LECTURE #14

CS 170

Spring 2021

Last time:

Linear programming: expressing and solving linear optimization problems.

Note: dynamic programming $\neq$ linear programming

[recursion + memoization] [high-dim optimization problem]

Today:

Maximizing flow in a network (graph with capacities)

- Some "direct" applications: pipes moving water/oil, traffic in roads, ...
- Numerous "indirect" applications as a powerful subroutine:

  bipartite matching, airline scheduling, edge-disjoint paths, ...

INPUTS: directed graph $G=(V,E)$

source $s\in V$ and sink $t\in V$

positive capacities $c: E \to \mathbb{Z}_{>0}$

OUTPUT: maximum flow from s to t

<u>def</u>: A (feasible) flow is a function $f: E\to\mathbb{R}$ that satisfies:

① capacity constraints: $\forall e\in E,\quad 0\leq f(e)\leq c(e)$

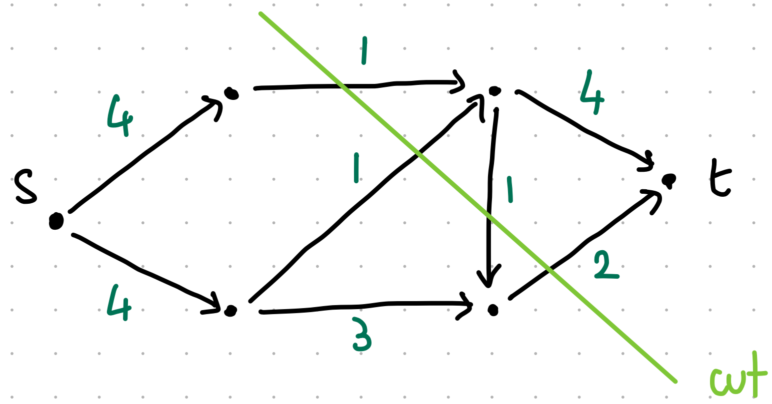
② conservation constraints: $\forall u\in V\setminus\{s,t\},\quad \sum_{(w,u)\in E} f(w,u) = \sum_{(u,v)\in E} f(u,v)$

<u>def</u>: The value of a (feasible) flow $f: E\to\mathbb{R}$ is

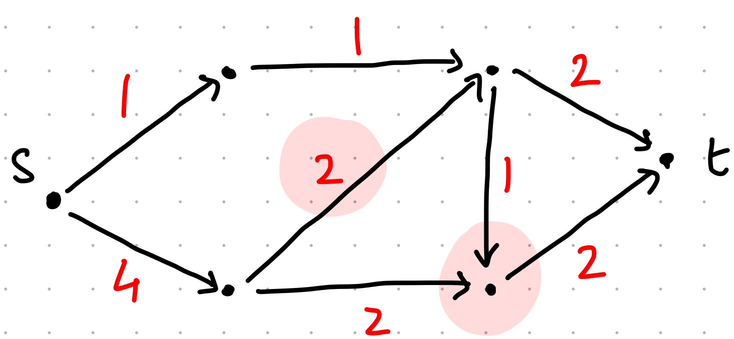
$$\mathrm{val}(f) := \sum_{(s,u)\in E} f(s,u) \quad \left(\text{also equals } \sum_{(u,t)\in E} f(u,t)\right).$$

Example

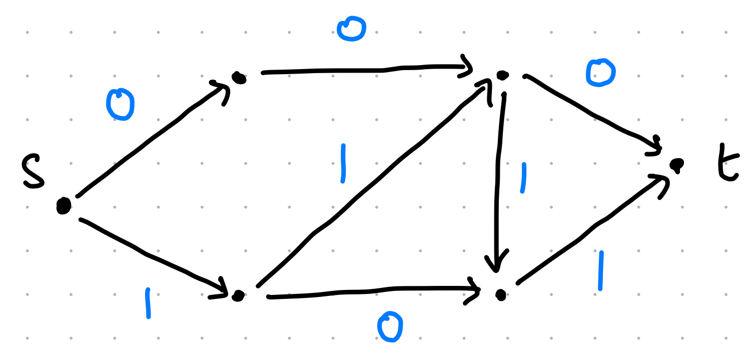
input :



infeasible flow

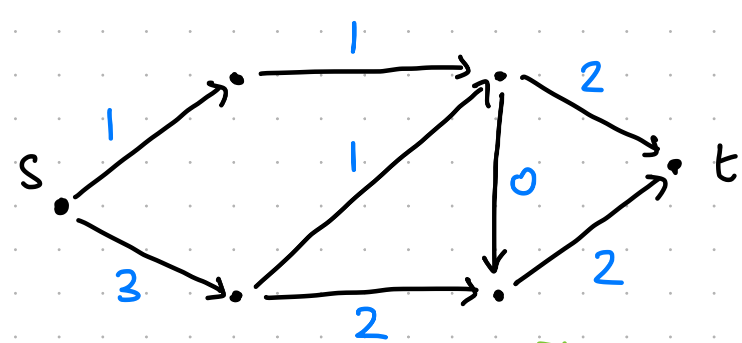


feasible flow



value = 1

feasible flow



value = 4

this is a maximum flow because of cut above

Q: how do we find maximum flows?

A generic solution

We want to solve this optimization problem:

$$\boxed{\max \ \mathrm{val}(f) \quad \text{s.t.} \quad f \text{ is feasible w.r.t. } (G,s,t,c).}$$

claim: network flow reduces to linear programming

proof: The variables are $\{f(e)\}_{e\in E}$. There are $O(|E|+|V|)$ linear constraints:

$$\left(\forall e \in E \quad f(e) \geq 0,\ f(e) \leq c(e)\right) \text{ and } \left(\forall u \in V\setminus\{s,t\},\ \sum_{(w,u)\in E} f(w,u) = \sum_{(u,v)\in E} f(u,v)\right).$$

$\geq$ & $\leq$

The objective function is linear: $\mathrm{val}(f) := \sum_{(s,u)\in E} f(s,u)$. $\blacksquare$

Hence we can solve the problem via linear programming:

$$x_{NF} = (G,s,t,c) \xrightarrow{\boxed{\text{claim}}} x_{LP} \longrightarrow \boxed{\text{LP algo}} \longrightarrow \text{LP solution} \longrightarrow \text{NF solution}$$

$(|x_{LP}| = O(|x_{NF}|))$

This takes time $\mathrm{poly}(|x_{LP}|) = \mathrm{poly}(|x_{NF}|)$. Can we do better?

Towards a direct algorithm for Network Flow (direct = w/o reducing to LP)

Attempt: greedy strategy

① init a zero flow: $\forall e \in E \;\; f(e) := 0$

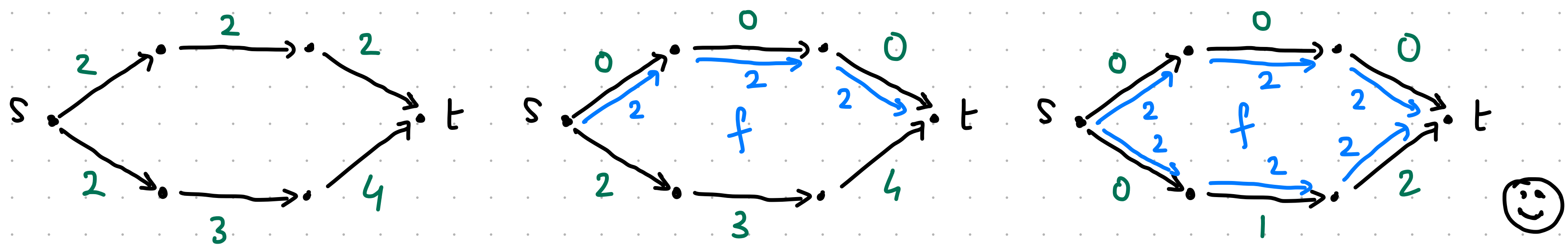
② repeat the following until no longer possible:

pick any path from s to t in the residual network G^f

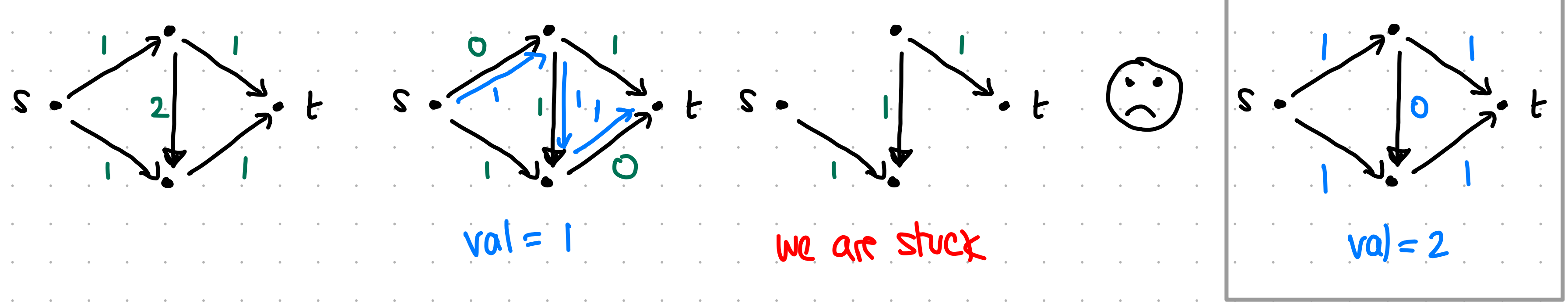
and increase the flow f by max amount possible on this path

- vertices $V^f := V$
- edges $E^f := E$
- capacities $c^f(e) := c(e) - f(e)$

Ex:



Ex:



The greedy strategy fails to find the maximum flow.

Idea: enable new paths to cancel existing flow (not greedy anymore)

New definition of residual network G^f

- vertices $V^f := V$ (as before)
- edges $E^f := \{(u,v) \text{ if } f(u,v) < c(u,v)\} \cup \{(u,v) \text{ if } f(v,u) > 0\}$
- capacities $c^f(u,v) := \begin{cases} \text{if } (u,v) \in E \text{ and } f(u,v) < c(u,v): & c(u,v) - f(u,v) \\ \text{if } (v,u) \in E \text{ and } f(v,u) > 0: & f(v,u) \end{cases}$

We can now add $c(u,v) - f(u,v)$ to an edge (u,v) at sub-capacity, or send back $f(v,u)$ on an edge (u,v) to cancel the flow $f(v,u)$ on (v,u).

only difference is def of residual network

Algorithm: ① init a zero flow: $\forall e \in E \;\; f(e) := 0$

② repeat the following until no longer possible:
pick any path from s to t in the residual network G^f
and increase the flow f by max amount possible on this path

We are left to discuss:
- efficiency
- correctness

Running time of Max Flow algorithm:

In each iteration:

$O(|E|)$ for deducing residual network, finding an (s,t)-path, updating the current flow

How many iterations?

- There are examples where the algorithm does NOT terminate!
- If capacities are integers in $\{1,2,\dots,c\}$ and arbitrary choice of (s,t)-paths:

  $val(f_i)$ is an integer and $val(f_{i+1}) > val(f_i)$

  so # iterations $\overset{\circledast}{\leq} val(f_{max}) \leq c \cdot |E|$. [$\circledast$ this bound is tight]

  $\Rightarrow$ this is the Ford–Fulkerson algorithm and has time $\underbrace{O(|E| \cdot val(f_{max}))}_{\text{only weakly polynomial time}}$.

- If use BFS (fewest edges) to find (s,t)-paths:

  # iterations $\leq O(|V| \cdot |E|)$ (this requires a proof but we do not discuss it)

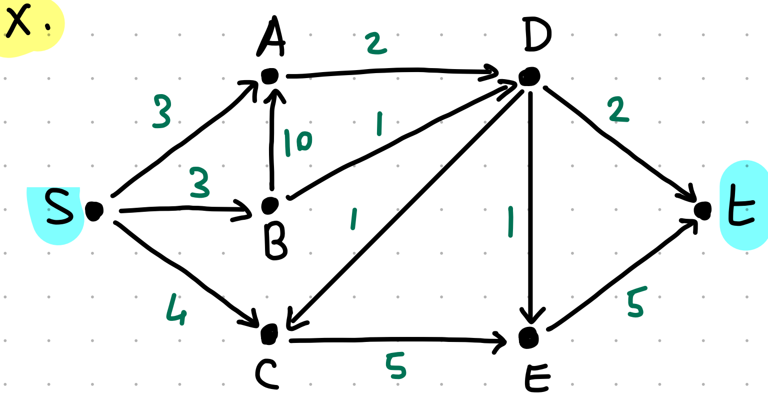
  $\Rightarrow$ this is the Edmonds–Karp algorithm and has time $O(|E|^2 \cdot |V|)$.

Cuts Bound Flows

An (s,t)-cut in $G=(V,E)$ is a pair (L,R) s.t. $s \in L$, $t \in R$, $L \cup R = V$, $L \cap R = \emptyset$.

<u>def:</u> capacity(L,R) := total capacity from L to R (w/o subtractions).

Ex.



$L = \{s, A, B, C, D\}$

$R = \{E, t\}$

capacity$(L,R) = 8$

$\mathrm{val}(f_{max}) \leq 8$

Observation: $\forall$ (s,t)-flow f $\forall$ (s,t)-cut (L,R) $\mathrm{val}(f) \leq$ capacity(L,R).

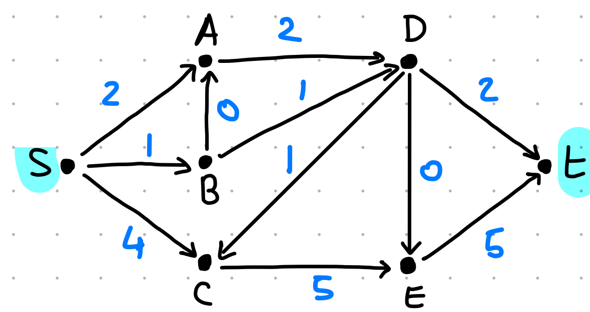
Other choices of cut give better upper bounds:

$L = \{S, A, B\}$

$R = \{C, D, E, t\}$

capacity$(L,R) = 7$

this is optimal because $\exists$ flow f s.t. $\mathrm{val}(f) = 7$ and so it certifies that f is optimal



Max-Flow Min-Cut Theorem

$$\max_{\text{flow } f} \ \mathrm{val}(f) = \min_{\text{cut } (L,R)} \ \mathrm{capacity}(L,R)$$

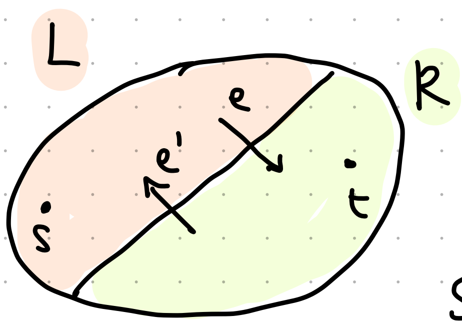
Proof:

We have already shown that, $\forall$ flow f $\forall$ cut (L,R), $\mathrm{val}(f) \leq \mathrm{capacity}(L,R)$.

Let f^* be a max flow in G.
There is no path from s to t in G_{f^*} (for otherwise we could improve f^*).
Define

$$L := \{ v \in V \text{ s.t. } \exists \text{ path from } s \text{ to } v \text{ in } G_{f^*} \} \text{ and } R := V \setminus L.$$

Note that (L,R) is an (s,t)-cut because $s \in L$, $t \in R$, and (L,R) partition V.
Moreover: $\mathrm{capacity}(L,R) = \mathrm{val}(f^*)$.



- $f^*(e) = c(e)$ [if $f^*(e) < c(e)$ then s has a path into R in G_{f^*}]
- $f^*(e') = 0$ [if $f^*(e') > 0$ then s has a path into R in G_{f^*}]

So the net flow across (L,R) is capacity(L,R). ∎

↳ We also deduce the correctness of the max flow algorithm!