Math 4707: Max-Flow Min-Cut

3/17

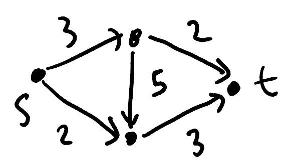
Not in LPV

Reminder: • HW #4 due in one week, on Wed. 3/24.

Building on the max. matching algorithm from last class, today we will discuss another optimization problem w/ a similar algorithmic solution: **Max-Flow**.

The input to Max-Flow is a directed graph $G=(V,E)$, or more specifically a **network**, which is a digraph w/ extra decoration: a **capacity function** $\kappa: E \to \mathbb{N}$ that assigns nonnegative capacities to each edge, and the choice of special **source** and **target** vertices $s,t \in V$:

e.g.

$G =$ 

($\kappa(e)$ written on $e \in E$)

harmless assumption: $\text{indeg}(s) = \text{outdeg}(t) = 0$

The **Max-Flow** problem asks, given a network, what is the maximum amount (of water, current, etc.) that we can flow from s to t, given that we cannot flow more than capacity at each edge, and flow has to be **conservative** at all vertices other than s,t.

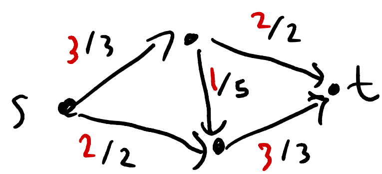
Formally...

Def'n A **flow** $f : E \to \mathbb{N}$ in a network G is an assignment of nonneg. integers to the edges s.t.:

- $f(e) \leq \kappa(e) \quad \forall e \in E$ (capacity constraint)
- $\sum_{(u,v)\in E} f((u,v)) = \sum_{(v,w)\in E} f((v,w)) \quad \forall\, v \in V - \{s,t\}$ (conservative except at s + t)

The **value** of a flow f is $\text{value}(f) := \sum_{(s,v)} f((s,v))$, which also equals $\text{value}(f) = \sum_{(v,t)} f((v,t))$ (amount we flow $s \to t$).

e.g.



flow $f(e)$ / capacity $\kappa(e)$

$\text{value}(f) = 5$

The **Max-Flow** problem: what is $\max_f \text{value}(f)$?

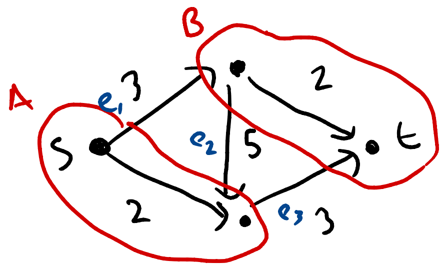
As we'll see, has an answer in terms of **cuts**.

Def'n An **(s,t)-cut** in a network G is a partition $C = (A, B)$ of the vertices V into two parts, w/ $s \in A$ and $t \in B$.

The **value** of a cut C is

$$\text{value}(C) = \sum_{(u,v) :\, u \in A,\, v \in B} \kappa((u,v)).$$

e.g.



Value(C) =
$\kappa(e_1)+\kappa(e_3)$
$= 3+3 = 6$

Intuitively, value of flow in a network cannot be more than the value of any cut, because cut acts as a bottleneck:

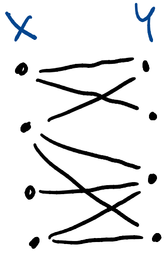
Prop. For any flow f and cut C in G, have
$$\text{value}(f) \le \text{value}(C).$$

Pf: Skipped. Exercise for you. ▲

So $\max_f \text{value}(f) \le \min_C \text{value}(C)$. Surprising fact is: we have an equality!

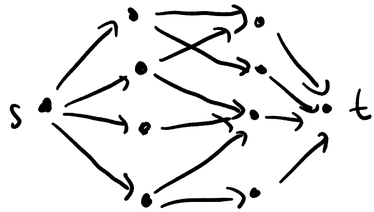
Thm (Max-Flow Min-Cut)
For any network, $\max_f \text{value}(f) = \min_C \text{value}(C)$.

Before we discuss proof, let's show how this generalizes max. matching in a bipartite graph problem, and Hall's Marriage Thm:



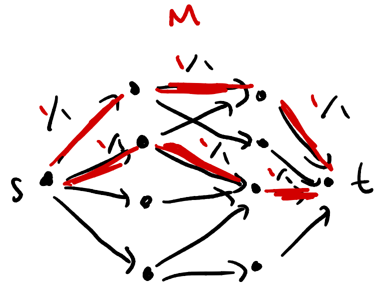
bipartite graph G

$\rightsquigarrow$



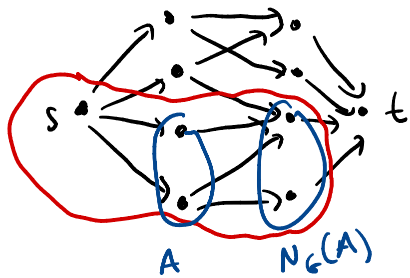
network $\tilde{G}$

all capacities $\kappa(e) = 1$



matching in G = flow in $\tilde{G}$

size M = value(f)



$A \subseteq X \rightsquigarrow$ cut C w/ one half $= \{s\} \cup A \cup N_G(A)$

$$(\#X - \#A) + \#N_G(A) = \text{value}(C)$$

$$\Rightarrow \underset{\text{in } G}{\text{max. size matching}} = \max_{f \text{ in } \tilde{G}} \text{value}(f) = \min_{C \text{ in } \tilde{G}} \text{value}(C) = \min_{A \subseteq X} \#X - \#A + \#N_G(A)$$

✓

Now let's discuss proof of Max-Flow Min-Cut.

The proof will be an algorithm for finding max. flow.

First consider a "naive" algorithm:

The 'naive' algorithm starts w/ any flow f (e.g., all zero flow) and looks for directed path P in G from s to t, w/ $\kappa(e) - f(e) > 0$ for all edges e in P. It then increases the flow along every e in P by $\Delta = \min_{e \in P} \kappa(e) - f(e)$ to make new flow f'.

We repeat this as long as we can. (See $f \rightsquigarrow f'$ above.)

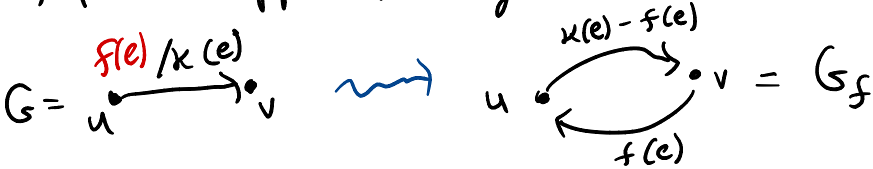
BUT in above example, no more paths P for f', even though value$(f') = 3$, and we know value of 5 is possible. So 'naive' alg. doesn't quite work...

To correct alg., need notion of residual network.

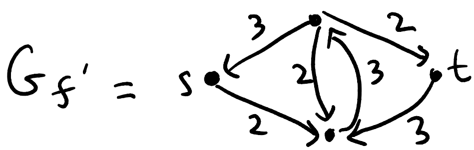
Def'n For f a flow in a network G, the residual network G_f is network w/ same vertices as G, and edges $e, \bar{e}$ for $e \in E(G)$, where if $e = (u,v)$, $\bar{e} = (v,u)$ is opposite edge.

The capacities in G_f are $\kappa(e) - f(e)$ for e and

$f(e)$ for an opposite edge $\bar{e}$:



e.g. with f' as above, residual network is

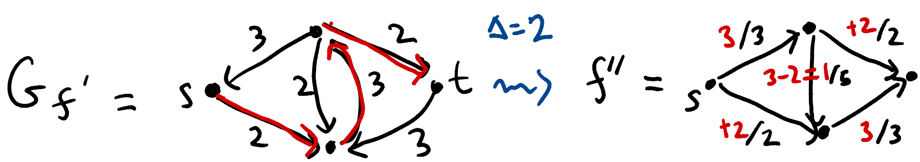


(note we DO NOT DRAW EDGES w/ ZERO CAPACITY)

The residual network allows us to "undo" some of our flow. We apply same idea as in 'naive' alg., but now search for s-to-t path P along edges w/ nonzero capacity in residual network G_f, and we update flow f by:

- adding $\Delta := \min_{e \in P} \kappa_{G_f}(e)$ to edges e in P that are real (forwards) edges in G,
- subtracting Δ from $\bar{e}$ in P that are opposite edges in G.

e.g., continuing above example:



$\text{value}(f'') = 5$ ✓

As in naive alg., we terminate when we cannot find an s-to-t path P to augment along. In above example, we stop at f'' (which is a max flow). This is called the Ford-Fulkerson algorithm and it works!

Thm • Ford-Fulkerson terminates in finite time.

• It produces a flow f w/ $\mathrm{value}(f) = \min_C \mathrm{value}(C)$, hence a maximal flow.

Pf sketch: • Termination: flow increases by $\Delta \geq 1$ at every step.

• Maximality: if there are no s-t paths in G_f, set $A := \{\text{vertices reachable from } s \text{ in } G_f\}$.
Then $C = (A, V-A)$ is a cut w/ $\mathrm{value}(f) = \mathrm{value}(C)$. $\square$

See Bondy-Murty Textbook for full proof.
This proves Max-Flow Min-Cut theorem.

Rmk: F-F also works w/ rational capacities, but bizarrely may not terminate w/ irrational capacities, even though MFMC is still true.

Now let's take a 5 min. break,

and when we come back,

run the F-F alg. on today's

worksheet in breakout groups...