# Lecture 13: Plz do survey!

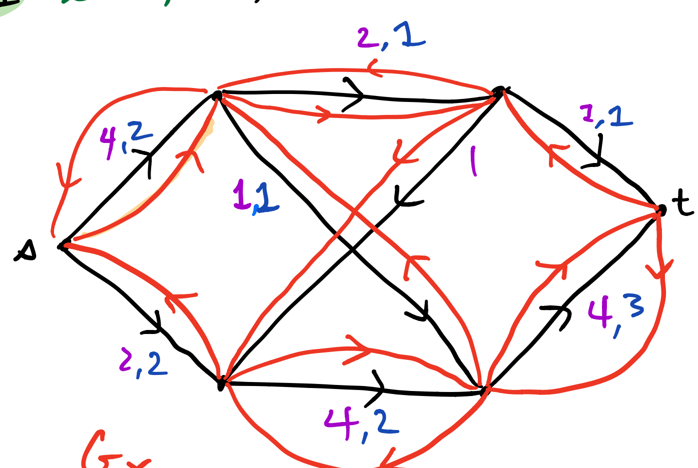
**Plan:**

1) algorithm for max flow
2) global min cut.

## Recap: Augmenting flows:

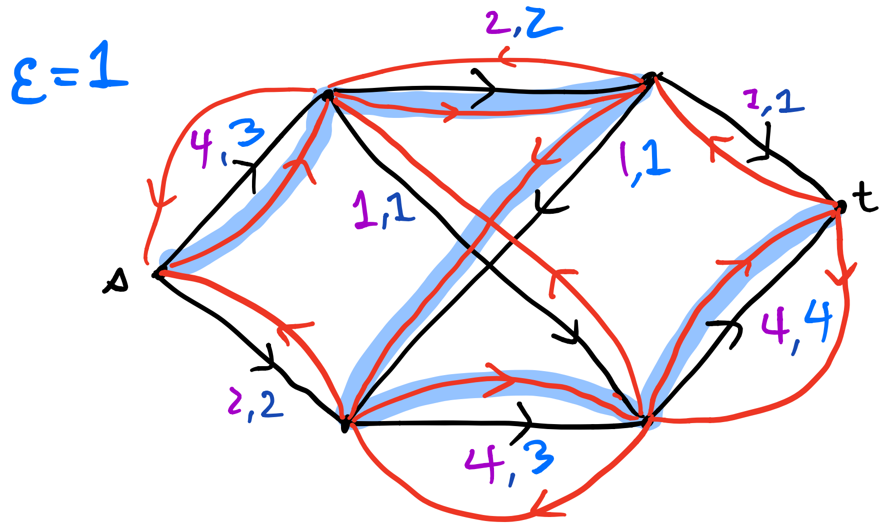
- Given a flow $x$, compute residual graph $G_x$.

e.g. $\ell = 0$, $u$, $x$

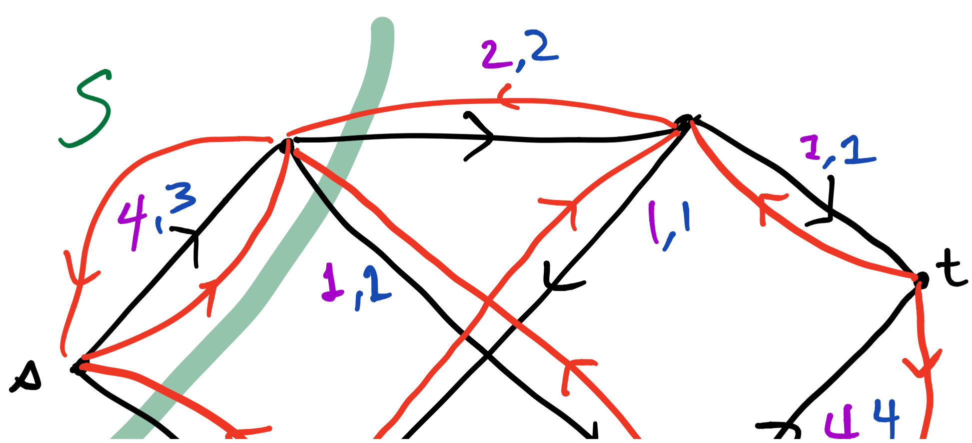


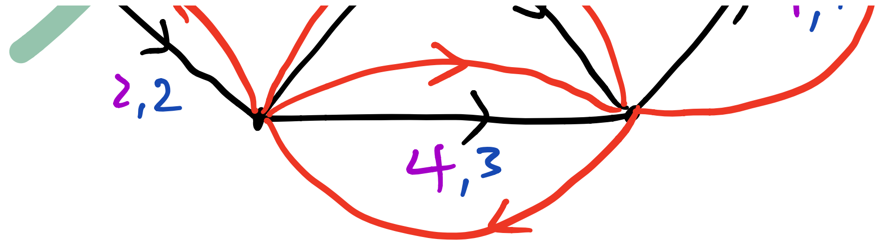

$G_x$.

- If ∃ s-t path in residual, ε-more units of flow can be sent along it.




- if no s-t path, is cut S with capacity $C(S) = |X|$, terminate.






**Problem:** might not terminate!

- if <u>irrational</u> may run forever.
- if rational, can multiply capacities by s.g. to make integers.
- If capacities integral, can take $\epsilon > 0$ to be integral, so must terminate.
- in integral case, #steps

naively bounded by

$$\boxed{\sum_e |u(e) - \ell(e)|}$$

but this is not polynomial in input size!

**Remember:** need only $1 + \log_2 |\ell(e)|$ bits to represent $\ell(e)$.

remedy:

**Edmonds-Karp alg.**

- variant of aug. flows.

- terminates in poly(m, n) steps, $m = |E|$, $n = |V|$, regardless of the capacities

"strongly polynomial time".

- Works even if capacities irrational.
- ✡ Unknown if ∃ strongly polynomial time for general LP's.
- **Algorithm**: same as before, but use <u>shortest</u> s-t path in residual. ← by # edges.

**Analysis** idea: show iterations <u>increase</u> s-t distance in residual.

- For $v \in V$, let $d_s(v)$ denote distance from $s$ to $v$ in $G_x$ (length of shortest $s \sim v$ path in $G_x$).

- Let $P$ shortest $s$-$t$ path in $G_x$.

  $P = \underset{v_0}{\underset{\shortparallel}{s}} - v_1 - v_2 \cdots - \underset{v_k}{\underset{\shortparallel}{t}}$, $d_s(v_j) = j$

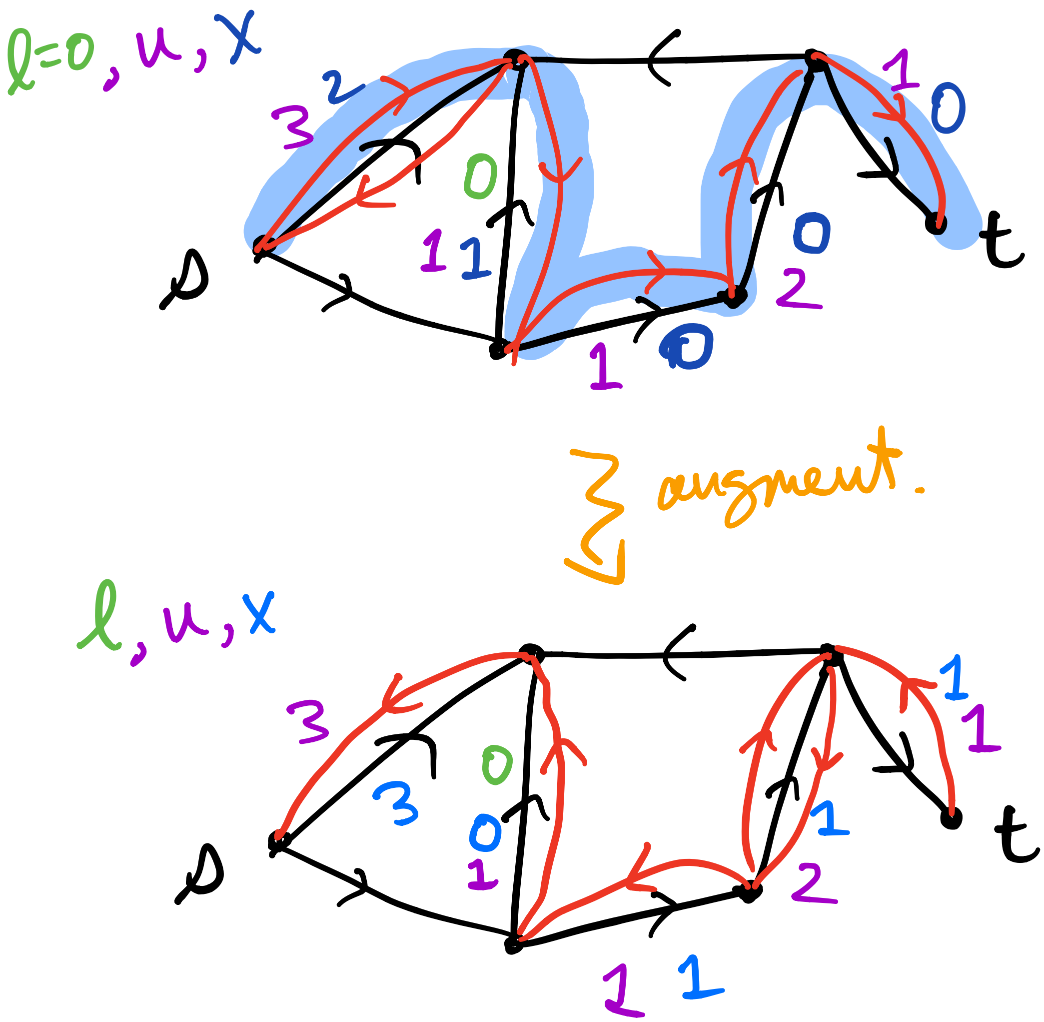
- $x'$ flow after augmenting along $P$ (as much as possible).

- Let $d_s'$ be distance labels for $G_{x'}$.

- **Note**: any edge $(i,j)$ added to $G_x$ in $G_{x'}$ goes opposite direction of $P$.

  $d_s(j) = d_s(i) + 1$ . $\Delta$

i.e. $d_s(i) = d_s(j) + 1$




- After augmenting,

  $d_s(j) - d_s(i) \le 1$ ☆

  for every edge $(i,j)$ in $E_{x'}$.

$\left( (i,j) \in E_x \Rightarrow \text{automatically} \; (i,j) \notin \; ds(j) - ds(i) = -1 \right)$ in $G_x$.

- for any $j \in V$, Sum along edges of shortest s-j path $P'$ in $G_{x'}$,

$$ds(j) = \sum_{(i,j) \in P'} ds(j) - ds(i) \le |P'| = ds'(j).$$

In particular, for $j = t$

$$\boxed{ds(t) \le ds'(t)}$$

- Distance to t can

increase $\le n-1$ times.

- **But how often must it increase?**

Each iteration, some edge with $ds(j) = ds(i) + 1$

is removed from $G_x$.

(P shortest path, & some edge along P must get removed).

- Thus after $\le m$ iterations must get $ds(t) < ds'(t)$.

(one ineq. in telescope becomes strict. ).

- **In summary!**

  (i) # augmentations $\le m(n-1)$

  (ii) time to build $G_x$, find $P = O(m)$.

  $\Rightarrow$ running time $O(m^2 n)$. $\square$

  not tight.

Best: $O(mn \cdot \log(m \dots n))$

Goldberg – Tarjan

---

# The initial feasible flow

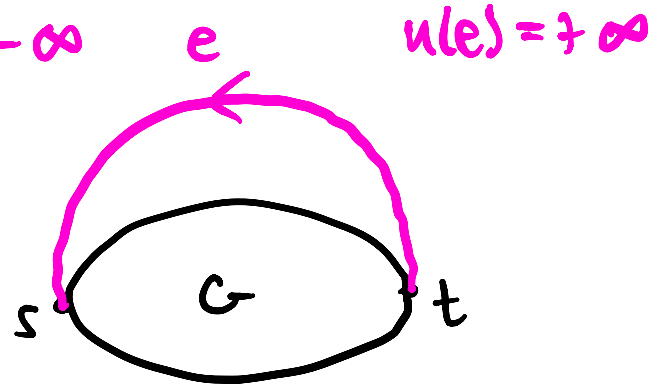
We still need to find flow to start with!

- if $\boxed{\ell(e) \le 0 \le u(e)}$ $\forall e$, use $x = 0$.
- if not, feasible flow is max flow for another network that's easy to initialize.

# Circulations

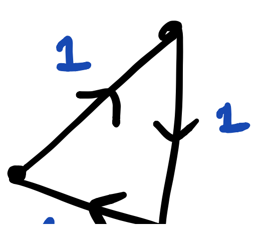
- first reduce finding feasible flow to finding "circulation" in new graph $\tilde{G}$:

$\tilde{G} =$

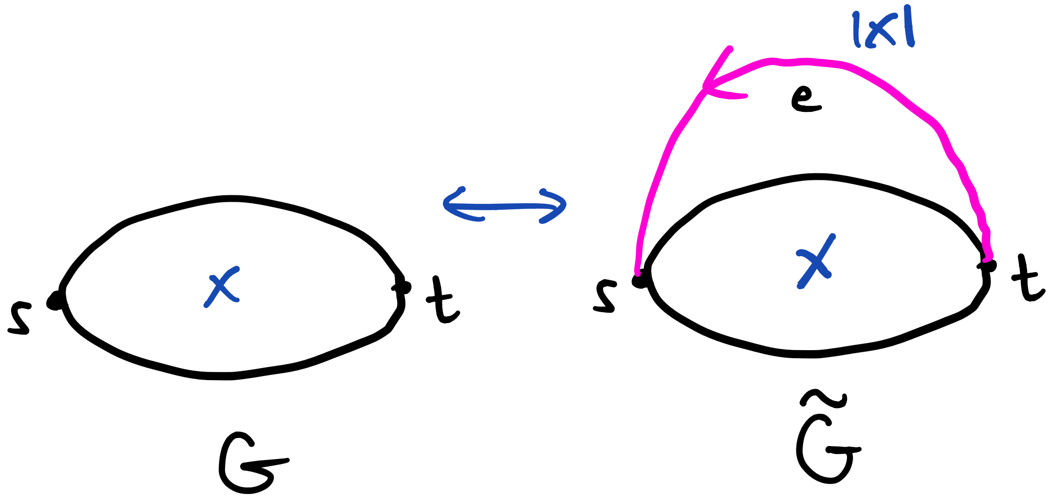



- Define **circulation** of $G, u, l$ as flow satisfying conservation at <u>all</u> $v \in V$ ($s, t$ no longer special).

e.g.



- Bijection between flows in $G$ & circulations in $\tilde{G}$:




( add $|x|$ to new edge ).

## finding Circulations!

Let $G=(V,E)$ arbitrary digraph w/ capacities $l, u$. $\boxed{l \le u}$

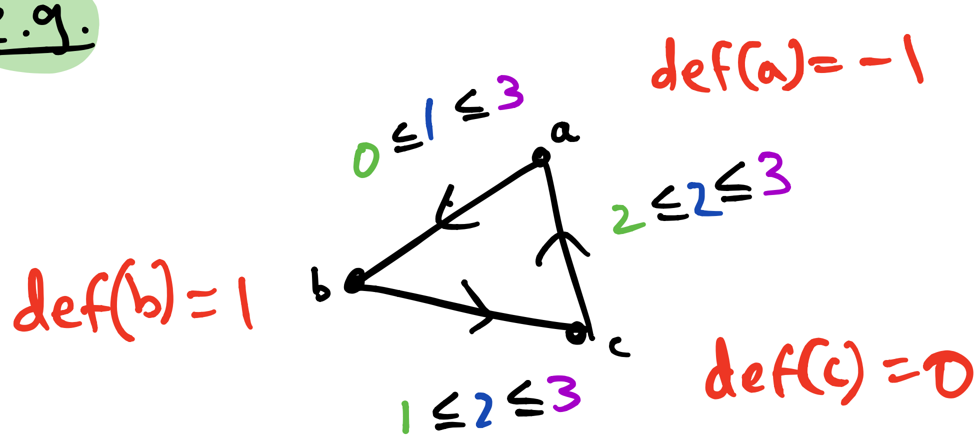
- First, choose arbitrary $y_e$ with

with $\boxed{l(e) \le y_e \le u(e)}$.

- $y_e$ need not be flow; define **deficit** at $v$

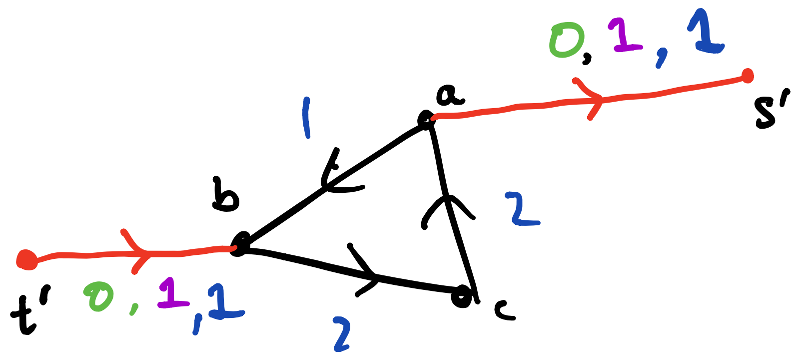
$$\text{def}(v) := \sum_{\delta^+(v)} y_e - \sum_{\delta^-(v)} y_e$$

e.g.




- To fix: add extra edges, source, sink to supply deficit.

e.g.



**Formally**: Let $G' = (V', E')$ with $V' = V \cup \{s', t'\}$.

(i) add two vertices $s', t'$

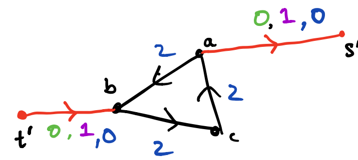
(ii) Let $V^+ = \{v : \text{def}(v) > 0\}$.

$V^- = \{v : \text{def}(v) < 0\}$.

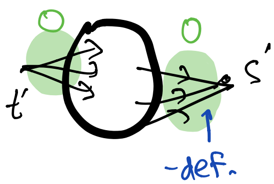
(iii) For $v \in V^+$, add edge $e = (t', v)$ with $l(e) = 0$, $\boxed{u(e) = \text{def}(v).}$

For $v \in V^-$, add $e = (v, s')$ w/ $l(e) = 0$, $\boxed{u(e) = -\text{def}(v)}$.

- Setting flow on new edges equal to upper capacities gives feasible flow for network $G'$ with source $s'$, sink $t'$.




- initial value is




$$\boxed{\sum_{v \in V^-} \mathrm{def}(v) < 0.}$$

- Using this <u>initial flow</u>, apply Edmonds-Karp in $G'$ to find max flow $x$;

  note $|x| \leq 0$.

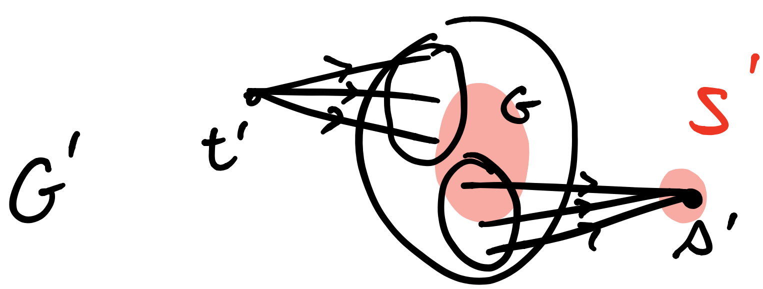
- If $|x| = 0$, restricting $x$ to $E$ gives circulation. (all new edges have 0 flow).
- If $|X| < 0$, then no circ. exists (if it did, set flows to circ. values on old edges, flow 0 on new, $\rightsquigarrow$ flow w/ value 0, contra.

diction).

- **In summary**: to find feas. flow in $G$, find circulation in $\tilde{G}$ by solving max flow in $\tilde{G}'$. (if max flow in $\tilde{G}' < 0 \Rightarrow$ no circ. in $\tilde{G} \Rightarrow$ no feas. flow in $G$.)

## When is a flow network feasible?

- Enough to decide if there's a circulation. (in $\tilde{G}$)
- Use max-flow min-cut in $G'$.




- $s'$-$t'$ cut in $G'$ is $S' = S \cup \{s'\}$, $S \subseteq V$.
- MFMC $\Rightarrow$ maxflow $= 0 \Leftrightarrow c_{G'}(S') \geq 0$

- Capacity is $\forall$ $s'$-$t'$ cuts $S'$.

$$\boxed{c_{G'}(\underbrace{S \cup \{s'\}}_{S'}) = c_G(S) = \sum_{\delta^+(S)} u(e) - \sum_{\delta^-(S)} \ell(e).}$$

(because lower caps all 0 of new edges,

all new edges go <u>into</u> $S'$.).

To summarize:

**Theorem** $G, \ell, u$ admits circulation iff $\forall S \subseteq V$,

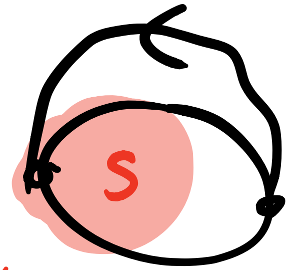
$$\boxed{\sum_{\delta^+(S)} u(e) - \sum_{\delta^-(S)} \ell(e) \geq 0.}$$ & $\ell \leq u$

**Corollary** Flow network feasible iff $\ell(e) \leq u(e) \; \forall e$ and

$\|$

$$\sum_{\delta^+(S)} u(e) - \sum_{\delta^-(S)} l(e) \geq 0$$

$\forall S$ s.t. $\boxed{|S \cap \{s,t\}| \neq 1}$.




$c(S)=\infty$.

Pf: apply theorem to $\tilde{G}$.

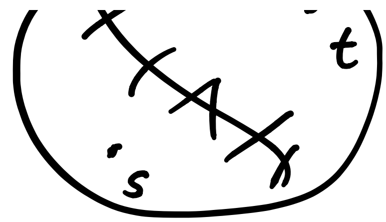
# (~~General~~ Global) min cut.

- Assume now $l=0$, so cut capacity is just
$$u(\delta^+(S)) := \sum_{\delta^+(e)} u(e).$$

- we've shown how to find min s-t cut using max-flow.

- Can also solve

$$\min_{s\text{-}t \text{ cuts } S} u(\delta(S))$$




in undirected graph by


- What about *global* minimum cut (not for fixed $s, t$)

- Can reduce to $2(n-1)$ maxflows:
  - (i) choose arbitrary vertex $s$
  - (ii) for any $t \in V \setminus \{s\}$,
    solve for min $s$-$t$ cut,
    min $t$-$s$ cut,
    take whichever is smaller.

do this for all $t$.

- Fastest maxflow algs take around $\tilde{O}(mn)$ time (Goldberg-Tarjan), so our naive alg. takes $\tilde{O}(mn^2)$ time.

- Hao-Orlin used relationships between the $O(n)$ flow problems to give an $O\left(mn \log\left(\frac{n^2}{m}\right)\right)$ time alg for global mincut.

---

Today: • different alg. for

**Today**: undirected graphs.

- Not using max flow
- Comparable runtime to Hao-Orlin

uses property of 'diminishing returns', a.k.a submodularity

---

**Setup**: • Let $G=(V,E)$ undirected,

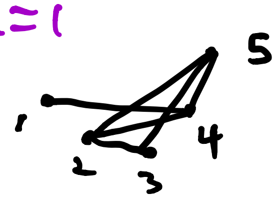
• $u: E \to \mathbb{R}_{\geq 0}$ nonneg. edge costs.

---

**Algorithm idea**:

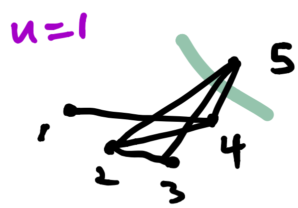
- starting with arbitrary vertex, build "max adjacency ordering", i.e. greedily add the vertex w/ min cost to previous ones.

**e.g.** $u=1$

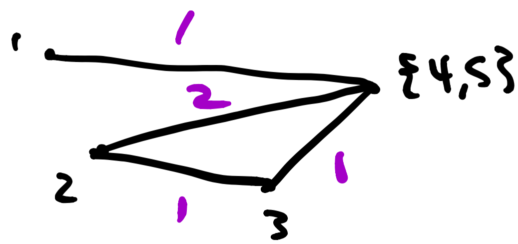


- Consider cut from last vertex, and also cuts obtained by shrinking last two vertices. (& recursing)

**e.g.**




and cuts in



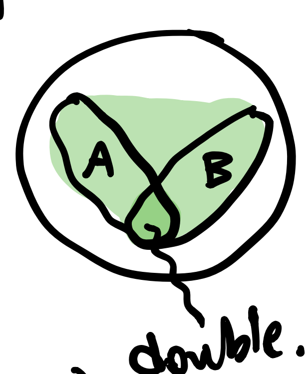

(duplicate edges get the sum of cost.)

- **Claim:** best cut found this way is global min cut.

---

**Def:** For $A, B \subseteq V$, define

$$u(A:B) := \sum_{\substack{i \in A \\ j \in B}} u((i,j)).$$




**Algorithm** (Stoer-Wagner)

MINCUT($G$) # outputs cut.

▷ let $v_1$ any vertex of $G$

▷ $n := |V(G)|$

# create ordering

▷ initialize $S =$

▷ **for** $i = 2 \dots n$:

  ▷ $v_i = \underset{v \in V \setminus S}{\arg\min}\; u(S : \{v\})$

  ▷ $S \leftarrow S \cup \{v_i\}$

▷ **if** $n = 2$:

  ▷ **return** $\delta(\{v_n\})$.

▷ **else**:

  ▷ Get $G'$ by shrinking $\{v_{n-1}, v_n\}$

  # recursive call

▷ Let $C = \text{MINCUT}(G)$

▷ return less costly of $C$, $\delta(\{v_n\})$.

**Analysis:** uses a claim.

**Claim:** $\boxed{\{v_n\} \text{ is a min } v_{n-1}\text{-}v_n \text{ cut.}}$

**Claim $\Rightarrow$ correctness:**

- The min cut is either a min $v_{n-1}$-$v_n$ cut, or not.
- If it is, claim $\Rightarrow$ alg outputs it ✓
- If not, by induction on $n = |V(G)|$, algorithm outputs mincut in G! ✓

## Proof of Claim:

Let $v_1, v_2, \ldots \ldots, v_{n-1}, v_n$ be the ordering from alg.

- $A_i :=$ sequence $v_1 \ldots\ldots v_{i-1}$

- Consider candidate $v_{n-1}$–$v_n$ cut, i.e. $C \subseteq V$ s.t.

$$v_{n-1} \in C, \; v_n \notin C.$$

- Want to show

$$u(\delta(A_n)) \leq u(\delta(C)),$$

''

$u(\delta(\{v_n\}))$

$u(\delta(\{v_n\}))$.

i.e. cut from $\{v_n\}$ is better than $C$.

- define $v_i$ to be **critical**

if either $v_i$ or $v_{i-1}$ in $C$ but not both.

$v_1, v_2, \dots, \dots, v_n$

$C$ critical

- **Subclaim:** Define $C_i := A_{i+1} \cap C$

If $v_i$ critical, then

$i=n$

$$\boxed{u(A_i : \{v_i\}) \le u(C_i : A_{i+1} \setminus C_i)}$$

$i=n$


replace these with

Subclaim suffices, because

subclaim $\Rightarrow \underbrace{u(\delta(A_n))}_{u(A_n : V_n)} \leq \underbrace{u(\delta(C))}_{u((C_n : A_{n+1}) \setminus C_n)}$

because $v_n$ is critical.

proof of subclaim:

by induction on seq. of critical vertices.

- (base:) true for first critical vertex.

- (inductive:) Assume true for critical $v_i$, let $v_j$ next critical.

- Then

$u(A_i : \{v_j\}) =$

$\leq$

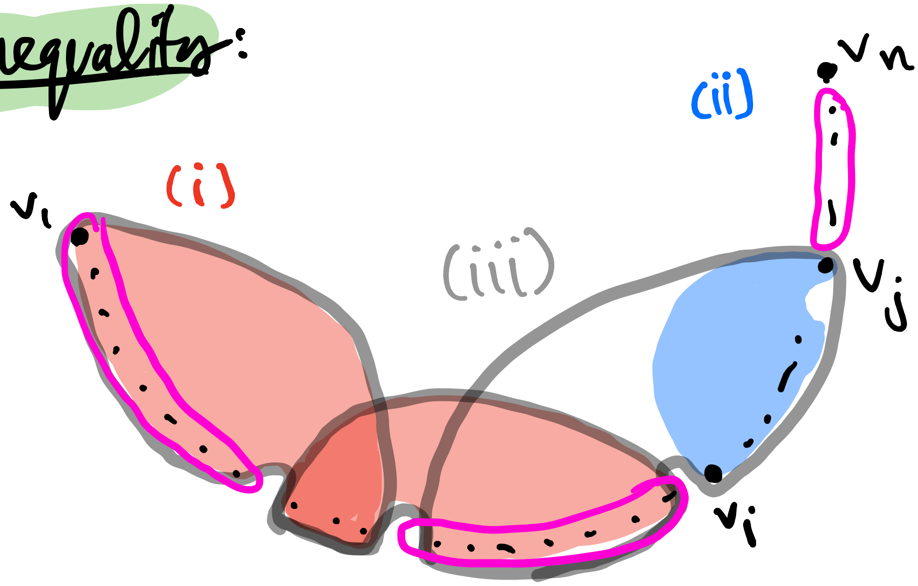
(i)

$\leq$

(ii)

$\leq u(C_j : A_{j+1} \setminus C_j)$. (iii)

**Last inequality:**




$\square$

## Running time:

Depends how you implement ordering:

- **Exercise:** each iteration can be done in $O(m + n\log n)$ time (use e.g. Fibonacci heaps.)
- overall, $\le n$ shrinks $\Rightarrow$ $O(mn + n^2 \log n)$ time.

side note:

## Submodularity

- Stoer-Wagner can be extended to minimize a more general class of functions than $S \mapsto u(\delta(S))$.
- function $f: 2^V \to \mathbb{R}$ submodular

- function $f: 2^V \to \mathbb{R}$ **submodular**

  if $\qquad \geq \qquad$.

- **Examples**:
  - $f(S) = \qquad ,$
  - $f(S) = \qquad$ for

    $u$ nonnegative, $G$ undirected

Submodularity equivalent to
"diminishing marginal returns":

For $S \supseteq T$, $v \notin S$,

- Above algorithm can be extended to minimize any **symmetric** ( ) submodular function.

↳ Queyranne '95.