CSE 373

Network Flow

Network Flow

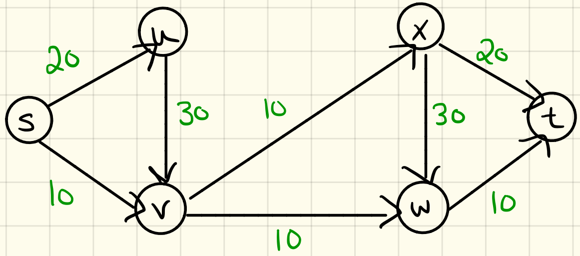
- Slightly different algo design technique
- Will see an algorithm for MaxFlow (and variants) that is able to solve a wide range of problems simply by posing them as network flow.

Flow Network

Def: Connected, directed graph $G = (V, E)$

- every edge e has an <u>integral</u>, non-negative <u>capacity</u> c_e
- there is a designated <u>source</u> node $s \in V$
- there is a designated <u>sink</u> node $t \in V$
- no edge enters the source or leaves the sink.

E.g.



Def: Flow

An s-t flow is a function $f: E \to \mathbb{R}^{\geq 0}$ that assigns a non-negative real number to each edge, subject to the flow constraints.

1) $0 \leq f(e) \leq c_e$ for each edge
2) For every node except s and t, we have:

$$\sum_{e \text{ into } v} f(e) = \sum_{e' \text{ leaving } v} f(e')$$

these "balance constraints" say that whatever incoming flow we have at a node must also leave that node.

Some notation:

The <u>value</u> of a flow f is: $v(f) = \sum_{e \text{ leaving } s} f(e)$

- the amount the flow is able to "send"

$$f^{in}(v) = \sum_{e \text{ into } v} f(v) \quad , \quad f^{out}(v) = \sum_{e \text{ leaving } v} f(e)$$

$\to$ balance constraints become $f^{in}(v) = f^{out}(v) \quad \forall v \in (V - \{s,t\})$

The Maximum Flow problem:

Given a flow network G

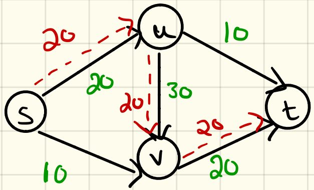
Find a flow f on G of maximum possible value.

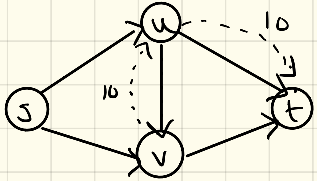
How to design an algorithm for such a problem? Thoughts?

What would a greedy approach do?

- Start with $f(e) = 0 \ \forall e$
- Pick some s-t path and "push" flow along it up to capacity. Repeat
- When we get "stuck", we can erase flow along some edges

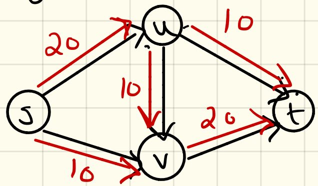
E.g.





After first path, have $v(f) = 20$ ... would like to send some flow "back"

Now, we've freed up capacity on the (v,t) edge



total $v(f) = 30$

Let's make this idea of "erasing" more formal

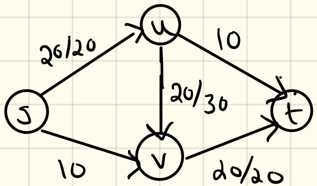
Residual graph: G_f — depends on the flow f

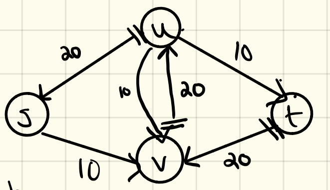
1) G_f contains the same nodes as G
2) Consists of two different types of edges:

- Forward Edges: For each $e = (u,v)$ of G for which $f(e) < c_e$, include edge $e' = (u,v)$ in G_f with capacity $c_e - f(e)$.

- Backward Edges: For each $e = (u,v)$ in G with $f(e) > 0$, we include an edge $e' = (v,u)$ in G_f with capacity $f(e)$

So, forward edges replace "original" edges, but modify their capacities to be the remaining / unused / residual capacities.
Backward edges can "erase" flow (up to $f(e)$) along edge e.





Residual graph G_f

⊩→ (backward edge) → (forward edge)

Def: Augmenting paths

- Let P be an s-t path in G_f
- Let b = bottleneck(P, f) be the smallest capacity in G_f on any edge of P
- If $b > 0$, then we can increase the total flow by sending b along path P.

Consider the following procedure

```
augment(f, P):
    b = bottleneck(P, f)
    for each (u, v) ∈ P:
        if e = (u, v) is a forward edge:
            increase f(e) in G by b
        else:
            e' = (v, u)
            decrease f(e') in G by b

    return f    ← return the new flow
```

Applying augmentation repeatedly leads to the Ford-Fulkerson algorithm

```
MaxFlowFF(G):

    Set f[e] = 0 ∀ e ∈ G
    While P = FindPath(s, t, Residual(G, f)):
        f = augment(f, P)
        update Residual(G, f)

    return f
```

Some relevant questions about this algorithm:

(1) Does the "augment" step preserve the validity of the flow?
(2) How long will it run? Does it terminate?
(3) How can we be sure the resulting flow is a maximum?

(1) Does the "augment" step preserve the validity of the flow?

Theorem: After $f' = \text{augment}(P, f)$, f' is still a valid flow

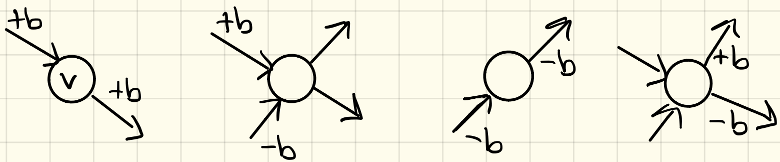
Lemma: Augmentation preserves the capacity constraints

- If e is a forward edge, it has capacity $c_e - f(e)$. Therefore:
  $f'(e) = f(e) + \text{bottleneck}(P, f) \leq f(e) + (c_e - f(e)) \leq c_e$

- If e is a backward edge, it has capacity $f(e)$. Therefore:
  $f'(e) = f(e) - \text{bottleneck}(P, f) \geq f(e) - f(e) = 0$

Lemma: Augmentation preserves the balance constraints:

Consider some node v; there are 4 possibilities if v is on P:



Each such situation preserves the balance constraints.

Termination:

(1) Since the original flow consisted of only integers, so does every augmenting flow (we didn't prove this; why is it true?).

(2) At every augmentation, we increase the values of the flow by bottleneck (P, f), which is always ≥ 1.

(3) We can never send more than $C = \sum_{e \text{ leaving } s} c_e$ total flow (this is not a tight bound)

Hence: The Ford-Fulkerson algorithm terminates in at most C iterations of the while loop.

Further: (1) If G has m edges, G_f has $\leq 2m$ edges
(2) We can find an s-t path in G_f in $O(m+n) = O(m)$ time (BFS / DFS)
(3) Since $m \geq n/2$ (every node is adjacent to some edge)

We have: The Ford-Fulkerson algorithm runs in $O(mC)$ time.

Note: This makes the FF algorithm <u>pseudopolynomial</u> time.

Note: This makes the FF algorithm <u>pseudopolynomial</u> time.

Other MaxFlow algorithms can rectify this

e.g. $O(nm^2) \rightarrow$ Edmonds - Karp

$O(m^2 \log C) \rightarrow$ Scaling max flow (this is polynomial)

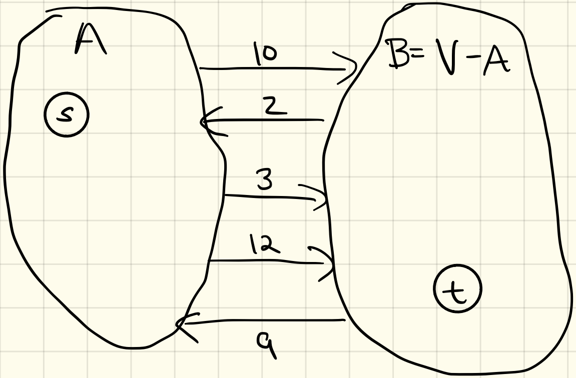
$O(n^2 m)$ or $O(n^3) \rightarrow$ preflow push

Now, the difficult question is:

How do we know that the flow we get back is maximum?

Cuts and Cut Capacity

Def: The capacity of an s-t cut (A, B) is the sum of the capacities of the edges leaving A



E.g. the capacity of this cut is

capacity(A, B) = 25

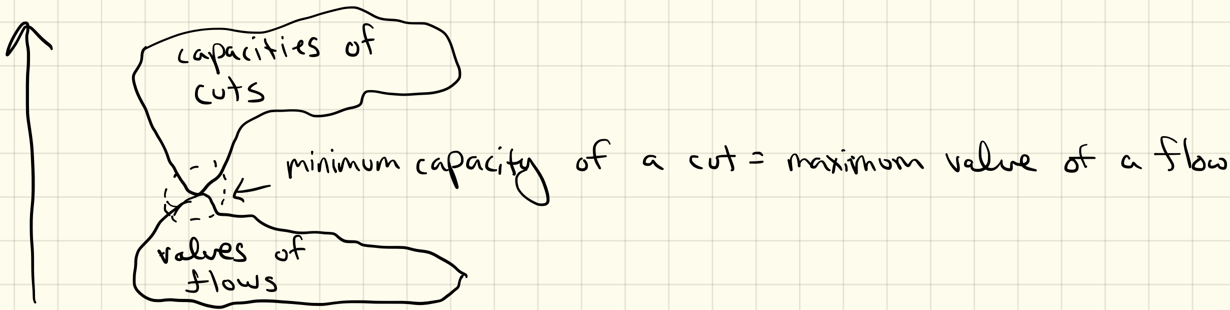
<u>Theorem</u>: Let f be an s-t flow and (A, B) be an s-t cut.
Then $v(f) = f^{out}(A) - f^{in}(A)$.

That is, the value of the flow is the same as the value it takes across any cut from s's component to t's component. Think about how to prove this (sec 7.2).

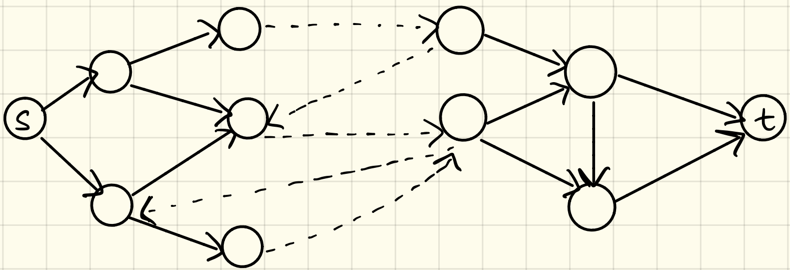
Theorem ✡: Let f be an s-t flow and (A,B) be <u>any</u> s-t cut. Then $v(f) \leq \text{capacity}(A,B)$.

Proof:
$$\begin{aligned} v(f) &= f^{out}(A) - f^{in}(A) \\ &\leq f^{out}(A) \\ &= \sum_{e \text{ leaving } A} f(e) \\ &\leq \sum_{e \text{ leaving } A} c_e \\ &= \text{capacity}(A,B) \end{aligned}$$

Theorem ✡ says that any cut is at least as big as any flow. Therefore, cuts constrain/bound flows. The <u>minimum capacity</u> cut bounds the <u>maximum flow</u>. <u>In fact</u>, the minimum cut value always equals the maximum flow value.



Let f^* be a flow returned by our algorithm.
Look at G_{f^*}, but define a cut in G:



A^* = nodes reachable from s in residual graph G_{f^*}

Cut = (A^*, B^*)

Along this cut, forward edges must be <u>saturated</u>, backward edges must have 0 flow.

This implies $v(f^*) = \text{capacity}(A^*, B^*)$

- (A^*, B^*) is an s-t cut because there is no path from s-t in the residual graph G_{f^*}

- Edges (u,v) from A^* to B^* must be saturated - otherwise there would be a forward edge (u,v) in G_{f^*} and v would be part of A^*.

- Edges (v,u) from B^* to A^* must be empty - otherwise, there would be a backward edge (u,v) in G_{f^*} and v would be part of A^*

Therefore:

- $v(f^*) = \text{capacity}(A^*, B^*)$
- No flow can have a value larger than capacity (A^*, B^*)
- So, f^* must be a maximum flow
- And, (A^*, B^*) must be a minimum capacity cut

Yielding Theorem (Max Flow = Min Cut):

- The value of the maximum flow in any flow graph is equal to the capacity of the minimum cut.

Finally, note this proof is <u>constructive</u>, it can be used to <u>find</u> the minimum capacity cut.

(1) Find the max flow f^*
(2) Construct the residual graph G_{f^*}
(3) Do a BFS to find the nodes reachable in G_{f^*} from s; let these define A^*
(4) Let B^* be all other nodes

(5) Return (A^*, B^*) as a minimum capacity cut.