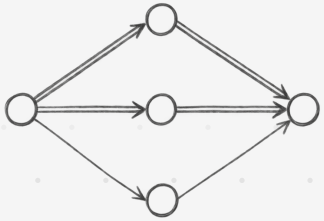
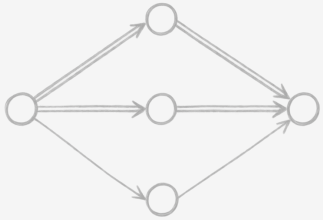


Algorithms ▸ Network Flows

Lecture 3

- MaxFlow Review
- Faster Augmenting Paths: Dinitz-Edmonds-Karp
- Modeling Problems with Maxflow



Algorithms ▸ Network Flows

Lecture 3

- MaxFlow Review
- Faster Augmenting Paths: Dinitz-Edmonds-Karp
- Modeling Problems with Maxflow

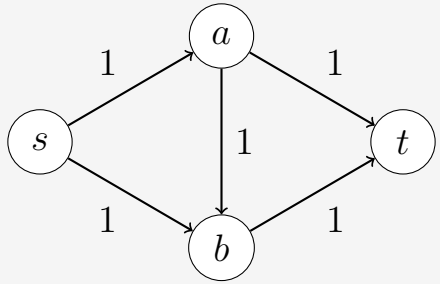
The Maximum Flow Problem

Problem

A flow network is a directed graph with edge **capacities** $c(u, v) \geq 0$, a **source** s, and a **sink** t. A flow is $f : E \to \mathbb{R}$ satisfying

$$\text{(conservation)} \ \sum_u f(u, v) = \sum_u f(v, u) \ \ \forall v \notin \{s, t\}, \qquad \text{(capacity)} \ 0 \leq f(u, v) \leq c(u, v).$$

The **value** of f is the net flow leaving s. Find a flow of maximum value.

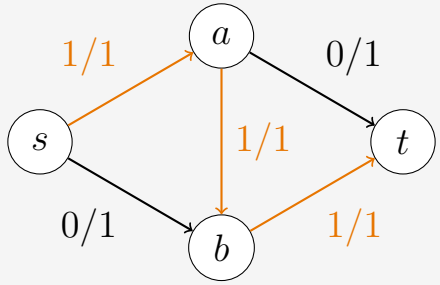


Lemma:

The net flow leaving s equals the net flow entering t.

A Feasible Flow

Every edge is labeled f/c: the flow currently sent, and its capacity.

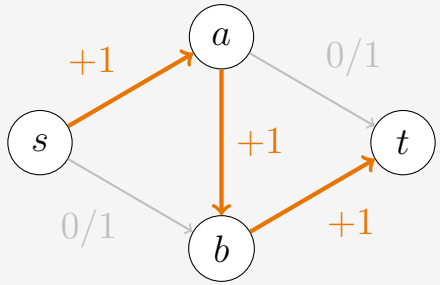


Conservation holds at a (1 in from s, 1 out to b) and at b (1 in from a, 1 out to t): this is a feasible flow, of value 1.

Increasing Flow Along a Path

Pick any s-t path P made of **unsaturated** edges, and any Δ at most the spare capacity of every edge on P. Adding Δ to f along P keeps every constraint satisfied: each internal vertex of P gains one unit of in-flow and one of out-flow, so conservation is untouched, and capacities hold by choice of Δ.

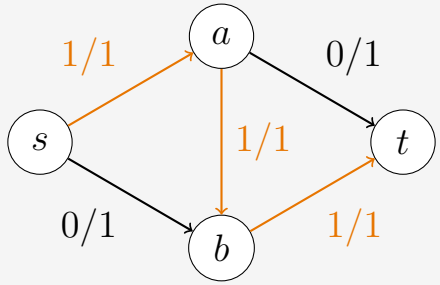
Sending $\Delta = 1$ along $P = s \to a \to b \to t$ from the all-zero flow:



Three forward edges, each raised by $\Delta = 1$: the same feasible flow from the previous slide.

Why Greedy Augmenting Paths Can Fail

Building a flow by repeatedly sending flow along any s-t path with spare capacity, until none remains, looks like a natural greedy algorithm:



No forward-only s-t path has spare capacity left: $s \to a$ and $b \to t$ are saturated, and the only spare edge $s \to b$ dead-ends at b (its only outgoing edge $b \to t$ is full). Yet the maximum flow is 2.

Remark

Sending flow along $a \to b$ was a mistake. To recover, we need a way to *undo* flow we already committed.

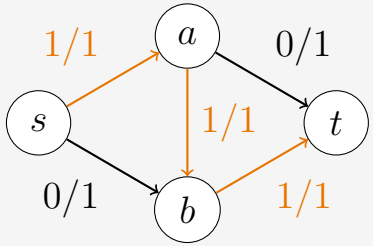
The Residual Graph

Definition (residual capacity): For edge (u, v) with capacity $c(u, v)$ and flow $f(u, v)$,

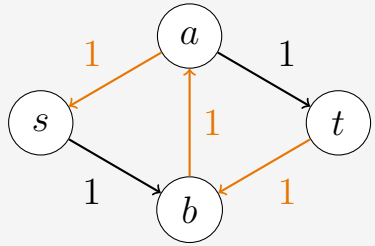
$$c_f(u, v) = \begin{cases} c(u, v) - f(u, v), & (u, v) \in E, \\ f(v, u), & (v, u) \in E. \end{cases}$$

Definition (residual network): G_f has an edge (u, v) whenever $c_f(u, v) > 0$.

Sending flow along a *reverse* edge (v, u) cancels flow already sent along (u, v): the "undo" operation we needed.



Flow f

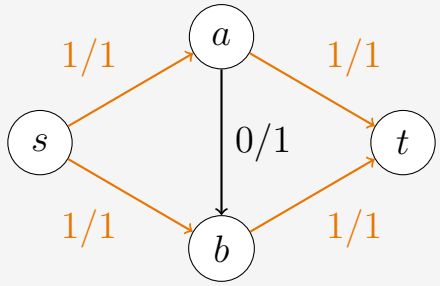


Residual graph G_f

Augmenting Paths and the Fix

Definition (augmenting path): An s-t path of positive residual capacity on every edge.

The path $s \to b \to a \to t$ is augmenting in G_f: it uses the fresh edge $s \to b$, the reverse edge $b \to a$ (residual capacity 1), and the fresh edge $a \to t$.



Sending 1 unit along it cancels the $a \to b$ flow and reroutes it

The Ford-Fulkerson Algorithm

Algorithm: Ford-Fulkerson

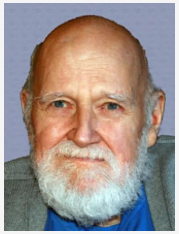
Initialize $f(u, v) \leftarrow 0$ on every edge.

1. While G_f has an s-t path (find one with BFS or DFS):
   - 1.1 Let P be such a path and $b = \min_{(u,v) \in P} c_f(u, v)$ its bottleneck residual capacity.
   - 1.2 Increase f by b along every forward edge of P, and decrease f by b along every edge of P that is a reversal.
2. Return f.

Remark

Any choice of augmenting path is allowed — correctness does not depend on which one we pick, only the running time does (more on this soon).

Ford and Fulkerson (1956)



Les Ford

Delbert Fulkerson

MAXIMAL FLOW THROUGH A NETWORK

L. R. FORD, JR. AND D. R. FULKERSON

the original paper

Remark

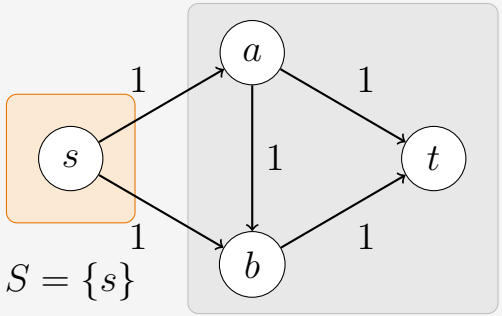
We still need to prove it always returns a maximum flow.

It helps to think again about *cuts*: a quantity we can compute that upper-bounds every flow, and that a correct algorithm should match.

Certifying Optimality: s-t Cuts

Definition (s-t cut): A partition (S, T) of the vertices with $s \in S$ and $t \in T$.
A cut can be thought of as a separator: a set of edges that disconnects s from t

Definition (capacity): $\text{cap}(S, T) = \sum_{u \in S} \sum_{v \in T} c(u, v)$.



This cut has capacity $c(s, a) + c(s, b) = 1 + 1 = 2$. **Every unit of flow leaving s must cross this cut**, so no flow can exceed value 2.

Net Flow Across a Cut

Definition (net flow): The net flow across a cut (S, T) is

$$f(S,T) = \sum_{u \in S} \sum_{v \in T} f(u,v) - \sum_{u \in S} \sum_{v \in T} f(v,u).$$

Lemma: Flow-value lemma

For any s-t cut (S, T), $f(S,T) = \text{value}(f)$.

Remark

See the Precept 2 notes for the proof

Weak Duality: Max-Flow $\leq$ Min-Cut

Theorem: For any flow f and any s-t cut (S, T), $\text{value}(f) = f(S, T) \leq \text{cap}(S, T)$.

Proof

$$f(S,T) = \sum_{u \in S, v \in T} f(u,v) - \sum_{u \in S, v \in T} f(v,u) \leq \sum_{u \in S, v \in T} f(u,v) \leq \sum_{u \in S, v \in T} c(u,v) = \text{cap}(S,T).$$

The first inequality drops a nonnegative term; the second uses the capacity constraint.

Corollary: $\text{value}(\text{any flow}) \leq \text{max-flow} \leq \text{min-cut} \leq \text{cap}(\text{any cut})$.

So a cut is a *certificate of optimality*: if we find a flow and a cut with equal value, we know the flow is maximum and the cut is minimum

Analysis: Correctness

Theorem: If all capacities are integers, Ford-Fulkerson terminates with a maximum flow.

Proof

When the algorithm halts, G_f has no s-t path. Let S be the vertices reachable from s in G_f, and $T = V \setminus S$; then (S, T) is a valid cut with $t \in T$.

Edges from S to T: take $(u, v) \in E$ with $u \in S, v \in T$. If $f(u, v) < c(u, v)$, then (u, v) is a forward residual edge, so v would be reachable from s — contradiction. So every such edge is saturated: $f(u, v) = c(u, v)$.

Edges from T to S: take $(v, u) \in E$ with $v \in T, u \in S$. If $f(v, u) > 0$, then the reverse edge (u, v) is a residual edge, so again v would be reachable from s — contradiction. So every such edge is empty: $f(v, u) = 0$.

Hence $f(S, T) = \text{cap}(S, T)$, so $\text{value}(f) = \text{cap}(S, T)$. By weak duality this is also an upper bound on every flow, so f is maximum (and (S, T) is a minimum cut).

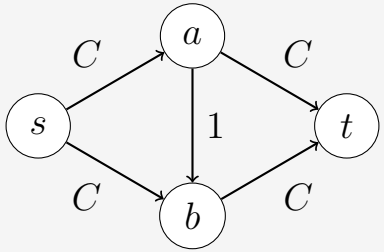
Analysis: Running Time

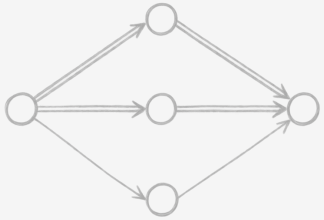
Theorem: If all capacities are integers, Ford-Fulkerson runs in $O(EF)$ time (writing $V = |V|, E = |E|$), where F is the value of the maximum flow.

Proof

Each iteration increases the flow value by an integer ≥ 1, so there are at most F iterations. Each iteration finds a path via BFS/DFS in $O(V + E) = O(E)$ time

This bound is tight. With C a large integer:





Algorithms ▸ Network Flows

Lecture 3

- MaxFlow Review
- Faster Augmenting Paths: Dinitz-Edmonds-Karp
- Modeling Problems with Maxflow

Choosing Better Augmenting Paths

The bad example used a *long, high-capacity* path whenever a *short, low-capacity* one was available.

Remark

What if we always chose a **shortest** augmenting path (fewest edges), found by BFS, instead of an arbitrary one (e.g., via DFS)?

On the bad example, BFS finds $s \to a \to t$ or $s \to b \to t$ first reaching the maximum flow in 2 iterations instead of $2C$.

The Dinitz-Edmonds-Karp Algorithm

Algorithm: Dinitz-Edmonds-Karp

Run Ford-Fulkerson, always choosing a **shortest** s-t path in G_f (by number of edges, found with BFS) as the augmenting path.

Theorem: Dinitz-Edmonds-Karp runs in $O(VE^2)$ time.

Notably, this bound has **no dependence on** F: the algorithm is strongly polynomial, since its running time does not depend on the numeric size of the input, only on V and E.

Dinitz, Edmonds, and Karp

Yefim Dinitz

Jack Edmonds

Richard Karp

Discovered independently: Dinitz (1970) in the Soviet Union, and Edmonds and Karp (1972) in the US/Canada

Analysis: Distances Never Decrease

Lemma:

Let $d_f(v)$ denote the distance from s to v in the residual graph G_f. Over the course of Dinitz-Edmonds-Karp, $d_f(t)$ never decreases.

Proof

- Let $s = v_1, v_2, \ldots, v_k = t$ be the shortest augmenting path used in one iteration, so $d_f(v_i) = d_f(v_{i-1}) + 1$ for every i: each vertex is exactly one farther from s than the last.
- Augmenting saturates at least one edge (v_{i-1}, v_i), which then **leaves** G_f. But if this edge goes from zero to nonzero flow, its reverse (v_i, v_{i-1}) may newly **appear** in G_f.
- **This new edge cannot shorten anyone's distance to s:** since $d_f(v_i) = d_f(v_{i-1}) + 1$, v_i is already farther from s than v_{i-1}, so a path routed through the new edge $v_i \to v_{i-1}$ can only be longer, never shorter.
- So **no vertex's distance from s can decrease** — in particular, $d_f(t)$ cannot.

Analysis: d Increases Every E Iterations

Lemma:

Every E iterations, $d_f(t)$ must increase by at least 1.

Proof

- Each iteration **saturates** at least one edge (u, v) on the augmenting path, which then leaves G_f.
- For (u, v) to become usable again, its reverse (v, u) must appear on some *future* shortest path.
- But while $d_f(v) = d_f(u) + 1$ still holds, (v, u) can never lie on a shortest path — that would need $d_f(u) = d_f(v) + 1$ instead, which by the previous lemma can only happen once distances have already grown.
- So as long as $d_f(t)$ hasn't changed, every iteration must saturate a **fresh** edge. With only E edges to spend, $d_f(t)$ must increase within E iterations.

Putting It Together: Running Time

- $d_f(t)$ is always between 0 and $V-1$, and by our first lemma it **never decreases**.
- By our second lemma, it must **increase at least once every E iterations**.
- So there are at most $O(V) \cdot O(E) = O(VE)$ iterations total, each costing $O(E)$ time: a single BFS in G_f.

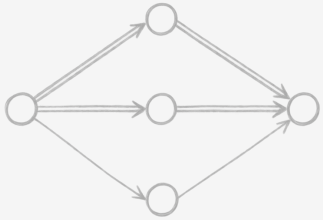
Theorem: Dinitz-Edmonds-Karp runs in $O(VE) \cdot O(E) = O(VE^2)$ time.

Modern Approaches to Maximum Flow

- **Push-relabel** (Goldberg–Tarjan, 1988): a different algorithmic paradigm, running in $O(V^2E)$ time.
- **Continuous optimization** [Christiano-Kelner-Madry-Spielman-Teng, . . .]: view max-flow as finding a point in the intersection of two convex sets.
- **Near-linear time** [Chen-Kyng-Liu-Peng-Probst Gutenberg-Sachdeva, 2022]: $O(E^{1+o(1)})$ time — a genuinely *near-linear* time algorithm!

Remark

The takeaway is that, seventy years after Ford and Fulkerson, maximum flow is still an active research area.



Algorithms ▸ Network Flows

Lecture 3

- MaxFlow Review
- Faster Augmenting Paths: Dinitz-Edmonds-Karp
- Modeling Problems with Maxflow

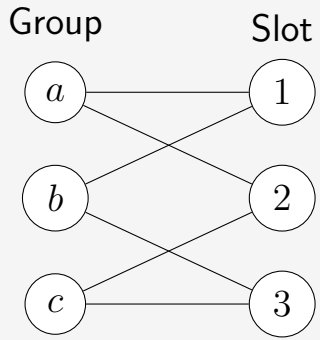
Quick note

We didn't cover this section in lecture, but I recommend reading the slides.

The Bipartite Matching Problem

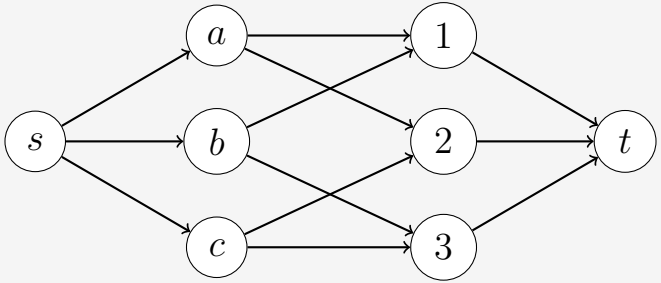
Problem

Given a bipartite graph $G = (L \cup R, E)$, find a largest possible matching: a set of edges with no endpoints in common.



Reducing Matching to Max-Flow

Build a flow network: add s connected to every group vertex, every slot vertex connected to t, and orient every bipartite edge from L to R. All capacities are 1.



Every edge has capacity 1, and every edge is oriented left to right.

Remark

Reduction claim: the maximum flow in this network equals the maximum matching in G. We must prove this in both directions.

Correctness of the Reduction

Proof: matching $\Rightarrow$ flow of the same value

Given a matching M, set $f(u, v) = 1$ for every matched edge $(u, v) \in M$, and $f(s, u) = f(v, t) = 1$ for every matched u, v; all other edges carry 0 flow. Since every vertex has at most one edge in M, every vertex has at most one unit of flow through it: capacities are respected. The value of f is $|M|$.

Proof: integral flow $\Rightarrow$ matching of the same value

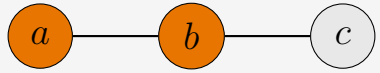
Ford-Fulkerson only ever adds integer amounts to f (each bottleneck is a minimum of integer residual capacities), so it returns a maximum flow f with every edge integral, hence 0 or 1 (capacities are 1). Let $M = \{(u, v) : f(u, v) = 1, u \in L, v \in R\}$. Conservation at each $u \in L$ (capacity 1 in from s) and each $v \in R$ (capacity 1 out to t) means u, v each appear in at most one edge of M: M is a matching, and $|M| = \text{value}(f)$.

Together: $\text{max-matching} \leq \text{max-flow}$ (first claim) and $\text{max-flow} \leq \text{max-matching}$ (second claim), so they're equal.

The Network Maintenance Problem

Problem

A network of cell towers has an edge between any two towers with overlapping coverage. Each tower is currently on Low-band or High-band. Auditing an edge costs B if its two towers share a band, or $C \geq B$ if they don't; reconfiguring a tower's band costs A. Choose a band for every tower (possibly its current one) minimizing the total audit and reconfiguration cost.



orange = Low-band, gray = High-band

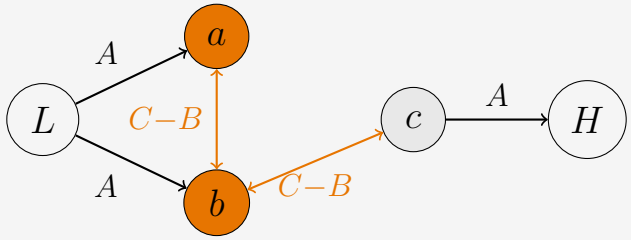
Remark

No source, no sink, no capacities to send anything along — nothing here looks like flow yet.

Reducing to a Min Cut

Build a graph with one node per tower, plus two terminals L and H:

- For every tower u currently on Low-band: an edge $L \to u$ of capacity A.
- For every tower u currently on High-band: an edge $u \to H$ of capacity A.
- For every pair of towers u, v with overlapping coverage: *both* edges $u \to v$ and $v \to u$, each of capacity $C - B$.



An L-H cut (S, T) (with $L \in S, H \in T$) proposes a configuration: put every tower in S on Low-band, every tower in T on High-band.

Correctness of the Reduction

Claim: The cost of the configuration proposed by (S, T) equals $B \cdot |E| + \mathrm{cap}(S, T)$.

Proof

Audit edges: a pair u, v on the same side costs B and crosses no directed copy of the edge; on opposite sides it costs C, and exactly one of the two directed copies $u \to v$, $v \to u$ crosses the cut, contributing $C - B$. Either way, the pair's cost is exactly B more than its contribution to $\mathrm{cap}(S, T)$.

Reconfiguration edges: take u originally Low-band. If $u \in S$ it keeps its band (cost 0) and $L \to u$ stays inside S (no crossing). If $u \in T$ it's reconfigured (cost A) and $L \to u$ crosses from S to T (contributing A). Symmetrically for u originally High-band. Either way, cost equals contribution to $\mathrm{cap}(S, T)$.

Since $B \cdot |E|$ doesn't depend on the configuration, minimizing cost is the same as minimizing $\mathrm{cap}(S, T)$: a **minimum** L-H cut gives an optimal configuration.

Solving It with Max-Flow

- Build the network above: $O(N + E)$ time.
- Run Dinitz-Edmonds-Karp to find a maximum L-H flow.
- Let S be the vertices reachable from L in the residual graph at termination — a minimum cut, by our correctness proof for Ford-Fulkerson.
- Reconfigure every tower that ends up on the opposite side from where it started.

Total running time: $O(VE^2) = O(N \cdot E^2)$, using Dinitz-Edmonds-Karp on this $O(N)$-vertex, $O(E)$-edge network.

Lessons: Modeling with Flows

Max-flow is a *powerful modeling tool*: many combinatorial problems reduce to it, inheriting a strongly-polynomial algorithm for free.

- A reduction is only useful once proven correct in **both** directions: a solution to one side must produce a solution of the same value on the other.
- Sometimes correctness needs the integrality theorem (matching); sometimes the cut itself already *is* the answer (towers).