Max Flow, Min Cut

Contents.
- Max flow.
- Min cut.
- Ford-Fulkerson augmenting path algorithm.
- Shortest augmenting path.
- Fattest augmenting path.
- Bipartite matching.

Maximum Flow and Minimum Cut

Max flow and min cut.
- Two very rich algorithmic problems.
- Cornerstone problems in combinatorial optimization.
- Beautiful mathematical duality.

Nontrivial applications / reductions.
- Network connectivity.
- Bipartite matching.
- Data mining.
- Open-pit mining.
- Airline scheduling.
- Image processing.
- Project selection.
- Baseball elimination.

Network reliability.
- Security of statistical data.
- Distributed computing.
- Egalitarian stable matching.
- Distributed computing.
- Many many more . . .

Soviet Rail Network, 1955


Minimum Cut Problem

Network: abstraction for material FLOWING through the edges.
- Directed graph.
- Capacities on edges.
- Source node s, sink node t.

Min cut problem. Delete edges to disconnect s from t.
Cuts

A cut is a node partition \((S, T)\) such that \(s\) is in \(S\) and \(t\) is in \(T\).

- \(\text{capacity}(S, T) = \text{sum of weights of edges leaving } S\).

Minimum \(s\)-\(t\) cut problem. Find an \(s\)-\(t\) cut of minimum capacity.

Minimum \(s\)-\(t\) cut problem. Find an \(s\)-\(t\) cut of minimum capacity.

Maximum Flow Problem

Network: abstraction for material FLOWING through the edges.

- Directed graph.
- Capacities on edges.
- Source node \(s\), sink node \(t\).

Max flow problem. Assign flow to edges so that:

- Equalizes inflow and outflow at every intermediate vertex.
- Maximizes flow sent from \(s\) to \(t\).

- Exactly as for min cut problem

- Not \(s\) or \(t\)
Flows

A flow $f$ is an assignment of weights to edges so that:

- Capacity: $0 \leq f(e) \leq u(e)$.
- Flow conservation: flow leaving $v =$ flow entering $v$.

Max flow problem: find flow that maximizes net flow into sink.

Flows and Cuts

Observation 1. Let $f$ be a flow, and let $(S, T)$ be any cut. Then, the net flow sent across the cut is equal to the amount reaching $t$. 
Observation 1. Let $f$ be a flow, and let $(S, T)$ be any cut. Then, the net flow sent across the cut is equal to the amount reaching $t$.

Observation 2. Let $f$ be a flow, and let $(S, T)$ be any cut. Then the value of the flow is at most the capacity of the cut.

Cut capacity = 30 $\Rightarrow$ Flow value $\leq$ 30

Observation 3. Let $f$ be a flow, and let $(S, T)$ be a cut whose capacity equals the value of $f$. Then $f$ is a max flow and $(S, T)$ is a min cut.

Cut capacity = 28 $\Rightarrow$ Flow value $\leq$ 28

Flow value = 28
Max-Flow Min-Cut Theorem

MAX-FLOW MIN-CUT THEOREM (Ford-Fulkerson, 1956): In any network, the value of max flow is equal to the capacity of min cut.

- Proof IOU: we find flow and cut such that Observation 3 applies.

\[
\text{Min cut capacity } = 28 \iff \text{Max flow value } = 28
\]

Towards an Algorithm

Find s-t path where each arc has \( f(e) < u(e) \) and "augment" flow along it.

- Greedy algorithm: repeat until you get stuck.

Flow value = 10

Towards an Algorithm

Find s-t path where each arc has \( f(e) < u(e) \) and "augment" flow along it.

- Greedy algorithm: repeat until you get stuck.
- Fails: need to be able to "backtrack".

Flow value = 10

Flow value = 14
**Residual Graph**

Original graph.
- Flow $f(e)$.
- Edge $e = v \to w$

Residual arc.
- Edge $e = v \to w$ or $w \to v$.
- "Undo" flow sent.

Residual graph.
- All the edges that have strictly positive residual capacity.

**Augmenting Paths**

Augmenting path = path in residual graph.
- If augmenting path, then not yet a max flow.
- If no augmenting path, is it a max flow???

Ford-Fulkerson Augmenting Path Algorithm

Ford-Fulkerson algorithm.
- Generic method for solving max flow problem.

Start with $f(e) = 0$ everywhere.

**REPEAT** (until no augmenting paths are left)
Increase the flow along any augmenting path.

Questions.
- Does this lead to a maximum flow?
- How do we find an augmenting path? $s$-$t$ path in residual graph
- How many augmenting paths does it take?
Max-Flow Min-Cut Theorem

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

Max-flow min-cut theorem. The value of the max flow is equal to the capacity of the min cut.

We prove both simultaneously by showing the following are equivalent:

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

(ii) \(\Rightarrow\) (i) equivalent to not (i) \(\Rightarrow\) not (ii)

Let \( f \) be a flow. If there exists an augmenting path, then we can improve \( f \) by sending flow along path.

(iii) \(\Rightarrow\) (ii) Next slide.

(iii) \(\Rightarrow\) (i) This was Observation 3.

Proof of Max-Flow Min-Cut Theorem

(ii) \(\Rightarrow\) (iii). If there is no augmenting path relative to \( f \), then there exists a cut whose capacity equals the value of \( f \).

Proof.

- Let \( f \) be a flow with no augmenting paths.
- Let \( S \) be set of vertices reachable from \( s \) in residual graph.
  - \( S \) contains \( s \); since no augmenting paths, \( S \) does not contain \( t \)
  - all edges \( e \) leaving \( S \) in original network have \( f(e) = u(e) \)
  - all edges \( e \) entering \( S \) in original network have \( f(e) = 0 \)

\[
|f| = \sum_{e \text{ out of } S} f(e) - \sum_{e \text{ in to } S} f(e) \\
= \sum_{e \text{ out of } S} u(e) \\
= \text{capacity}(S, T)
\]

Ford-Fulkerson Algorithm: Implementation

Two representations of each edge in residual graph.

- May need to traverse edge in forward or reverse direction.
- Let \( e \) be edge in original network with flow \( f(e) \) and capacity \( u(e) \).
- In residual graph, include reverse edge and maintain anti-symmetry:
  - forward edge: flow = \( f(e) \), residual capacity = \( u(e) - f(e) \)
  - reverse edge: flow = \(-f(e)\), residual capacity = \(-u(e)\)

![Diagram](image)

```c
typedef struct node *link;
struct node {
  int w;       // target vertex w in v-w
  int cap;     // capacity u(e)
  link dup;    // reversal of edge
  link next;   // next edge in adjacency list
};
```

Ford-Fulkerson Algorithm: Implementation

Two representations of each edge.

- Need to update BOTH representations.
- Maintain link connecting forward and backward representations.

```c
typedef struct node *link;
struct node {
  int w;       // target vertex w in v-w
  int cap;     // capacity u(e)
  link dup;    // reversal of edge
  link next;   // next edge in adjacency list
};
```
Ford-Fulkerson Algorithm: Implementation

Compute Max Flow

```c
int residualCapacity(link e) {
    if (e->cap < 0) return -e->flow;  // reverse edge
    else return e->cap - e->flow;      // forward edge
}

void GRAPHmaxflow(Graph G, int s, int t) {
    int v, bottle;
    link e;
    find augmenting path
    while (augpath(G, s, t) == TRUE) {
        bottle = INFINITY;
        for (v = t; v != s; v = e->dup->v) {
            e = G->path[v];
            bottle = min(bottle, residualCapacity(e));
        }
        for (v = t; v != s; v = e->dup->v) {
            e = G->path[v];
            e->flow += bottle;  // update flow on forward and reverse edges
            e->dup->flow -= bottle;
        }
    }
}
```

Ford-Fulkerson Algorithm: Analysis

Assumption: all capacities are integers between 1 and U.

Invariant: every flow value and every residual capacities remain an integer throughout the algorithm.

Theorem: the algorithm terminates in at most \( |f^*| \leq VU \) iterations.

Corollary: if \( U = 1 \), then algorithm runs in \( O(V) \) iterations.

Integrality theorem: if all arc capacities are integers, then there exists a max flow \( f \) for which every flow value is an integer.

Choosing Good Augmenting Paths

Use care when selecting augmenting paths.

Original Network

Choosing Good Augmenting Paths

Use care when selecting augmenting paths.

Original Network
Choosing Good Augmenting Paths

Use care when selecting augmenting paths.
- Some choices lead to exponential algorithms.
- Clever choices lead to polynomial algorithms.
- Optimal choices for real world problems ???

Design goal is to choose augmenting paths so that:
- Can find augmenting paths efficiently.
- Few iterations.

Choose augmenting path with: Edmonds-Karp (1972)
- Fewest number of arcs. (shortest path)
- Max bottleneck capacity. (fattest path)

Shortest Augmenting Path

Shortest augmenting path.
- Easy to implement with BFS.
- Finds augmenting path with fewest number of arcs.

```c
while (!QUEUEisEmpty()) {
    v = QUEUEget();
    for (e = G->adj[v]; e != NULL; e = e->next) {
        if (residualCapacity(e) > 0) {
            w = e->w;
            if (G->dist[w] > G->dist[v] + 1) {
                G->dist[w] = G->dist[v] + 1;
                G->path[w] = e;
                QUEUEput(w);
            }
        }
    }
}
return (dist[t] < INFINITY);
```

Fattest Augmenting Path

Fattest augmenting path.
- Finds augmenting path whose bottleneck capacity is maximum.
- Delivers most amount of flow to sink.
- Solve using Dijkstra-style (PFS) algorithm.

```c
if (wt[w] < min(wt[v], cap[e]))
    wt[w] = min(wt[v], cap[e])
    Relax an edge e = v-w
```

Analysis:
- O(E log V) per augmentation with binary heap.
- O(E + V log V) per augmentation with Fibonacci heap.
- Fact: O(E log U) augmentations if capacities are between 1 and U.

Shortest Augmenting Path Analysis

Length of shortest augmenting path increases monotonically.
- Strictly increases after at most E augmentations.
- At most E V total augmenting paths.
- O(E^2 V) running time.
Choosing an Augmenting Path

Choosing an augmenting path.
- Any path will do => wide latitude in implementing Ford-Fulkerson.
- Generic priority first search.
- Some choices lead to good worst-case performance.
  - shortest augmenting path
  - fastest augmenting path
  - variation on a theme: PFS
- Average case not well understood.

Research challenges.
- Practice: solve max flow problems on real networks in linear time.
- Theory: prove it for worst-case networks.

An Application

Jon placement.
- Companies make job offers.
- Students have job choices.

Can we fill every job?
Can we employ every student?

Bipartite Matching

Bipartite matching.
- Input: undirected, bipartite graph G.
- A set of edges M is a matching if each vertex appears at most once.
- Max matching: find a max cardinality matching.

\[
\begin{align*}
1 & \rightarrow A \\
2 & \rightarrow B \\
3 & \rightarrow C \\
4 & \rightarrow D \\
5 & \rightarrow E \\
L & \rightarrow 1 \\
B & \rightarrow 2 \\
C & \rightarrow 3 \\
D & \rightarrow 4 \\
E & \rightarrow 5 \\
\end{align*}
\]

Matching
1-B, 3-A, 4-E

<table>
<thead>
<tr>
<th>Year</th>
<th>Discoverer</th>
<th>Method</th>
<th>Big-Oh</th>
</tr>
</thead>
<tbody>
<tr>
<td>1951</td>
<td>Dantzig</td>
<td>Simplex</td>
<td>(E V U \uparrow)</td>
</tr>
<tr>
<td>1955</td>
<td>Ford, Fulkerson</td>
<td>Augmenting path</td>
<td>(E V U \uparrow)</td>
</tr>
<tr>
<td>1970</td>
<td>Edmonds-Karp</td>
<td>Shortest path</td>
<td>(V^2)</td>
</tr>
<tr>
<td>1970</td>
<td>Edmonds-Karp</td>
<td>Max capacity</td>
<td>(E \log U (E + V \log V) \uparrow)</td>
</tr>
<tr>
<td>1970</td>
<td>Dinitz</td>
<td>Improved shortest path</td>
<td>(E V^2)</td>
</tr>
<tr>
<td>1972</td>
<td>Edmonds-Karp, Dinitz</td>
<td>Capacity scaling</td>
<td>(E^2 \log V \uparrow)</td>
</tr>
<tr>
<td>1973</td>
<td>Dinitz-Gabow</td>
<td>Improved capacity scaling</td>
<td>(E V \log U \uparrow)</td>
</tr>
<tr>
<td>1974</td>
<td>Karzanov</td>
<td>Preflow-push</td>
<td>(V^3)</td>
</tr>
<tr>
<td>1983</td>
<td>Sleator-Tarjan</td>
<td>Dynamic trees</td>
<td>(E V \log V)</td>
</tr>
<tr>
<td>1986</td>
<td>Goldberg-Tarjan</td>
<td>FIFO preflow-push</td>
<td>(E V \log (V^2 / E))</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(E^2 V \log (V^2 / E) \log U \uparrow)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(E^2 V^2 \log (V^2 / E) \log U \uparrow)</td>
</tr>
</tbody>
</table>

\(\uparrow\) Arc capacities are between 1 and U.
**Bipartite Matching**

Bipartite matching.
- Input: undirected, bipartite graph $G$.
- A set of edges $M$ is a matching if each vertex appears at most once.
- Max matching: find a max cardinality matching.

- **Matching**
  - 1-A, 2-B, 3-C, 4-D

**Bipartite Matching: Proof of Correctness**

Claim. Matching in $G$ of cardinality $k$ induces flow in $G'$ of value $k$.
- Given matching 1-2’ 3-1’ 4-5’ of cardinality 3.
- Consider flow that sends 1 unit along each of 3 paths: $s-1-2'-t, s-3-1'-t, s-4-5'-t$.
- $f$ is a flow, and has cardinality 3.

**Bipartite Matching: Proof of Correctness**

Claim. Flow $f$ of value $k$ in $G'$ induces matching of cardinality $k$ in $G$.
- By integrality theorem, there exists 0/1 valued flow $f$ of value $k$.
- Consider $M = \text{set of edges from } L \text{ to } R \text{ with } f(e) = 1$.
  - each node in $L$ and $R$ participates in at most one edge in $M$
  - $|M| = k$: consider cut $(L \cup s, R \cup t)$
Reduction

Reduction.
- Given an instance of bipartite matching.
- Transform it to a max flow problem.
- Solve max flow problem.
- Transform max flow solution to bipartite matching solution.

Issues.
- How expensive is transformation?
- Is it better to solve problem directly? $O(E \sqrt{V})$ bipartite matching

Bottom line: max flow is an extremely rich problem-solving model.
- Many important practical problems reduce to max flow.
- We know good algorithms for solving max flow problems.