

```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) { ... }
    public void addResidualFlowTo(int vertex, double delta) { ... }
}
```
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();
}

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

- forward edge
- backward edge
- residual capacity
- forward edge
- backward edge

**Graph:**
- Vertex `V` with flow `f_e = 7` and capacity `c_e = 9` to `W`.
- Vertex `V` to `W` with residual capacity `7` and forward edge.
- Vertex `W` with forward and backward edges.

**Flow Tree:**
- Flow directed from `V` to `W` with residual capacity `7`.
- Forward edge from `V` to `W`.
- Backward edge from `W` to `V`.

**Flow Process:**
- Calculate residual capacity for vertex `W`.
- Add residual flow to vertex `V` with `delta`.
- Update flow `flow` accordingly.

**Graph Diagram:**
- Directed edges with `V` to `W` and `W` to `V`.
- Residual capacity annotations for `V` to `W`.
- Forward and backward edge labels.

**Implementation Details:**
- Method `residualCapacityTo` calculates residual capacity.
- Method `addResidualFlowTo` adds residual flow to vertex.

**Algorithm:**
- Flow network algorithms.
- Graph theory.
- Java implementation for residual flow calculations.
Flow network API

public class FlowNetwork

FlowNetwork(int V)  create an empty flow network with V vertices
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 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).

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>();
    queue.enqueue(s);
    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

```
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. */
    }

    public double value() {
        return value;
    }

    public boolean inCut(int v) {
        return marked[v];
    }
}
```
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.
N students apply for N jobs.

Each gets several offers.

Is there a way to match all students to jobs?
Bipartite matching problem

Given a bipartite graph, find a perfect matching.

---

**perfect matching (solution)**

- Alice → Google
- Bob → Adobe
- Carol → Facebook
- Dave → Yahoo
- Eliza → Amazon

---

**bipartite graph**

---

**bipartite matching problem**

1. Alice → Adobe
   Alice → Amazon
   Alice → Google
   Alice → Yahoo
   Alice → Amazon
   Alice → Facebook
   Alice → Google
   Alice → Yahoo
   Alice → Amazon
   Alice → Facebook

---

6. Adobe → Alice
   Adobe → Bob
   Adobe → Carol
   Adobe → Dave
   Adobe → Eliza
   Adobe → Carol
   Adobe → Dave
   Adobe → Eliza
   Adobe → Carol
   Adobe → Dave

---

7. Amazon → Alice
   Amazon → Bob
   Amazon → Carol
   Amazon → Dave
   Amazon → Eliza
   Amazon → Carol
   Amazon → Dave
   Amazon → Eliza
   Amazon → Carol
   Amazon → Dave

---

8. Facebook → Dave
   Facebook → Carol
   Facebook → Dave
   Facebook → Carol
   Facebook → Dave
   Facebook → Carol
   Facebook → Dave
   Facebook → Carol
   Facebook → Dave
   Facebook → Carol

---

9. Google → Alice
   Google → Bob
   Google → Carol
   Google → Alice
   Google → Carol
   Google → Alice
   Google → Carol
   Google → Alice
   Google → Carol
   Google → Alice

---

10. Yahoo → Dave
    Yahoo → Eliza
    Yahoo → Dave
    Yahoo → Eliza
    Yahoo → Dave
    Yahoo → Eliza
    Yahoo → Dave
    Yahoo → Eliza
    Yahoo → Dave
    Yahoo → Eliza
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).
Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value $N$. 

Network flow formulation of bipartite matching

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.

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

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

\[ |S| > |T| \]
What the mincut tells us

**Mincut.** Consider mincut \((A, B)\).
- Let \(S = \) students on \(s\) side of cut.
- Let \(T = \) companies on \(s\) side of cut.
- Fact: \(|S| > |T|\); students 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>MON</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>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.
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>–</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>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>

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>–</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>

**AL East (August 30, 1996)**

**Detroit is mathematically eliminated.**

- Detroit finishes with \( \leq 76 \) 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.
**Baseball elimination problem: maxflow formulation**

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

**Fact.** Team 4 not eliminated iff all edges pointing from $s$ are full in maxflow.
### 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>Cherkasky</td>
</tr>
<tr>
<td>1978</td>
<td>blocking flow</td>
<td>$E^{7/3}$</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>
<tr>
<td>?</td>
<td>?</td>
<td>$E$</td>
<td>?</td>
</tr>
</tbody>
</table>

Maxflow algorithms for sparse digraphs with $E$ edges, 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.

**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\(^1\) and Andrew V. Goldberg\(^2\)

\(^1\) Central Institute for Economics and Mathematics, Krasikova St. 32, 117418, Moscow, Russia  
cher@ceml.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 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!