Lecture 17

Approximation algorithms:
Max-Cut and Vertex Cover

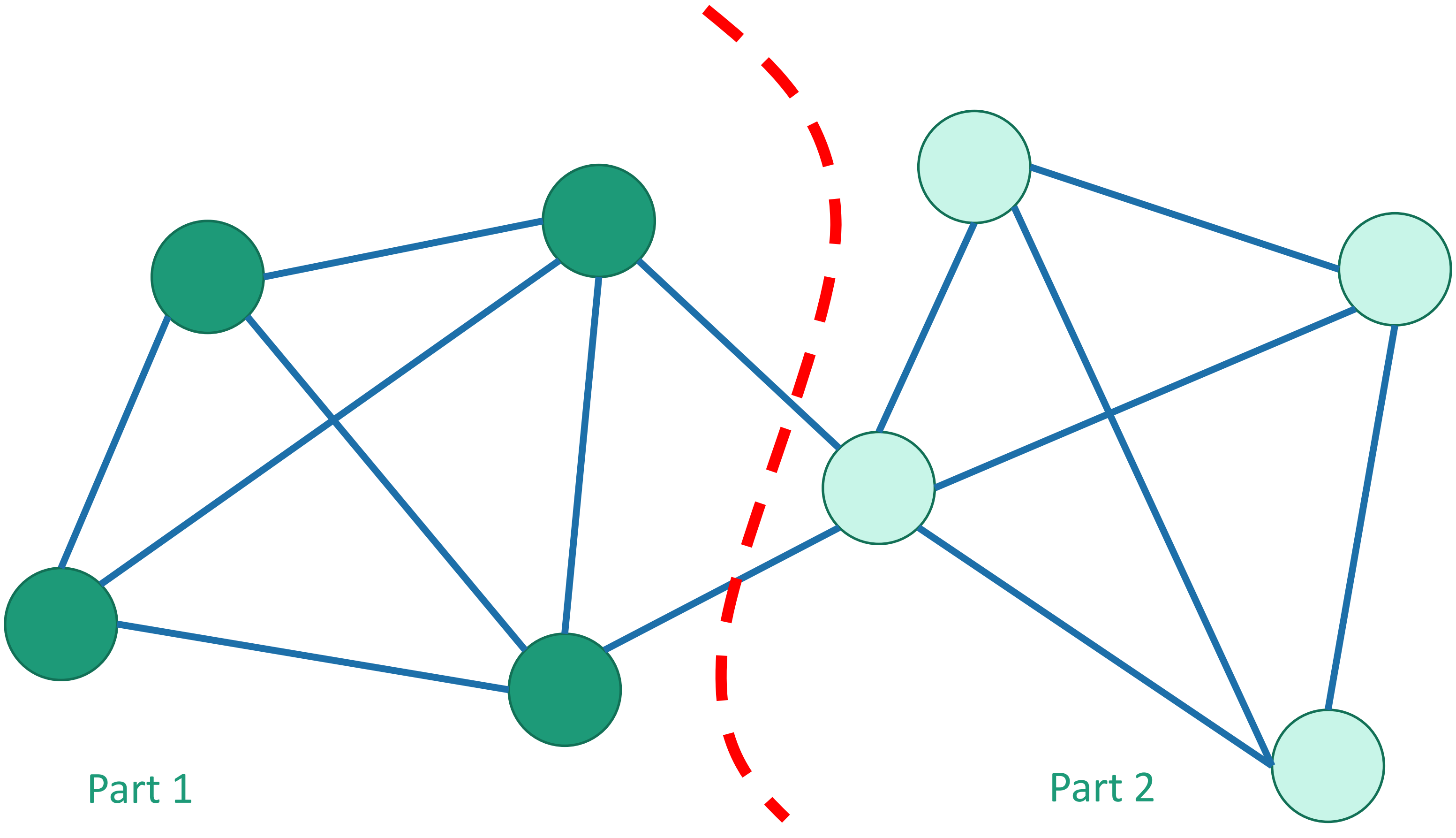
Last week

- Min-Cut problem

Algorithms

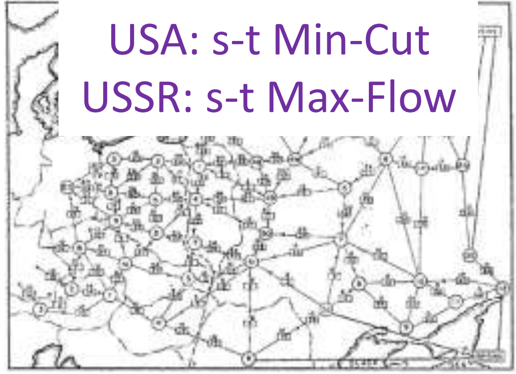
s-t min-cut: Ford-Fulkerson

Global min-cut: Karger-Stein



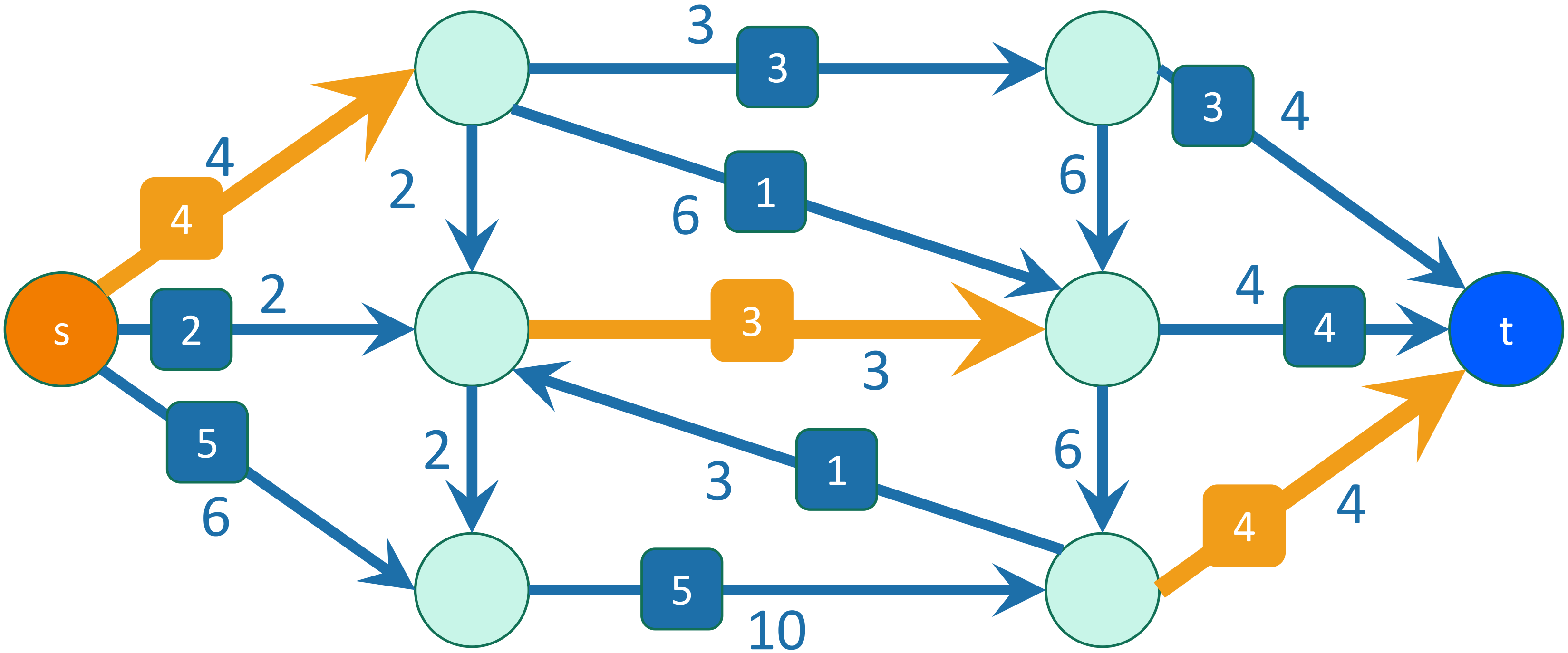
Last week: **s-t** Min-Cut

Max-flow min-cut theorem

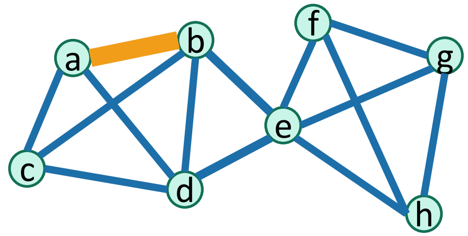


The value of a max flow from s to t
is equal to
the cost of a min s-t cut.

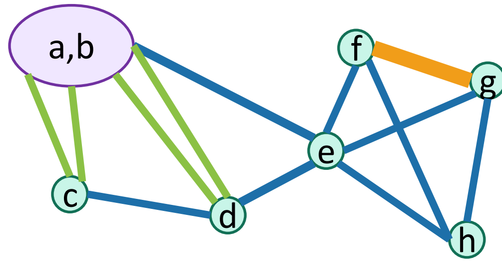
Intuition: in a max flow, the min cut better fill up, and this is the bottleneck.



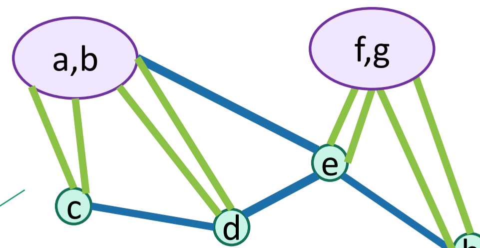
Last week:
Global Min-Cut



Contract!

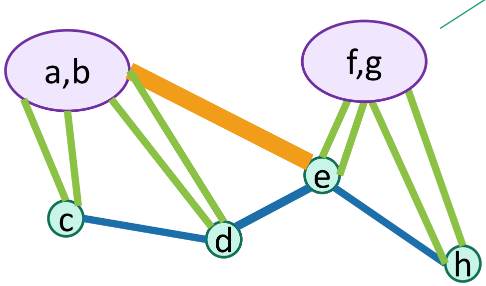


Contract!

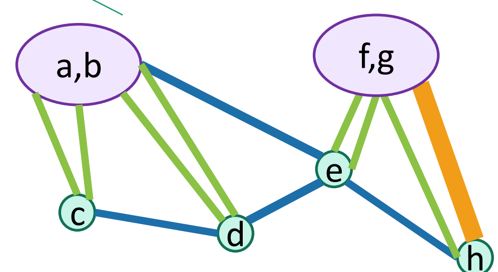


This branch made a bad choice.

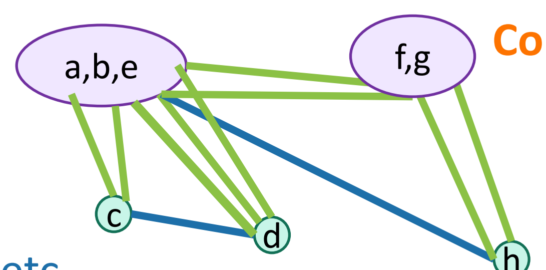
But it's okay since this branch made a good choice.



FORK!

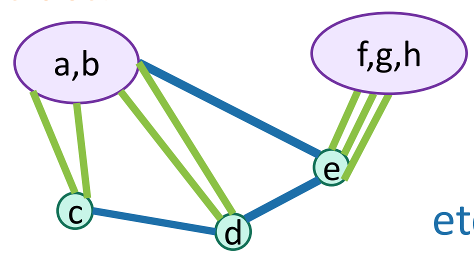


Contract!



etc

Contract!



etc

Last week:
Global Min-Cut: when things cancelled nicely

Suppose we contract n – t edges, until there are t supernodes remaining.

- Suppose the first n-t edges that we choose are

$$e_1, e_2, \ldots, e_{n-t}$$

- **PR**[none of the e_i cross S* (up to the n-t'th)]

$$= \left(\frac{\cancel{n-2}}{n}\right)\left(\frac{\cancel{n-3}}{n-1}\right)\left(\frac{\cancel{n-4}}{\cancel{n-2}}\right)\left(\frac{\cancel{n-5}}{\cancel{n-3}}\right)\left(\frac{\cancel{n-6}}{\cancel{n-4}}\right)\cdots\left(\frac{\cancel{t+1}}{\cancel{t+3}}\right)\left(\frac{t}{\cancel{t+2}}\right)\left(\frac{t-1}{\cancel{t+1}}\right)$$

$$= \frac{t\cdot(t-1)}{n\cdot(n-1)}$$ **Choose $t = n/\sqrt{2}$**

$$= \frac{\frac{n}{\sqrt{2}}\cdot\left(\frac{n}{\sqrt{2}}-1\right)}{n\cdot(n-1)} \approx \frac{1}{2}$$ when n is large

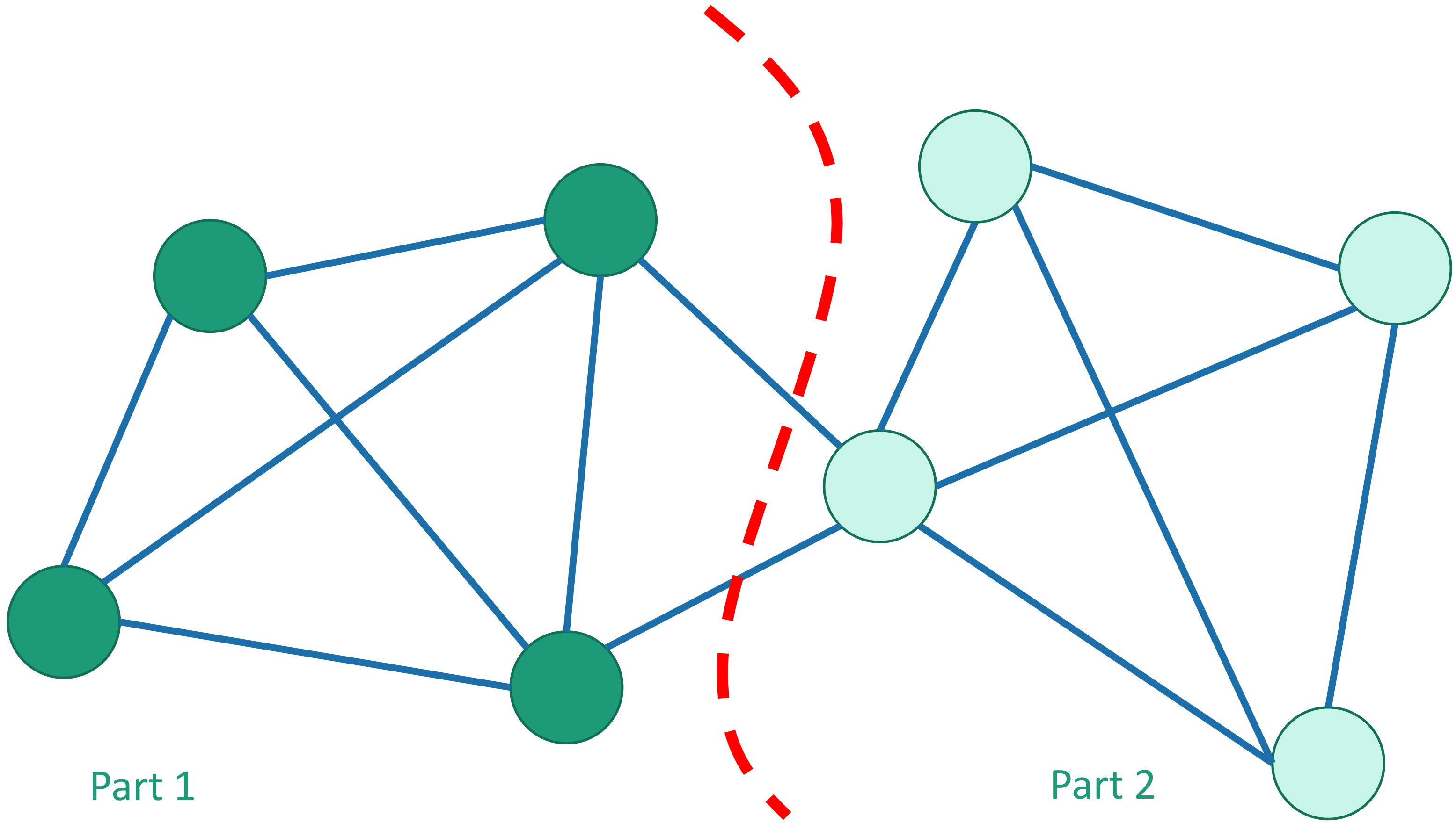
Last week

- Min-Cut problem

Algorithms

s-t min-cut: Ford-Fulkerson

Global min-cut: Karger-Stein



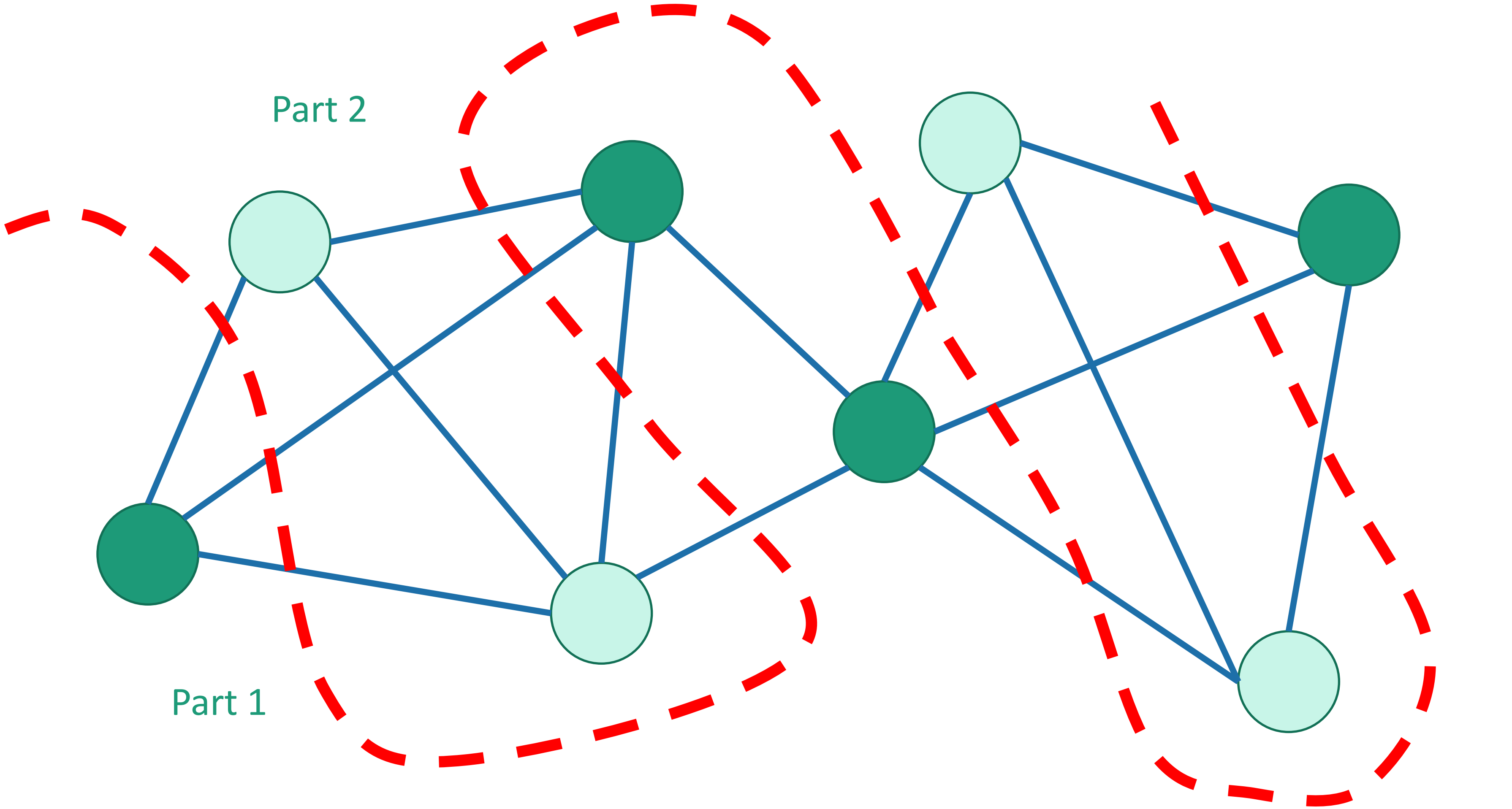
Today

<u>Possible Algorithms?</u>

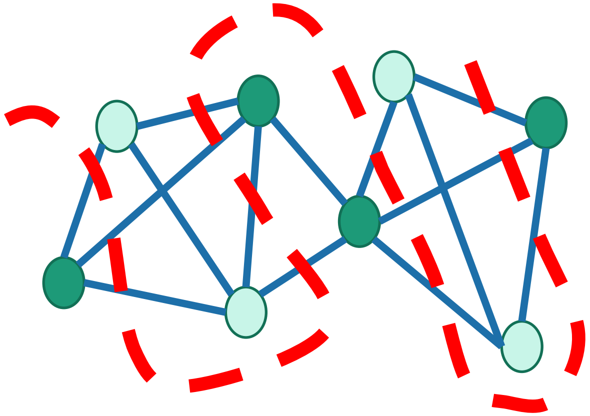
~~Max-Cut Min-Flow??~~

~~Contract random non-edge??~~

- ***Max***-Cut problem



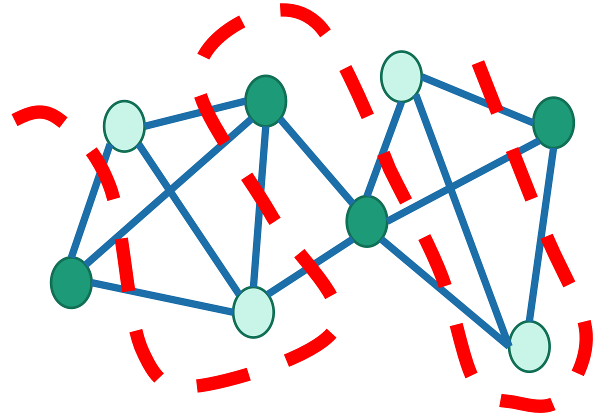
Max-Cut is NP-hard



- The Max-Cut problem is NP-hard
  - We won't formally discuss what this means (CS103, CS154, etc)
  - But we are unlikely to find efficient algorithms ("unless P=NP")
  - So there are no efficient algorithms ☹

Oh well, see you on Thursday!

Max-Cut is NP-hard



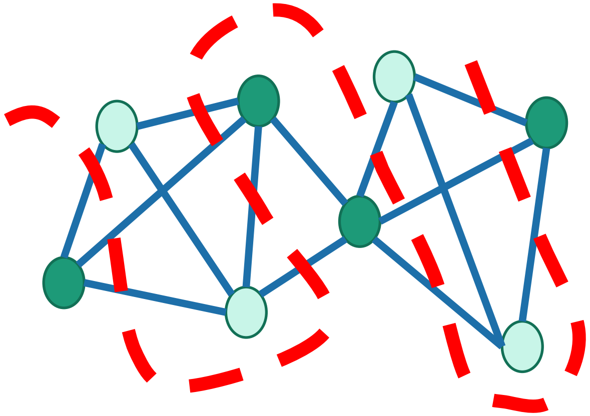
Coping with NP-hardness

- Option 1: making more assumptions on the inputs
  - In the DP lecture, we saw an algorithm for
    Maximal Weight Independent Set *on trees*
  - Max-Cut has an efficient algorithm if the graph is "*planar*"
  - In practice, a common assumption is that real instances are "*nice*"

What does "nice" mean?

Plucky the pedantic penguin

Max-Cut is NP-hard

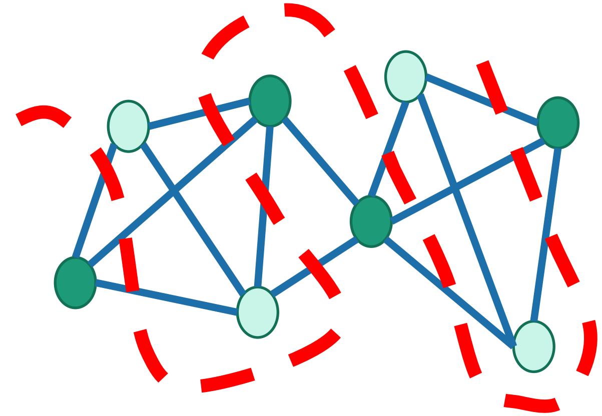


Coping with NP-hardness

- Option 1: Making more assumptions on the inputs
- Option 2: **Approximation algorithms**

(we'll see more on those today!)

Max-Cut is NP-hard



Coping with NP-hardness

- Option 1: Making more assumptions on the inputs
- Option 2: **Approximation algorithms**
- Option 3: *Sometimes* it's OK to run in exponential time

  (Only when it's exponential in a small number!)

Approximation Algorithms

Suppose we have an optimization problem $\boldsymbol{P}$

$$\max_x f(x)$$

With optimal solution x^*.

We say that algorithm $\boldsymbol{A}$ is α-approximation for $\boldsymbol{P}$
if it returns solution y such that

$$f(y) \geq \alpha \cdot f(x^*)$$

(Note that always $f(y) \leq f(x^*)$, so $\alpha \leq 1$.)

Approximation Algorithms (min)

Suppose we have an optimization problem $\boldsymbol{P}$

$$\min_x f(x)$$

With optimal solution x^*.

We say that algorithm $\boldsymbol{A}$ is α-approximation for $\boldsymbol{P}$ if it returns solution y such that

$$f(y) \leq \alpha \cdot f(x^*)$$

(Note that always $f(y) \geq f(x^*)$, so $\alpha \geq 1$.)

Remember:
α closer to 1 is better

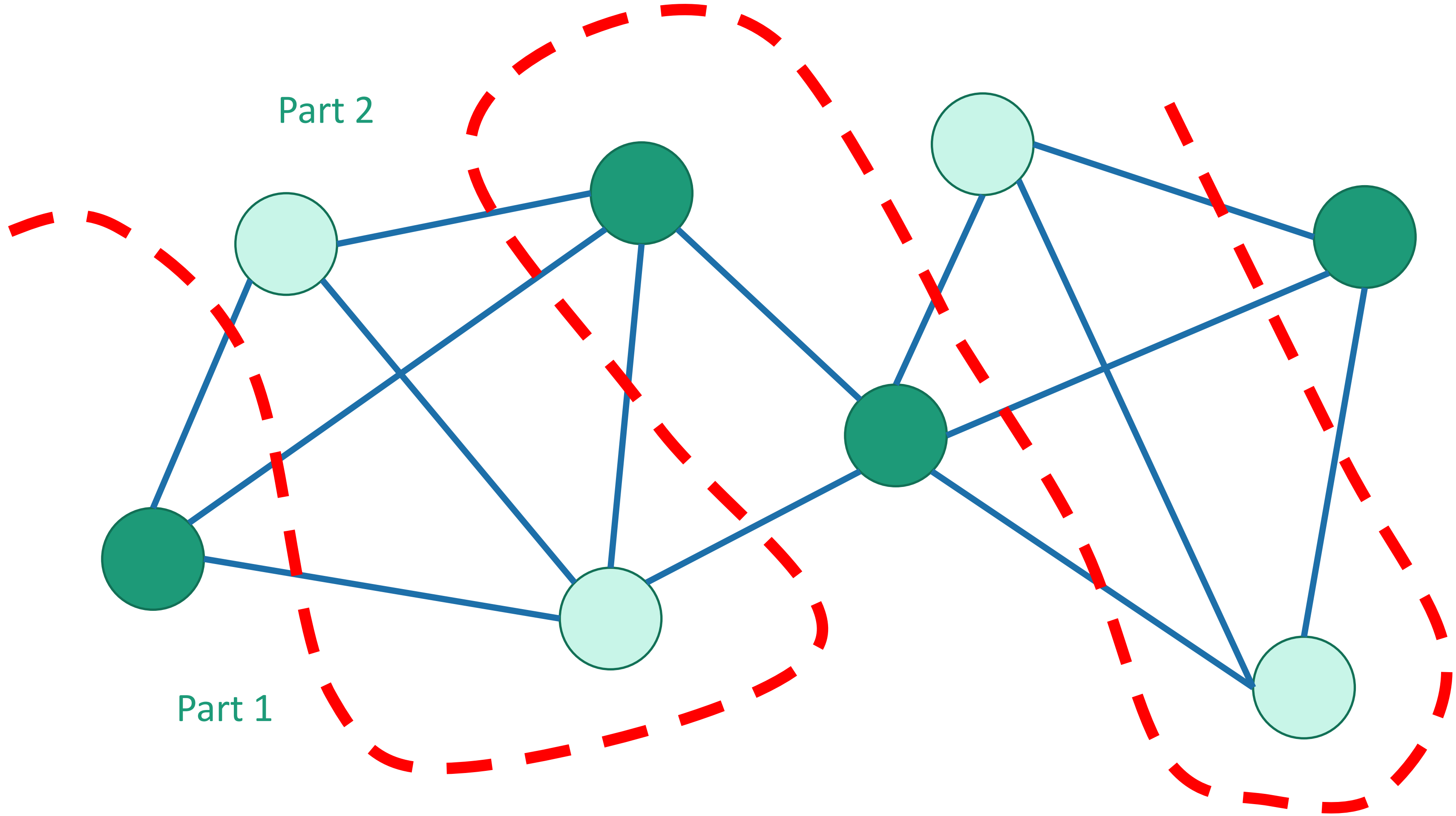
Today

Approximation Algorithms

Greedy

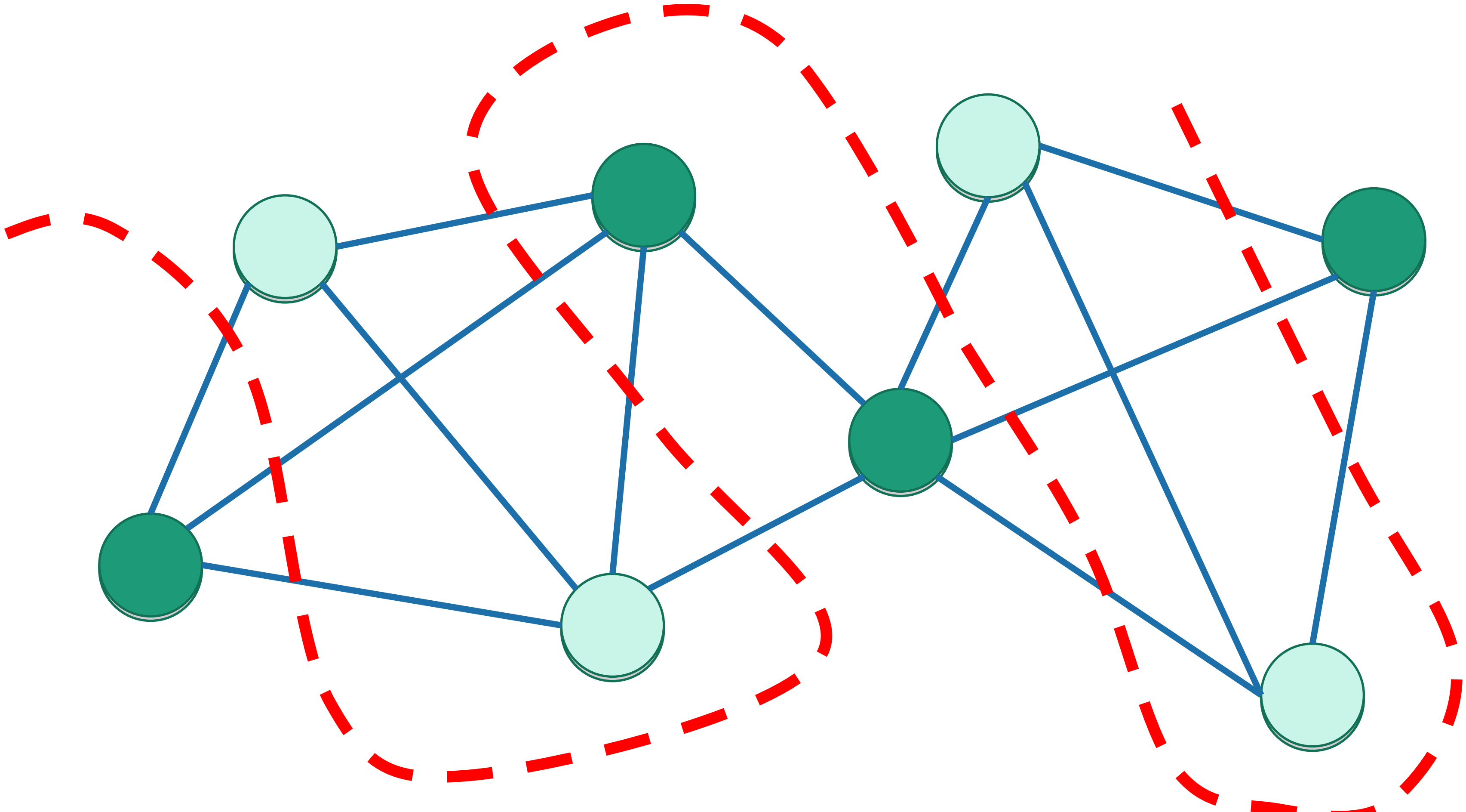
Randomized

- ***Max***-Cut problem

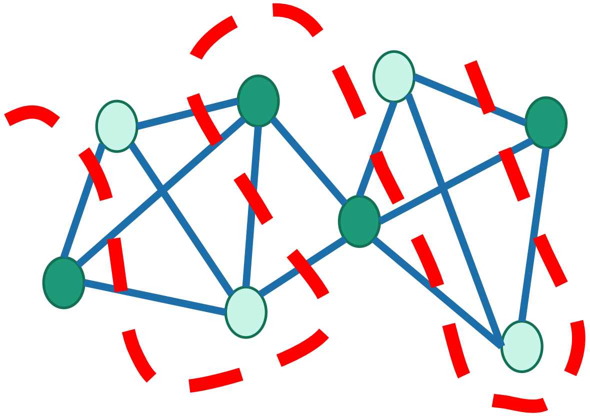


Algorithm 1: Greedy

(Iteratively add vertices
to the side that maximizes the cut)



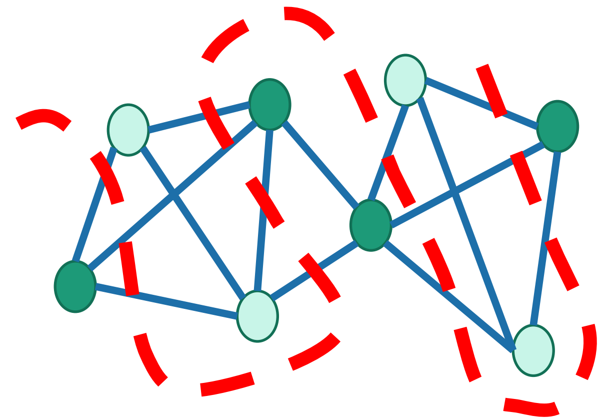
Algorithm 1: Greedy



Greedy_MaxCut$(G = (V, E))$:

- $A, B \leftarrow \emptyset$
- For i in V:
  - If (# of edges from i to A) $>$ (# of edges from i to B)
    - $B \leftarrow B \cup \{i\}$
  - Else
    - $A \leftarrow A \cup \{i\}$
- Return A, B

Algorithm 1: Greedy



Does it work?

- Claim: Greedy_MaxCut is a 1/2-approximation algorithm.

Think-Pair-Share:
Why is the claim true?
Bonus: how would you *prove* it?
2X Bonus: what about more than 1/2?

Greedy_MaxCut$(G = (V, E))$:

- $A, B \leftarrow \emptyset$
- For i in V:
  - If more edges from i to A
    - $B \leftarrow B \cup \{i\}$
  - Else
    - $A \leftarrow A \cup \{i\}$
- Return A, B

Greedy: 1/2-approximation

Claim: Greedy_MaxCut is a 1/2-approximation algorithm.

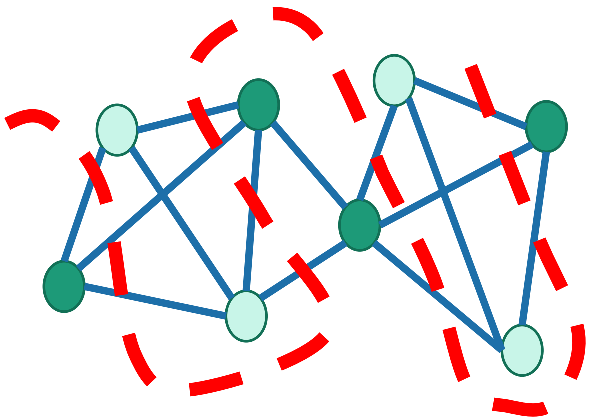
Proof:
At iteration i, at least 1/2 of the edges from i to $A \cup B$ are added to the cut.

In total, the cut returned by Greedy_MaxCut contains at least 1/2 of all the edges in the graph.

Therefore, the cut returned by Greedy_MaxCut contains at least 1/2 as many edges as the optimum.

$$ALG \geq |E|/2 \geq OPT/2$$

Algorithm 1: Greedy

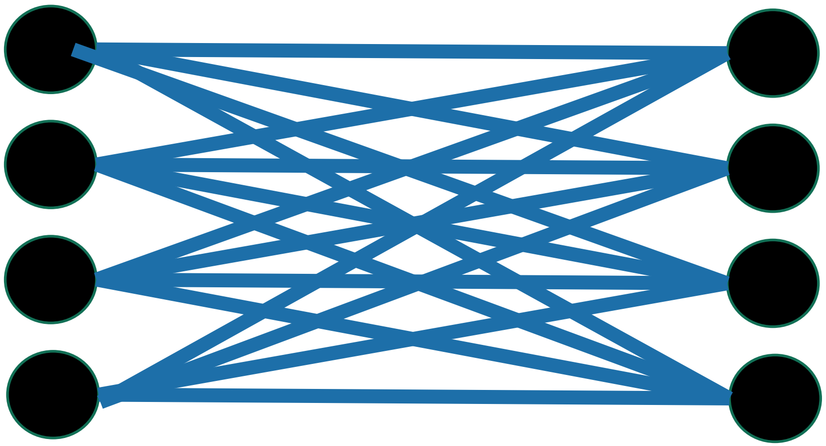


Does it work?

- Claim: Greedy_MaxCut is a 1/2-approximation algorithm.
- What about better than 1/2-approximation?

Can Greedy beat 1/2?

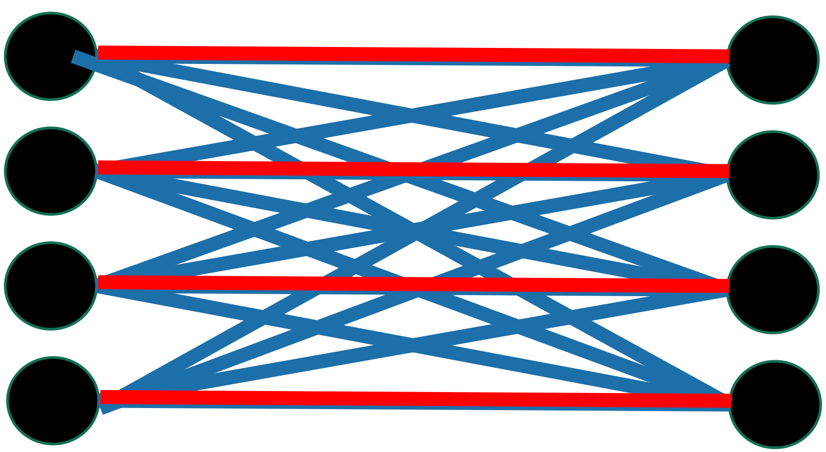
This is the complete bipartite graph.
In this graph, all the edges belong to max-cut,
and the greedy solution will always be optimal!



Size of optimal cut = $(n/2)^2$

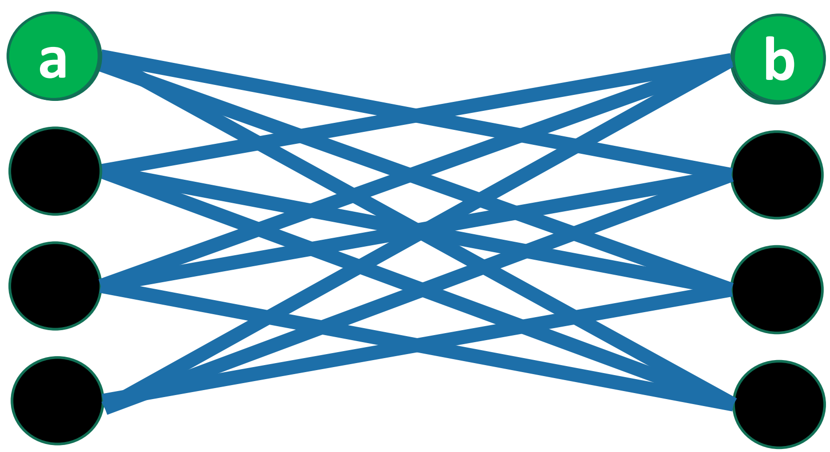
Can Greedy beat 1/2?

What happens when we remove 1 edge for each vertex?



Can Greedy beat 1/2?

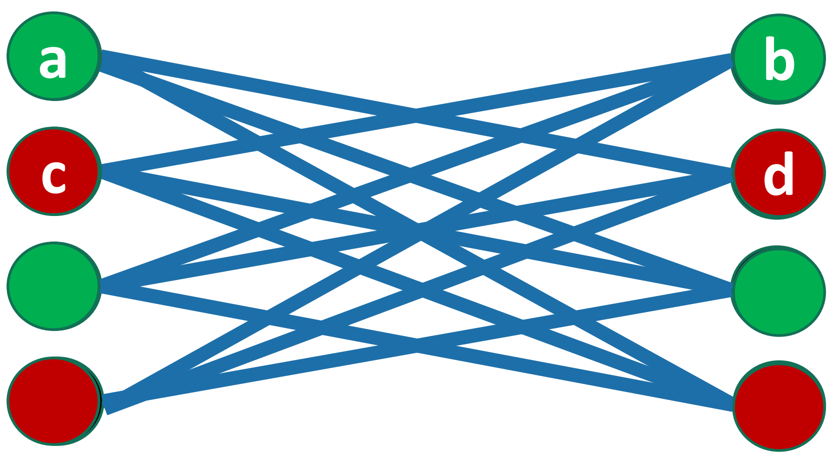
Suppose that Greedy first considers vertices a,b:
Since we haven’t seen any edges, both will be added to A



Conclusion:
Greedy's approximation factor is
$1/2 + O(1/n)$ $\square$

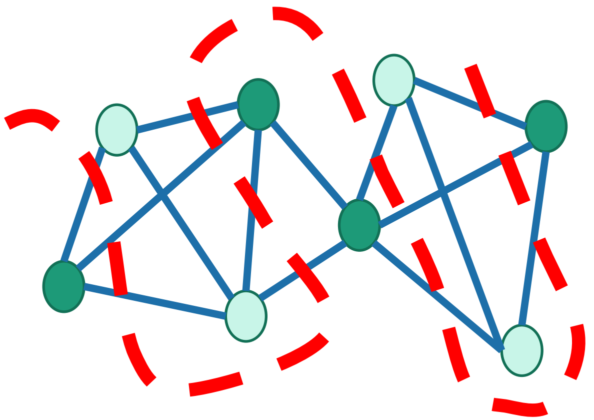
Can Greedy beat 1/2?

If the algorithm now proceeds to c,d,
it'll add them to $\boldsymbol{B}$
… and so on



Size of optimal cut = $(n/2)^2 - n/2$
Size of Greedy's cut =$(n/2)^2/2$

Algorithm 1: Greedy



Does it work?

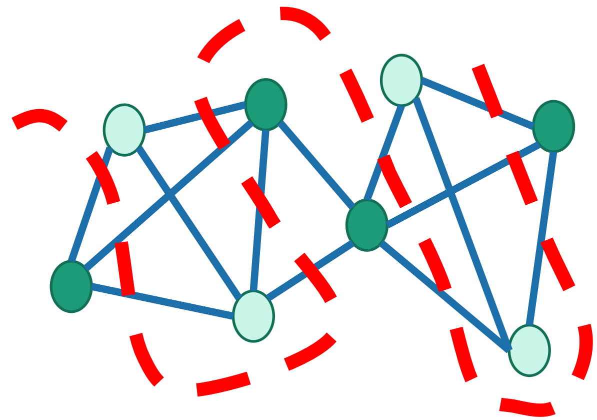
- Claim: Greedy_MaxCut is a 1/2-approximation algorithm
- Claim: Greedy_MaxCut is not much better than 1/2-approx
- Can other algorithms get better approximation?

Goemans-Williamson Algorithm

(You're not responsible for this rectangle)

- Approximates Max-Cut to within 0.878-factor
- Uses *Semi-Definite Programming (SDP)*
- Runs in polynomial time
- Whether 0.878 is optimal is an important open problem ("Unique Games Conjecture")

Algorithm 1: Greedy



Does it work?

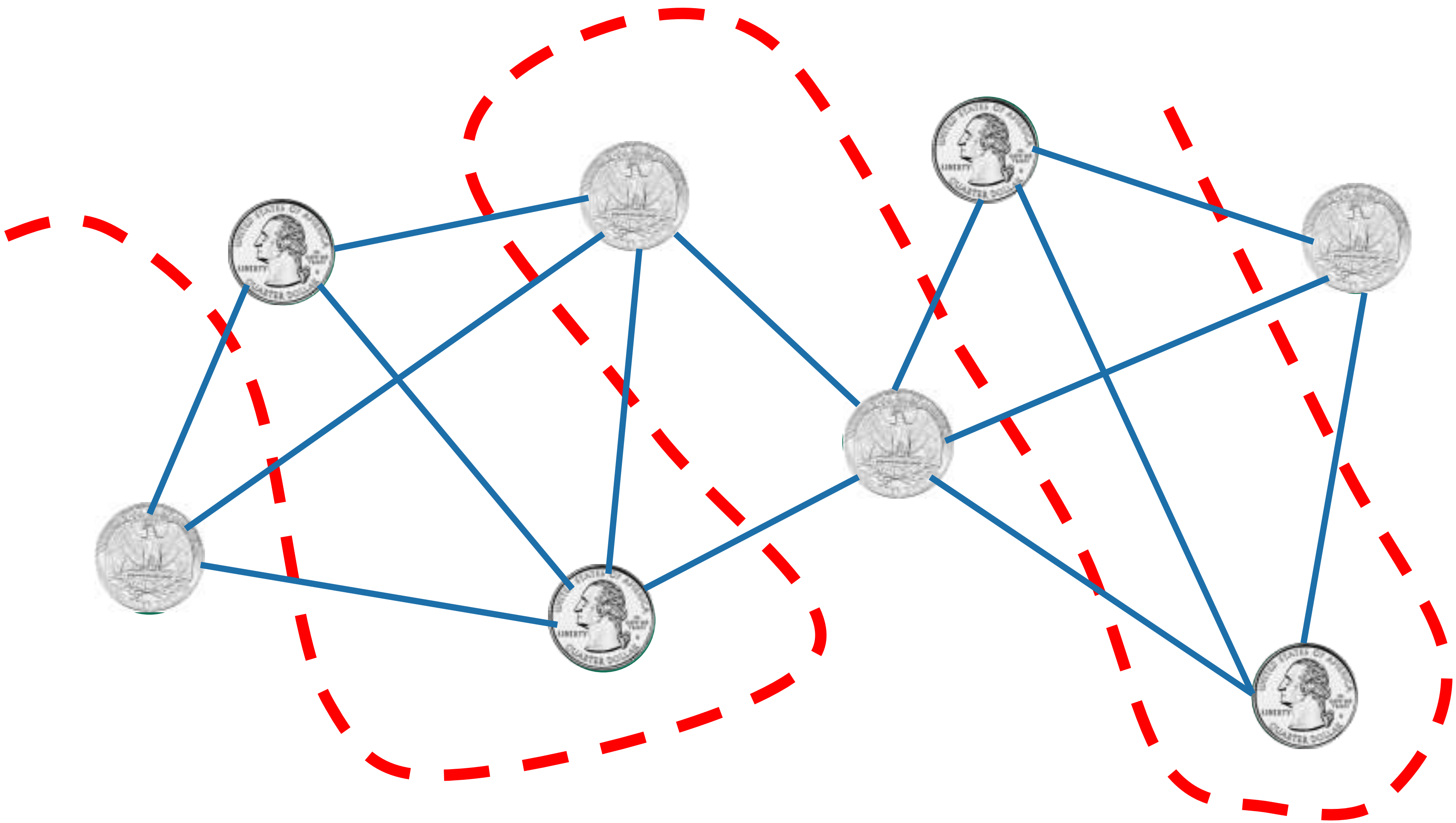
- Claim: Greedy_MaxCut is a 1/2-approximation algorithm
- Claim: Greedy_MaxCut is not much better than 1/2-approx

Is it fast?

- Yes! $O(n+m)$
- Can we do faster?

(Note: faster algorithms can’t even read the entire input!)

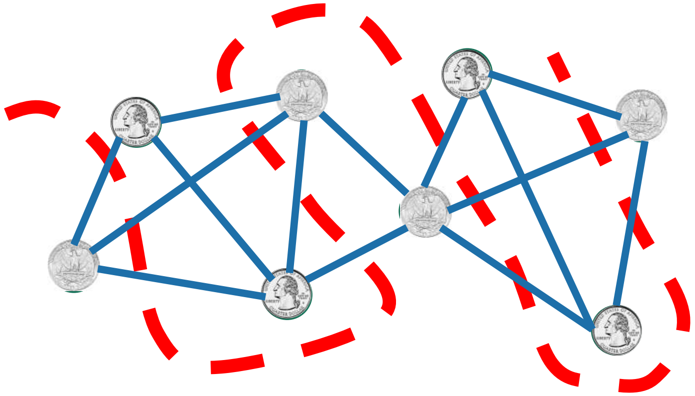
Algorithm 2: Random cut!



Algorithm 2: Random cut!

Random_MaxCut$(G = (V, E))$:

- $A, B \leftarrow \emptyset$
- For i in V:
  - Add i to A or B at random
- Return A, B

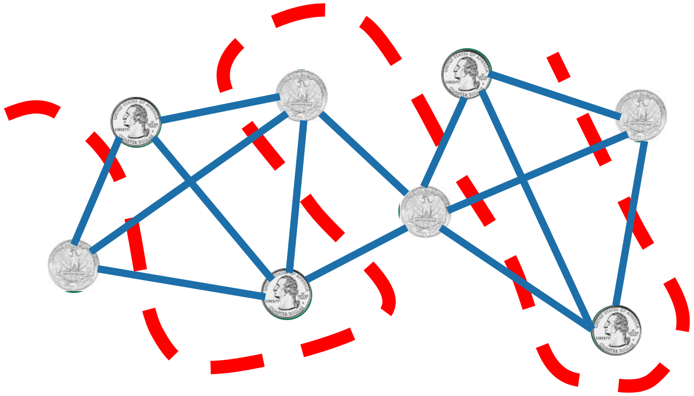


Algorithm 2: Random cut!

Does it work?

- Claim: Random_MaxCut is a 1/2-approximation algorithm ***in expectation***.

Think-Pair-Share:
Prove the claim!



Random: 1/2-approximation

Claim:
Random_MaxCut is a 1/2-approximation algorithm ***in expectation***.

Proof:
Every edge has probability exactly 1/2 of crossing the cut

In expectation, the cut returned by Random_MaxCut contains exactly 1/2 of all the edges in the graph.

Therefore, in expectation, the cut returned by Random_MaxCut contains at least 1/2 as many edges as the optimum.

$$\mathrm{E}[ALG] \geq |E|/2 \geq OPT/2$$

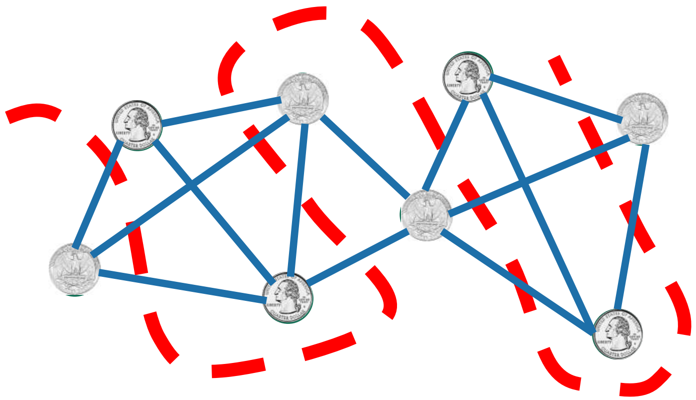
Algorithm 2: Random cut!

Does it work?

- Claim: Random_MaxCut is a 1/2-approximation algorithm ***in expectation***.

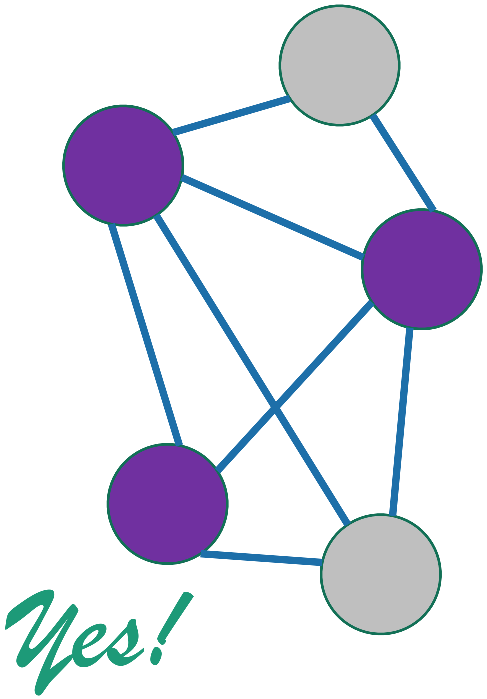
Is it fast?

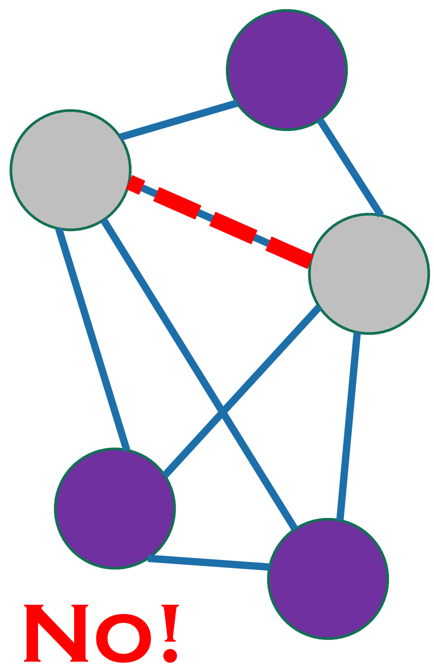
- *Very* fast, $O(n)$ time!
  (Faster than reading the input!)

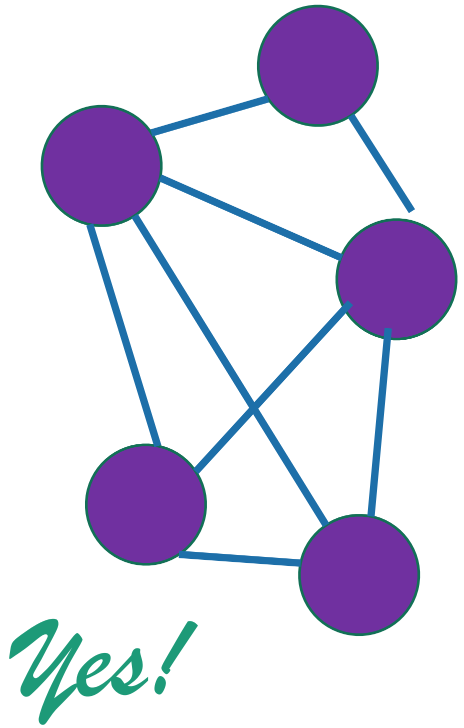


Vertex Cover

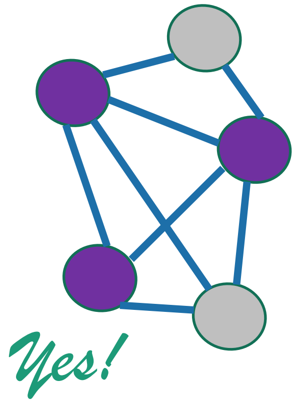
Def'n:
A *vertex cover* is a set S of vertices that "covers" all the edges (i.e. each edge has an endpoint in S).







Vertex Cover Problem



Input: $G = (V, E)$

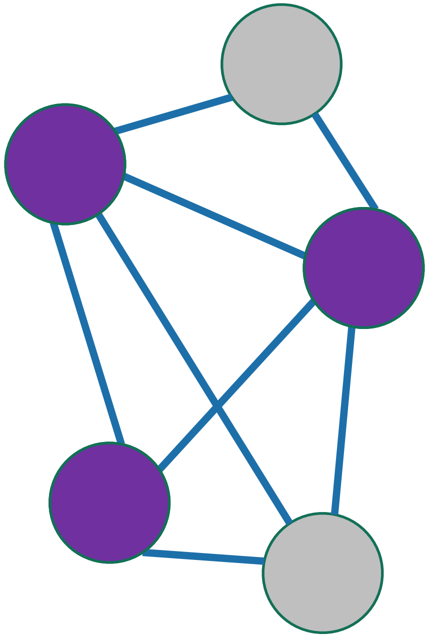
Output: Smallest vertex cover $S \subseteq V$

Obstacle: This problem is also NP-hard

Today: Approximation algorithm!

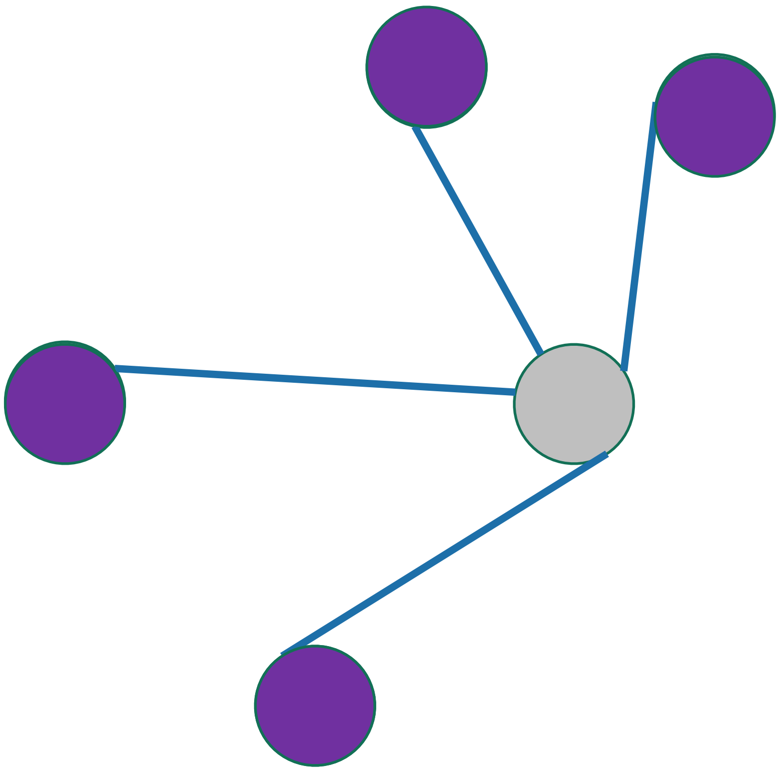
Think-Pair-Share:
Design an approx. algorithm!

Attempt 1: Greedy



Looks good! ☺

Attempt 1: Greedy



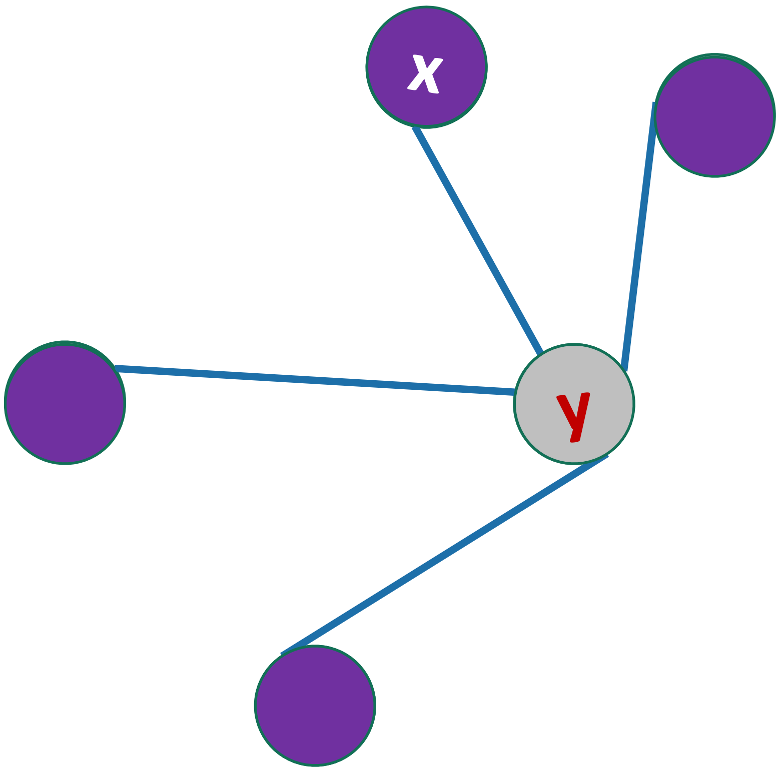
How about this graph?

Conclusion:
Greedy's approximation factor is
$\Omega(n)$

Since this is a min problem,
the approx. factor is > 1.
(We still want to be close to 1)

Plucky the
pedantic penguin

Attempt 1: Greedy



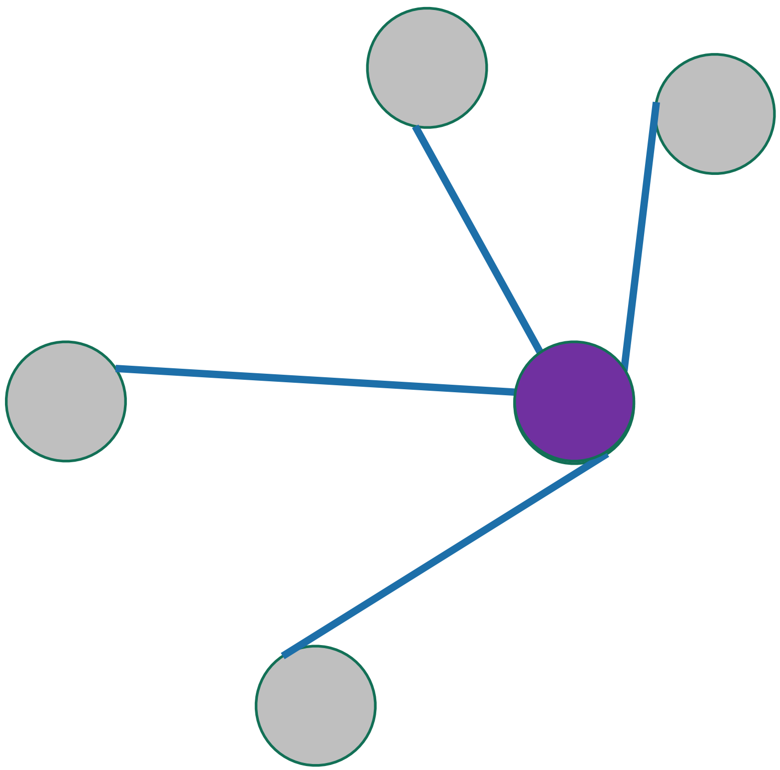
What went wrong?

Consider edge (**x**,**y**) -
we have two choices:
Add **x** or **y** to the cover.

We probably should've picked **y**
since it has higher degree
(so it covers more edges)

This suggests another
greedy algorithm…

Attempt 2: DegreeGreedy



DegreeGreedy $(G = (V, E))$:

- $S \leftarrow \emptyset$
- While E isn't empty
  - $v^* \leftarrow \arg\max_{v \in V} \deg(v)$
  - Add v^* to S
  - Remove v^*'s edges from E
- Return S

Looks good…?

Attempt 2: DegreeGreedy

[On whiteboard:
example where DegreeGreedy
has a bad approximation]

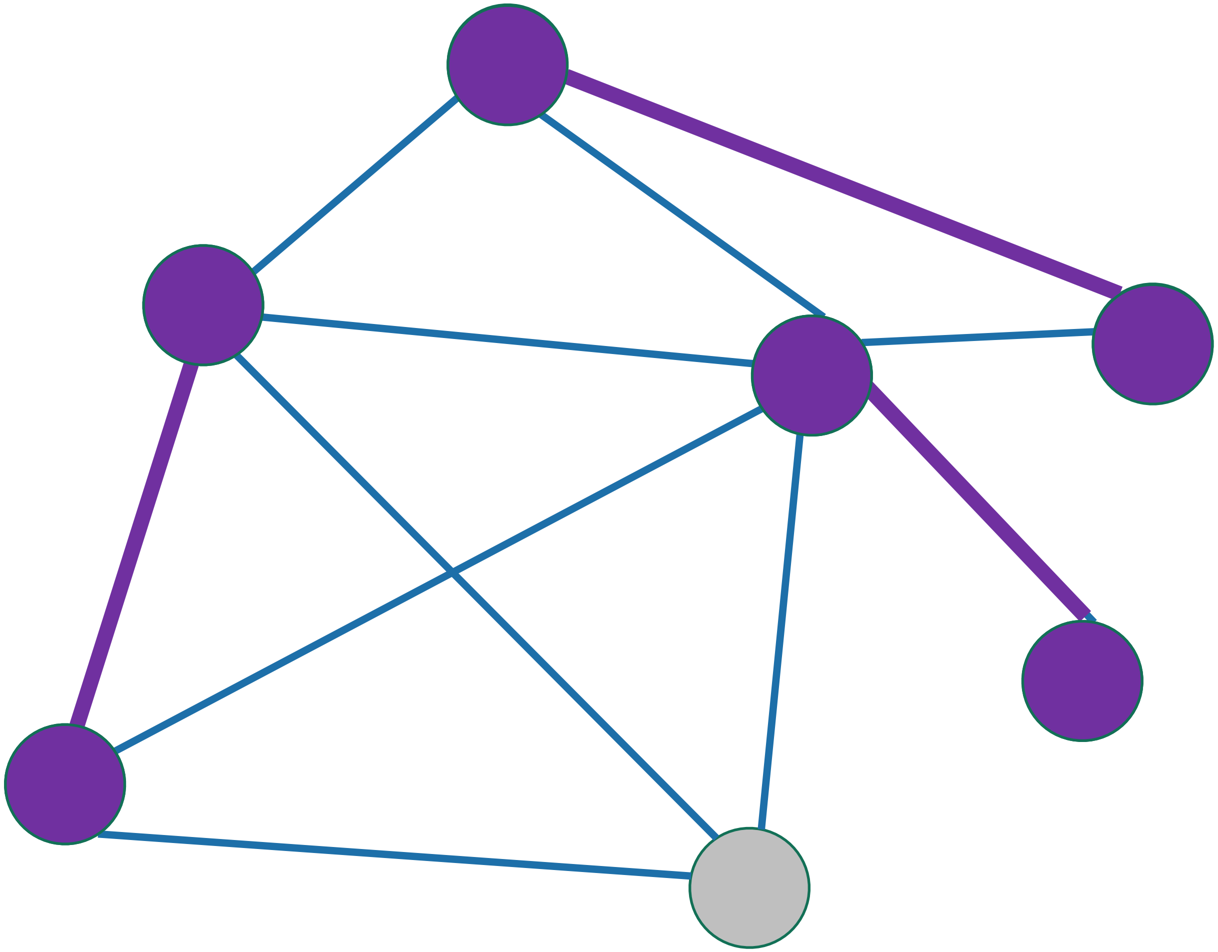
Conclusion:
Greedy's approximation factor is
$\Omega(\log n)$

What went wrong?

Consider edge (**x**,**y**) -
we have two choices:
Add **x** or **y** to the cover.

And we really don't know
which one is better…

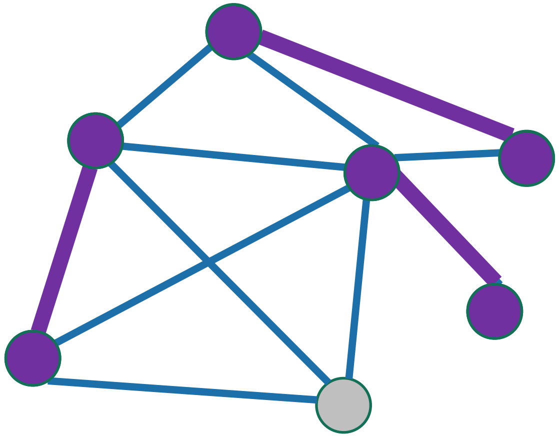
Attempt 3: Take both **x** and **y**!



Attempt 3: Take both **x** and **y**!

Approx_VertexCover $(G = (V, E))$:

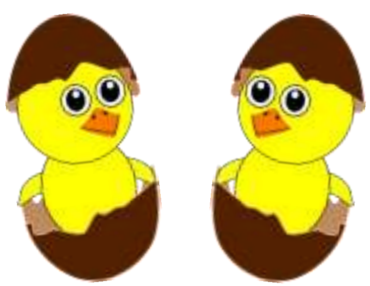
- $S \leftarrow \emptyset$
- While E isn't empty
  - $(\mathrm{x}, \mathrm{y}) \leftarrow$ arbitrary edge $\in E$
  - Add x and y to S
  - Remove their edges from E
- Return S



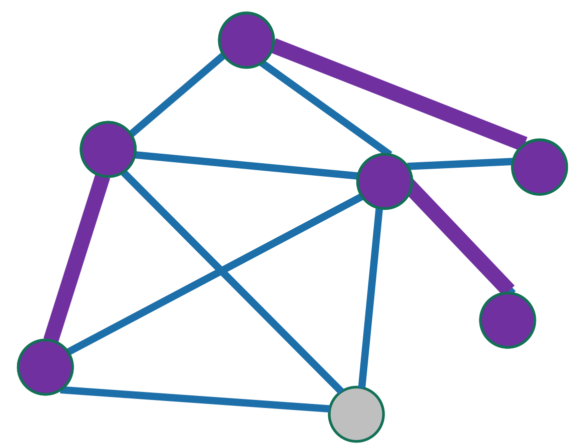
Attempt 3: Take both **x** and **y**!

Does it work?

- Claim: Approx_VertexCover is a 2-approximation algorithm
- Proof:

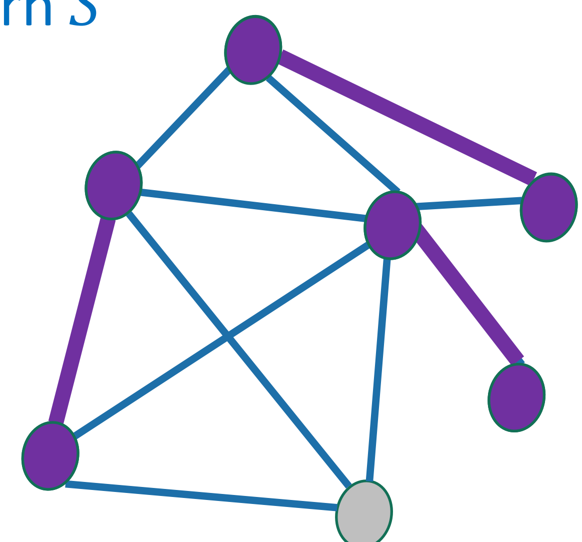


Think-Pair-Share!



Attempt 3: Take both **x** and **y**!

Approx_VC $(G = (V, E))$:
- $S \leftarrow \emptyset$
- While E isn't empty
  - $(\mathrm{x}, \mathrm{y}) \leftarrow$ arbitrary edge
  - Add x and y to S
  - Remove their edges
- Return S



Claim: Approx_VC is a 2-approximation algorithm

Proof

The **edges** selected by the Approx_VC:
1. Don't share any endpoints
2. We need ≥ 1 vertex for each edge

Therefore, $OPT \geq (\#\ \textbf{edges})$

The cover S that we return has 2 vertices for each **edge**

Therefore, $ALG = 2(\#\ \textbf{edges})$

$ALG \leq 2OPT$

Attempt 3: Take both **x** and **y**!

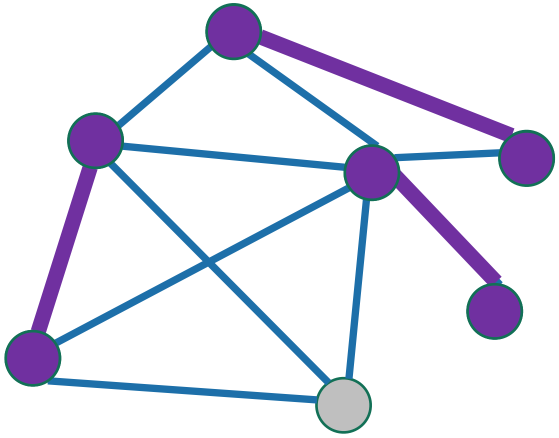
Does it work?

- Claim: Approx_VertexCover is a 2-approximation algorithm
- Proof idea:
  ∀ edge (x,y), we have to add at least one of x,y to S;
  adding both x,y only costs a factor of 2.

Better approximation?

Open problem!
(also related to “Unique Games Conjecture”)

(You’re not responsible
for this rectangle)



Attempt 3: Take both **x** and **y**!

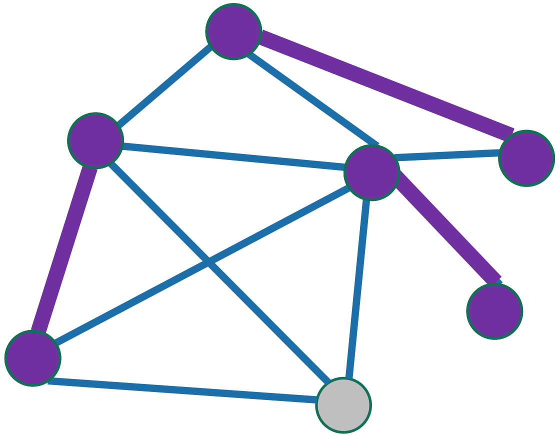
Does it work?

- Claim: Approx_VertexCover is a 2-approximation algorithm
- Proof idea:

  ∀ edge (x,y), we have to add at least one of x,y to S; adding both x,y only costs a factor of 2.

Is it fast?

$$O(n+m)$$



Recap

Coping with NP-hard problems

- Useful special cases
- **Approximation algorithms**
- Exponential time algorithms

Max-Cut

- Greedy algorithm is a 1/2-approximation algorithm
- Random cut is also 1/2-approximation algorithm (in expectation)

Min Vertex Cover

- Take-both-endpoints-of-an-edge is a 2-approximation

Next Time

- CS161 recap + the world beyond

Before next time

- Review session in section!

Next Tuesday (12/11)

- **Finally, your final!**
  Hewlett 200 3:30-6:30

- New: partial credit for wrong algorithms.
  - **You have to clearly state that the algorithm is incorrect**
  - + short explanation of why the algorithm is incorrect
    + short explanation of why you're stuck
  - **Demonstrate some knowledge of course material**

- Read instructions carefully!
  - We really hate taking off points because you didn't understand the question (and we know you hate it too…)

Final plan

(This is just my guess!)

1. A few short questions (25 pts ~ 30 min)

2. **Choose** 3 out of 4:
   Design+analyze an algorithm (25 pts ~ 50 min each)
   (Note: this is NOT best 3 out of 4)

Study resources:

Practice Problems

Review session

Piazza

Textbooks

Classmates

Office Hours

HW, Sections, Lecture notes

Course Feedback

- You’ve done a great job giving feedback throughout the quarter, let’s do it one more time! (This one is “official”.)

- I will only be see to aggregate, anonymous responses.

- Your course feedback is very important.

- Axess > Student > Course and Section Evaluations