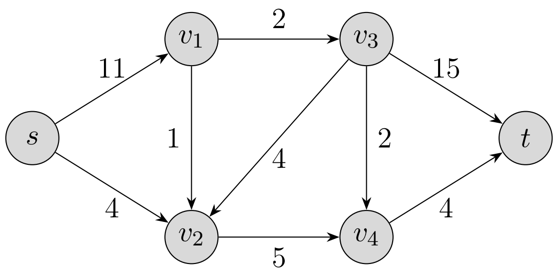
COL751 - Lecture 8

1 Flows and Cuts

A flow network is a (directed/undirected) graph $G = (V, E, c)$ satisfying $c(e) \geqslant 0$, for each edge e.



For any flow $f : E \to \mathbb{R}^+ \cup \{0\}$, the following constraints must be satisfied:

1. Capacity constraint: Flow $f(e)$ through edge e is bounded by its capacity $c(e)$.
2. Flow conservation: Flow entering a node x is identical to flow exiting a node x, for every $x \neq s, t$.

We define

$$f_{out}(x) = \sum_{(x,\, y)\, \in E} f(x, y).$$

Similarly,

$$f_{in}(x) = \sum_{(y,\, x)\, \in E} f(y, x).$$

Definition 1 *The value of a flow f, denoted by $value(f)$, is defined as $f_{out}(s)$.*

Definition 2 *An (s,t)-cut is any partition (A, B) of vertices satisfying $s \in A,\ t \in B$. The capacity of cut (A, B) is defined as*

$$\sum_{\substack{(x,y)\in \\ (A\times B)\cap E}} c(x, y).$$

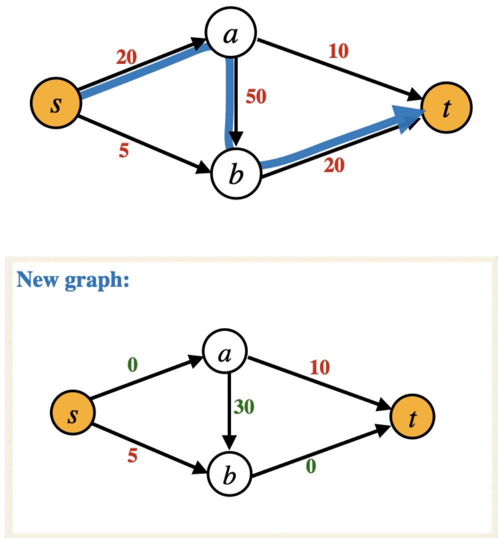
Lemma 3 *For any (s,t)-cut (A, B) and any flow f,*

$$value(f) \;=\; f_{out}(A) - f_{in}(A).$$

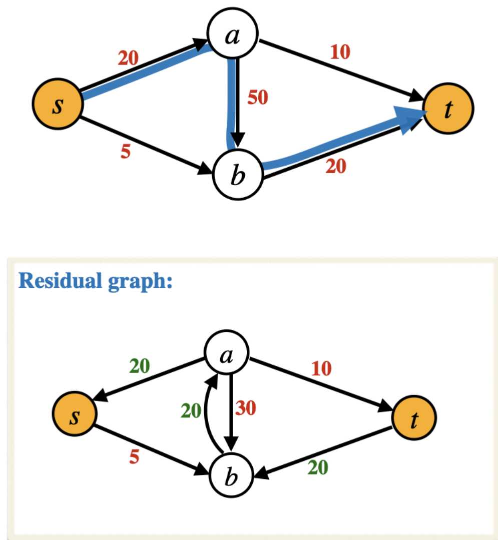
Corollary 4 *For any flow f, we have $f_{out}(s) = f_{in}(t)$.*

2 Computing (s,t) max-flow

Consider the problem of computing a valid flow f for a G with source s and destination t that maximizes $f_{out}(s)$. A naive greedy approach would be to pass maximum possible flow along a path, and cancel the saturated edges. However, this strategy doesn't work. See Figure (a).



(a) Passing maximum possible flow through an (s,t) path.



(b) Introducing reverse edges to cancel flow through existing edges.

Given a flow f, we construct a *residual graph* G_f with respect to f as below:

If $c(x,y) - f(x,y) > 0$	Include (x,y) in G_f with $c_r(x,y) = c(x,y) - f(x,y)$	Forward Edge
If $f(x,y) > 0$	Include (y,x) in G_f with $c_r(y,x) = f(x,y)$	Backward Edge

We will argue that algorithm below computes a max-flow.
Note that

1. Capacity constraint is satisfied for each edge.
2. Flow at each node other than source/destination is conserved.
3. Flow increases in each round.
4. If all the edges have integer capacity then number of rounds is at most $\sum_{e \in E} c(e)$, or alternatively $O(m \cdot \text{maxflow}(s,t,G))$.

Lemma 5 *An (s,t)-flow f is a max-flow if and only if there is no (s,t) path in residual graph G_f.*

Initialise $f = 0$;
while ($\exists\ s \to t$ *path in* G_f) **do**
  Compute residual graph G_f, and find an (s,t) path P in G_f;
  Let $c_{min} = \min\{c(e) \mid e \in P\}$;
  foreach $(x,y) \in P$ **do**
    if (x,y) *is a forward edge* **then** $f(x,y) = f(x,y) + c_{min}$;
    if (x,y) *is a backward edge* **then** $f(x,y) = f(x,y) - c_{min}$;
  end
end

Algorithm 1: Ford-Fulkerson-algorithm(G, s, t)

Proof: Let f be the max-flow computed from Ford-Fulkerson algorithm. Let A be vertices reachable from s in G_f, and let $B = V \setminus A$. Then, it can be argued that

1. For each edge $(x,y) \in A \times B$, $f(x,y) = c(x,y)$.

2. For each edge $(x,y) \in B \times A$, $f(x,y) = 0$.

This proves the claim. $\square$

Theorem 6 *For any directed/undirected graph $G = (V, E, c : E \to \mathbb{Z}^+)$ with a source s and destination t, an (s,t)-max-flow can be computed in $O(m \cdot \text{maxflow}(s,t,G) + n)$ time.*

In proof of Lemma 5, we see that there exists an (s,t)-cut (A, B) satisfying $c(A, B) =$ value of (s,t)-max-flow. Since all min-cuts have same size, and their size is lower bounded by (s,t)-flow value, we get the following theorem.

Theorem 7 (Max-Flow Min-Cut Theorem) *For any directed/undirected graph $G = (V, E, c)$, the maximum amount of flow passing from a source s to a sink t is equal to the total weight of the edges in a minimum (s,t) cut, i.e., the smallest total weight of the edges which if removed would disconnect the source from the sink.*

3 An application of Max-Flow Min-Cut Theorem

Definition 8 *An undirected graph $G = (V, E)$ with at least three vertices is said to be*

- *k-edge connected if for all distinct pairs $(x,y) \in V \times V$, there are k-edge disjoint paths between x and y in G.*

- *k-vertex connected if for all distinct pairs $(x,y) \in V \times V$, there are k-internally-vertex disjoint paths between x and y in G.*

Homework Let G be a k-edge-connected graph. Show how to compute in $O(mk)$ time a sparse subgraph H of G with $O(nk)$ edges such that H is also k-edge-connected.