CS1570 Lecture Notes

Claire Mathieu

Fall 2026

1 October 29: Network flows

The network flow problem consists of routing as much "stuff" as possible through a network whose links have limited capacities. The algorithmic idea is the same as for matchings, namely improving the current solution along an augmenting path. A duality theorem, max-flow min-cut, comes out of the analysis naturally, in the same way that König's theorem comes out of the matching algorithm.

1.1 Definitions and an example

Definition 1. A *flow network* is a directed graph $G = (V, E)$ with two distinguished vertices, a *source* s and a *sink* t, and a *capacity* $c(e) \geq 0$ on each edge e. A *flow* is a function $f : E \to \mathbb{R}$ such that

- (capacity) $0 \leq f(e) \leq c(e)$ for every edge e;
- (conservation) for every vertex $v \neq s, t$: $\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$.

The *value* of f is the net amount leaving the source: $|f| = \sum_{e \text{ out of } s} f(e) - \sum_{e \text{ into } s} f(e)$. The *maximum flow problem* asks for a flow of largest value.

For simplicity, we assume that no edge enters s.

Applications. Goods shipped through a road or rail network with limited capacities; data sent through a communication network with limited bandwidth; water or oil through pipes. Less obviously, many combinatorial problems reduce to maximum flow.

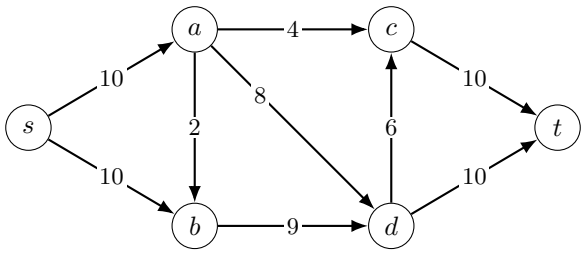


Figure 1: A flow network; numbers are capacities. Its maximum flow has value 19.

Example. In Figure 1, sending 4 units along $s\,a\,c\,t$, 6 along $s\,a\,d\,t$ and 4 along $s\,b\,d\,t$ gives a flow of value 14. Is that the best possible? Clearly not, since $s\,b\,d\,c\,t$ still has spare capacity. But how would we know when to stop? We want an algorithm, and a proof that its answer is optimal.

1.2 The Ford–Fulkerson algorithm

A first attempt, and why it fails. A greedy algorithm would start from the empty flow and gradually add flow. If we consider the problem on a per-edge basis, we can, indeed, add flow on an edge and make sure that the capacity constraint is preserved, but how about the flow conservation constraints? Thos are impossible to maintain if we act on a per-edge basis and hard to recover once we are out of balance. However, note that if a put flow on a path from s to t then we preserve those constraints, so then we must just make sure to not push too much, so that the capacities constraints remain satisfied on every edge along the path. Thus st-paths are the right "atomic" objects to consider for this problem.

The natural greedy algorithm repeatedly finds an s–t path whose edges all have spare capacity and pushes as much flow along it as possible, that is, the minimum spare capacity along the path. However, it can get stuck. In Figure 2, if the first path chosen is $s\,a\,b\,t$, no further path exists, yet the maximum flow has value 2.

To recover, the algorithm must be able to *undo* flow: push one unit back along ab. This is similar to the idea of flipping a matched edge along an alternating path, in the matching context.

Definition 2. Let f be a flow in G. The *residual graph* G_f has the same vertices and, for each edge $e = uv$ of G:

- a *forward* edge uv with residual capacity $c(e) - f(e)$, if $f(e) < c(e)$ (we can push more);
- a *backward* edge vu with residual capacity $f(e)$, if $f(e) > 0$ (we can cancel flow).

An *augmenting path* is a path from s to t in G_f. Its *bottleneck* is the smallest residual capacity on it.

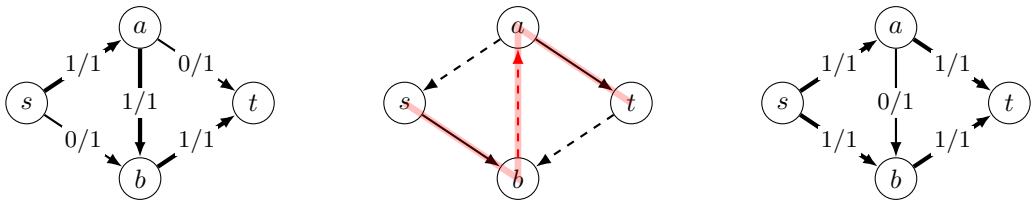


Figure 2: Labels are flow/capacity; all capacities are 1. Dashed edges in the residual graph are backward edges.

Lemma 1. *Let P be an augmenting path for f with bottleneck $\delta > 0$. Define f' by adding δ to $f(e)$ for every forward edge e of P, and subtracting δ from $f(e)$ for every edge e whose backward copy is on P. Then f' is a flow, and $|f'| = |f| + \delta$.*

Proof. The capacity constraints hold by the choice of δ. For conservation, consider an interior vertex v of P. Whether each of the two path edges at v is forward or backward, there are four cases. In each case, the change in inflow at v equals the change in outflow. For example, with a forward edge in and a backward edge out, the inflow increases by δ on one edge and decreases by δ on another. Finally, the first edge of P leaves s and is a forward edge (no edge enters s), so $|f|$ increases by δ. □

Algorithm (Ford–Fulkerson). Start with $f = 0$. While the residual graph of G for f, G_f, has an augmenting path P, augment f along P by its bottleneck.

We did not have time to discuss runtime (deferred to the next lecture).

When the algorithm stops, is the flow maximum?

1.3 Cuts and the max-flow min-cut theorem

Definition 3. An s–t *cut* is a partition (S, T) of V with $s \in S$ and $t \in T$. Its *capacity* is the total capacity of the edges going from S to T: $\mathrm{cap}(S, T) = \sum_{u \in S,\, v \in T,\, uv \in E} c(uv)$.

Intuitively, every unit of flow must cross from S to T at some point, so a cut is a bottleneck for all flows at once.

Lemma 2. *The net amount of a flow crossing a cut is the same for every st-cut. In other words, for every flow f and every s–t cut (S,T),*

$$|f| \;=\; \sum_{\substack{uv\in E\\ u\in S,\, v\in T}} f(uv) \;-\; \sum_{\substack{uv\in E\\ u\in T,\, v\in S}} f(uv) \;\le\; \operatorname{cap}(S,T).$$

Proof. (We did not have time to do this prof in class, so do it as an exercise when you work on the lecture materials.) Sum, over all $v \in S$, the outflow of v minus the inflow of v. By conservation, this sum is $|f|$, since only s contributes. An edge with both endpoints in S contributes $+f(e)$ once and $-f(e)$ once, so it cancels. What remains is the flow on edges from S to T minus the flow on edges from T to S. The first sum is at most $\operatorname{cap}(S,T)$ and the second is nonnegative. □

So *any* cut gives an upper bound on *every* flow: $\max_f |f| \le \min_{(S,T)} \operatorname{cap}(S,T)$. The following analysis of Ford–Fulkerson shows that the bound is tight at the same time as showing correctness of the Ford-Fulkerson algorithm.

Theorem 1 (Max-flow min-cut; Ford–Fulkerson 1956)**.** *For a flow f, the following are equivalent:*

1. *f is a maximum flow;*
2. *there is no augmenting path for f;*
3. *$|f| = \operatorname{cap}(S,T)$ for some s–t cut (S,T).*

In particular, the maximum value of a flow equals the minimum capacity of an s–t cut.

Proof. (1)⇒(2): if there were an augmenting path, Lemma 1 would give a larger flow.
(3)⇒(1): by Lemma 2, every flow g satisfies $|g| \le \operatorname{cap}(S,T) = |f|$.
(2)⇒(3): this is the heart of the matter, and the cut comes from the algorithm. Let S be the set of vertices reachable from s in G_f, and $T = V \setminus S$. Then $s \in S$, and $t \in T$ because there is no augmenting path. Consider an edge uv of G.

- If $u \in S$ and $v \in T$, then $f(uv) = c(uv)$. Otherwise the forward edge uv would be in G_f, and v would be reachable.
- If $u \in T$ and $v \in S$, then $f(uv) = 0$. Otherwise the backward edge vu would be in G_f, and u would be reachable.

By the equality in Lemma 2, $|f| = \operatorname{cap}(S,T) - 0$. □

Corollary 1 (Correctness)**.** *With integer capacities, Ford–Fulkerson terminates with a maximum flow, and with a minimum cut: the set of vertices reachable from s in the last residual graph.*

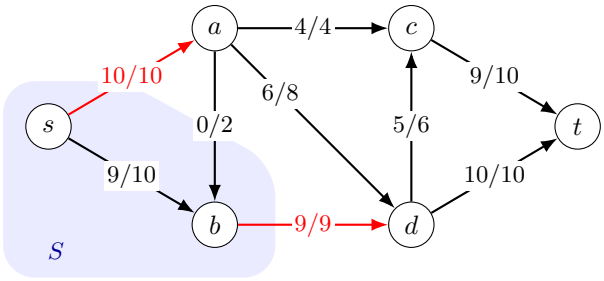


Figure 3: A maximum flow (flow/capacity) of value 19 for the network of Figure 1. In the residual graph, s can reach only b. The cut $S = \{s, b\}$ (shaded) has capacity $10 + 9 = 19$: its two outgoing edges (red) are saturated, and the edge ab entering S carries no flow. This cut certifies optimality.