Network Flow

**Definition:** A network is a directed graph \( N = (V, E) \) with

- a single source \( s \in V \) with no incoming edge,
- a single sink \( t \in V \) with no outgoing edge,
- a nonnegative integer capacity \( c(e) \) for each edge \( e \in E \).

**Network flow problem:** Assign flow \( f(e) \) to each edge \( e \) such that we have maximum flow in the network, subject to:

- **Capacity constraint:** for each edge \( e \), \( 0 \leq f(e) \leq c(e) \) (flow does not exceed capacity);
- **Conservation constraint:** for each vertex \( v \neq s, t : f_{in}(v) = f_{out}(v) \), where \( f_{in}(v) = \text{total flow into } v = \sum_{(u,v) \in E} f(u,v) \) and \( f_{out}(v) = \text{total flow out of } v = \sum_{(v,u) \in E} f(v,u) \);
- total flow in network is denoted by \( |f| \) and defined as \( |f| = f_{out}(s) \) (by conservation, \( |f| = f_{in}(t) \); this will be proved later).

**Previous approaches fail:**

Brute force? \( \Omega(\prod_{e \in E} c(e)) \) for integer flows – each edge \( e \) can get a flow of \( 0, 1, 2, ..., c(e) \), and we consider all possibilities independently of other edges – much worse than simple exponential!

Greedy? No way to select any part of flow greedily.

Dynamic programming? No way to break down problem into independent recursive sub-problems.

**An idea:** Local search strategy: start with initial assignment of flow guaranteed to be correct but not necessarily maximum, then try to make incremental improvements – stop when no improvement possible.

---

**Algorithm 1:** Ford-Fulkerson Algorithm

1. start with any valid flow \( f \) (e.g., \( f(e) = 0 \) for all \( e \in E \))
2. while there is an augmenting path \( P \) do
   3. augment \( f \) using \( P \)
3. return \( f \)

**Augmenting paths?**

**Intuition:** Since all flow must start at \( s \) and end at \( t \), find \( s-t \) paths along which flow can be increased. Instead of adding flow to edges in haphazard manner, this preserves conservation.

**First idea:** path \( P = s \to \cdots \to t \) where \( f(e) < c(e) \) for each \( e \). Define residual capacity \( \Delta f(e) = c(e) - f(e) \), and residual capacity \( \Delta f(P) = \text{MIN}_{e \in P} \Delta f(e) \). Augment path by adding \( \Delta f(P) \) to all edge flows.

**Problem:** notion too narrow, can get stuck with sub-optimal solution. (Example.)

**Second idea:** allow reverse edges on path and re-define residual capacity of \( e \):

- \( \Delta f(e) = c(e) - f(e) \) if \( e \) is an original edge on the path;
- \( \Delta f(e) = f(e) \) if \( e \) is a reverse edge on the path.
Intuition: original edge has unused capacity that can be used to push more flow from \( s \) to \( t \); reverse edge has surplus flow that can be redirected to push more flow from \( s \) to \( t \).

Note: this is a form of backtracking – changing our mind about previously assigned flow.

Augmenting path: \( s-t \) path where each edge has positive residual capacity (i.e., \( c(e) - f(e) > 0 \) for original edges \( e \), \( f(e) > 0 \) for reverse edges \( e \)).

Augmentation: add \( \Delta f(P) \) (defined as before) to original edges, subtract it from reverse edges. (Example.)

Correctness of Ford-Fulkerson Algorithm:
A cut is a partition of \( V \) into \( V_s, V_t \) (i.e., \( V = V_s \cup V_t \) and \( V_s \cap V_t = \{ \} \)) such that \( s \in V_s \) and \( t \in V_t \):

- an edge \((u, v)\) with \( u \in V_s, v \in V_t \) is a forward edge;
- an edge \((u, v)\) with \( u \in V_t, v \in V_s \) is a backward edge.

For any cut \( X = (V_s, V_t) \),

- The capacity of cut \( X \) is the sum of the capacities of the forward edges: \( c(X) = \sum_{e: \text{forward}} c(e) \).
- The flow across \( X \) is the total flow forward minus the total flow backward across the cut: \( f(X) = \sum_{e: \text{forward}} f(e) - \sum_{e: \text{backward}} f(e) \).

Lemma: For any cut \( X \) and any flow \( f \), \( f(X) \leq c(X) \).

Proof: \( f(X) = \sum_{e: \text{forward}} f(e) - \sum_{e: \text{backward}} f(e) \leq \sum_{e: \text{forward}} c(e) = c(X) \).

Lemma: For any cut \( X \) and any flow \( f \), \( f(X) = |f| \).

Proof: Consider cut \( X = (V_s, V_t) \). By conservation, \( f^{out}(v) = f^{in}(v) \) for each \( v \) except \( s, t \). By definition, \( f^{out}(s) = |f| \) and \( f^{in}(s) = 0 \). Hence, by definition of \( f^{out} \) and \( f^{in} \):

\[
|f| = f^{out}(s) = \sum_{v \in V_s} f^{out}(v) - f^{in}(v) = \sum_{v \in V_s} \sum_{(u,v) \in E} f(v,u) - \sum_{(u,v) \in E} f(u,v) \tag{1}
\]

For each edge \( e = (u,v) \),

- if \( u, v \in V_t \), then \( f(u,v) \) does not appear in Equation 1.
- if \( u, v \in V_s \), then \( f(u,v) \) appears twice in Equation 1: once positively in \( f^{out}(u) \) and once negatively in \( f^{in}(v) \), both of which cancel each other out.
- if \( u \in V_s, v \in V_t \), then \( f(u,v) \) appears once in Equation 1: positively in \( f^{out}(u) \).
- if \( u \in V_t, v \in V_s \), then \( f(u,v) \) appears once in Equation 1: negatively in \( f^{in}(v) \).

Hence, the only terms that appear in Equation 1 without canceling each other out are \( f(u,v) \) for \( u \in V_s, v \in V_t \) and \(-f(u,v)\) for \( u \in V_t, v \in V_s \), i.e.,

\[
|f| = \sum_{u \in V_s} \sum_{v \in V_t} f(u,v) - \sum_{u \in V_t} \sum_{v \in V_s} f(u,v) = \sum_{e: \text{forward}} f(e) - \sum_{e: \text{backward}} f(e) = f(X).
\]

Corollary: For any cut \( X \) and any flow \( f \), \(|f| \leq c(X) \). (From two facts above). In particular, max flow in network \( \leq \min \) capacity of any cut.
Theorem (Ford-Fulkerson): For any network \( N \) and flow \( f \), \(|f|\) is maximum (and equal to \( c(X) \) for some cut \( X \)) if and only if there is no augmenting path.

Proof: \((\Rightarrow)\) augment

\((\Leftarrow)\) Construct cut \( X \) as follows:

- Let \( V_s \) be all nodes in \( V \) that are reachable from \( s \) in \( G_f \).
- Let \( v_t = V - V_s \) (all nodes not reachable from \( s \) in \( G_f \)).

Since there is no augmenting path, \( t \in V_t \). By definition of \( X \), every edge crossing \( X \) has property that \( f(e) = c(e) \) for forward edges and \( f(e) = 0 \) for backward edges (otherwise you can find a path from \( s \) to a node in \( V_t \)). Hence, \(|f| = f(X) = c(X)\).

Corollary (max-flow/min-cut theorem): For any network, the maximum flow value equals the minimum cut capacity.

Additional property: because of nature of augmentation, we can prove by induction that max flow can always be achieved with integer flow values (as long as all capacities are integer).