PCP Course; Lectures 1, 2

Mario Szegedy

September 10, 2011

Easy and Hard Problems

Easy:	Polynomial Time	$5n$; n^2; $4n^3 + n$
Hard:	Super-polynomial Time	$n^{\log n}$; 2^n; $n!$

Proving that a problem is easy: provide an algorithm
Proving that a problem is hard: ???

Proving hardness is the biggest open problem in complexity theory.

Conditional Hardness

We want to show that if Problem A is hard then problem B is also hard. So we prove the **counter-positive:** If B is easy then A is also easy. (And to prove easiness is easy! We just have to find an algorithm.)

Polynomial Time Reduction:

$$\text{INPUTS}(A) \quad \xrightarrow{\phi} \quad \text{INPUTS}(B)$$

ϕ is a polynomial time map such that $A(x) = B(y)$ for every $x \in \text{INPUTS}(A)$ and $y = \phi(x) \in \text{INPUTS}(B)$.

Terminology: B is hard conditioned on A

Combinatorial Search Problems

- Property P depends on some input x together with witness y;
- P can be computed in polynomial time.

Is there $y \in \{0.1\}^m$ such that property P holds for y, and for input x? (In mathematical notation: $\exists y\ P^y(x)$?)

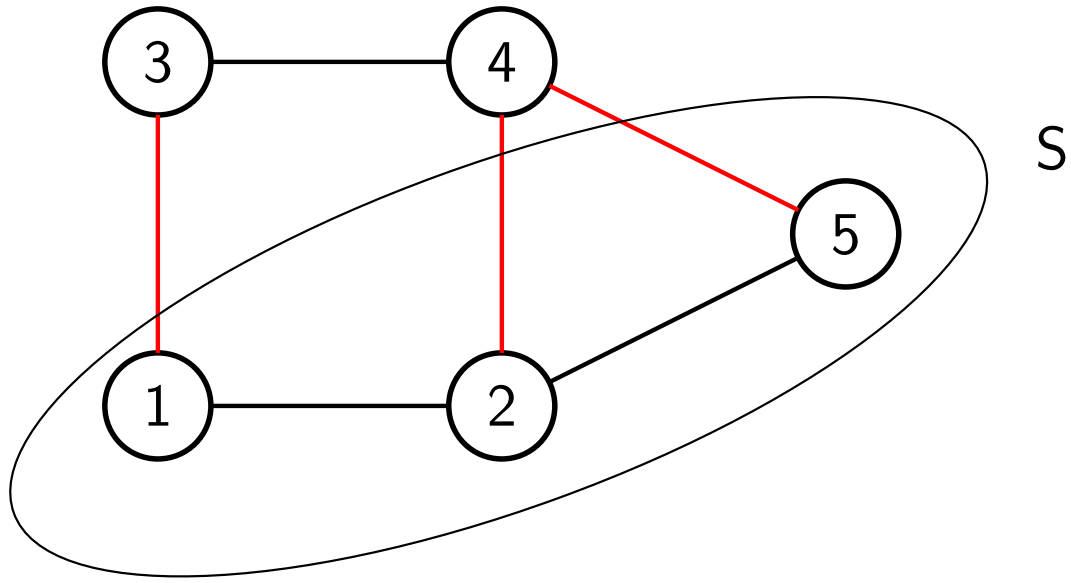
x is a *positive input* (or 1-*input*) if and only if

$$\exists y\ P^y(x)$$

Note: We put the witness into the upper index to differentiate it from x, but some simply write $P(x, y)$.

Example: Exact-CUT-Language

Find a subset $S \subseteq V(G)$ of a graph G such that the cut $(S, \overline{S})$ has size exactly k.



INPUT $= G, k$ OUTPUT $\in \{$ yes, no $\}$

Can you identify our concepts in case of the Exact-CUT-Language?

- **Input** x is the (graph, number) pair (G, k)
- **Witness (or advice)** is the cut S
- **Property** P is the property that S cuts exactly k edges of G.

$$P^S(G,k) = \begin{cases} 1 & (\text{or "yes"}) \quad \text{if cut } S \text{ cuts exactly } k \text{ edges of } G \\ 0 & (\text{or "no"}) \quad \text{otherwise} \end{cases}$$

(G, k) is a positive input instance if there exists an S such that $P^S(G, k) = 1$:

$$\exists (G,k)\ P^S(G,k).$$

How do we think about it?

- We think of property P as fixed.
- For most of the time we think of input x as fixed, given to us.
- We think of advice y as something that we have to find, or if it is given to us as an advise, we need to verify (check) it. So (somewhat unfairly to x:) y is really the main character of the story here.

Hardness and Combinatorial Search Problems

To cycle through all $y \in \{0,1\}^m$s would take $2^n \mathrm{poly}(n)$ time. Is there anything quicker?

ETH (Exponential Time Hypothesis): 3SAT with n variables requires $(1+\epsilon)^n$ time to solve for some fixed $\epsilon > 0$. This is a stronger assumption than $P \neq NP$.

Why did I mention ETH?

Because sometimes we shall use assumptions like

$$\begin{aligned} NP &\not\subseteq DTIME(n^{\log n}) \\ NP &\not\subseteq DTIME(n^{\log\log n}) \\ \ldots &\quad \ldots \end{aligned}$$

These all follow from ETH (in fact, much weaker assumptions than ETH, but stronger than $P \neq NP$).

Another remark: Terminology

1. Combinatorial Search Problems
2. Non-determinististic (Poly Time) Problems
3. NP Problems

MEAN ALL THE SAME

y is called the element of the search domain, advise, or non-determinism, in short.

Cook-Karp-Levin-reduction between NP problems

Reduction from P_1 to P_2:

Instance Transformation :

$$\{ \text{ inputs for } P_1 \} \quad \xrightarrow{\phi} \quad \{ \text{ inputs for } P_2 \}$$

Witness Transformation :

$$\{ \text{ witnesses for } \phi(x) \} \quad \xrightarrow{\psi_x} \quad \{ \text{ witnesses for } x \}$$

Here x denotes a (generic) input of P_1.

Note: In the literature the instance transformation part is called Karp reduction. I have learned the witness reduction from Leonid Levin, so sometimes I will call the whole system Levin-Reduction.

Focus on the witness transformation

The input transformation is dull:) (input transformation ensures that positive P_1-instances go into positive P_2-instances)

What requires smartness is the witness transformation. Witness transformation ensures that positive P_2-instances have positive pre-images. Therefore, by contradiction, negative P_1-instances go into negative instances.

Well, in reality you design the two reductions together.

An example: The transformation 3SAT $\longrightarrow$ Max Clique

$(x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_4)$

$\downarrow$

$(x_1 = 0; x_2 = 0; x_3 = 1)$ $\qquad$ $(x_1 = 0; x_2 = 0; x_4 = 0)$

$\ldots$ $\qquad$ $\ldots$

$(x_1 = 1; x_2 = 1; x_3 = 1)$ $\qquad$ $(x_1 = 1; x_2 = 1; x_4 = 1)$

Alltogether $7m$ nodes, where m is the number of clauses. Column i contains the 7 assignments that satisfy the i^{th} clause. We call these *local satisfying assignments*. Two local assignments (in different columns) are connected by an edge if they are non-contradictory. Example: $(x_1 = 0; x_2 = 0; x_3 = 1)$ and $(x_1 = 1; x_2 = 1; x_4 = 1)$ are contradictory, because x_1 is evaluated to 0 and 1, respectively.

A note you should read

MAX-CLIQUE is not a language, because it returns a number: the size of the maximal clique in the input graph (rather than "yes" or "no"). In order to make it a language, we apply the same trick as in the case of the Exact-CUT-Language: We shall add a target number to the graph – the size of the clique we are searching for.

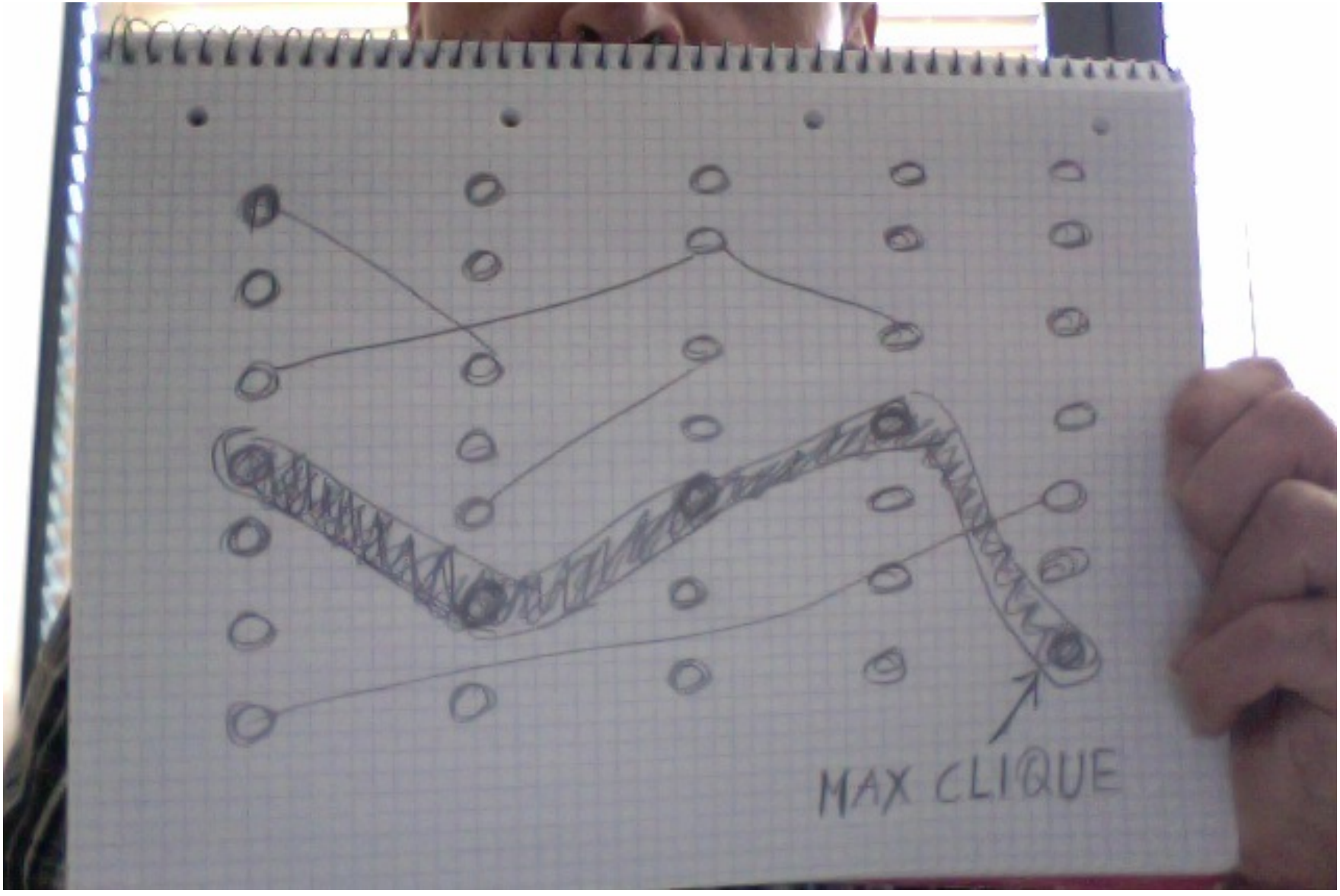
In our reduction we fix this target number. We shall set this number to m. (Recall: m was the number of clauses of the 3SAT instance = the number of columns in our clique instance) We show that even this problem is as hard as 3SAT.

So what is the witness transformation here?

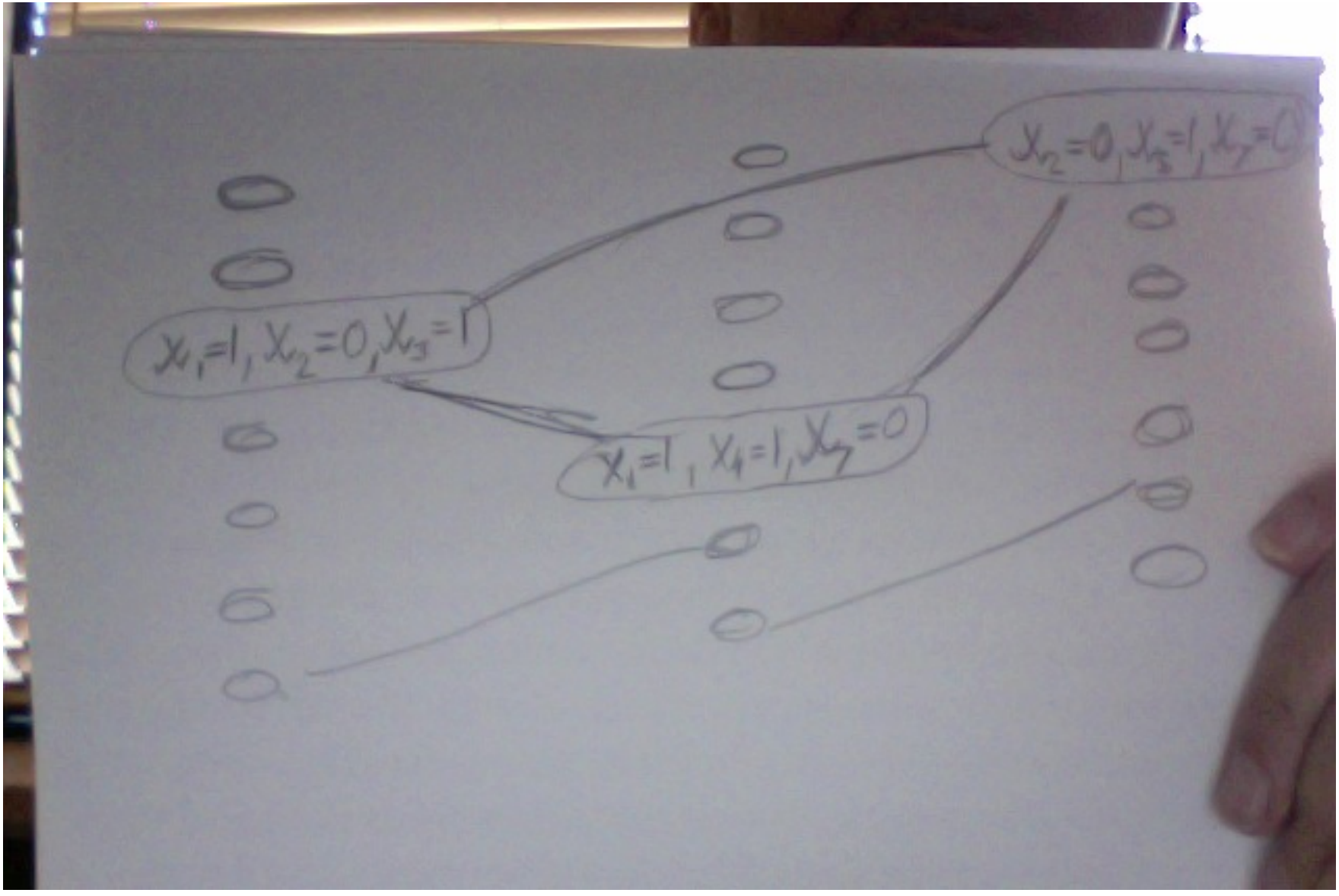
1. **A witness of the clique instance:** Any clique of size m (a clique that picks a node from every column).
2. We have to transform such a witness to a satisfying assignment of the 3SAT instance.
3. **Witness transformation:** Create an assignment from the clique by combining the local assignments at the clique nodes together: (e.g. $(x_1 = 0; x_2 = 0; x_3 = 1)$ and $(x_1 = 0; x_2 = 0; x_4 = 0)$ becomes $x_1 = 0; x_2 = 0; x_3 = 1; x_4 = 0$. We will never find a contradiction, since we started from a clique, so every two local assignments are joined by an edge, meaning they are non contradictory.

Note: the construction satisfies *all* clauses, since the clique had size m.

Turning Max Clique into a 3-SAT assignment



Try to combine the local assignments in the clique of size three below



Hello there!

The topics of this course

1. **Proving hardness of approximating combinatorial optimization problems.**
2. **Probabilistic verification (PCPs).**

What is the relation between the two?

You will learn.

But first you have to understand each separately.

NP Optimization Problems (NPO)

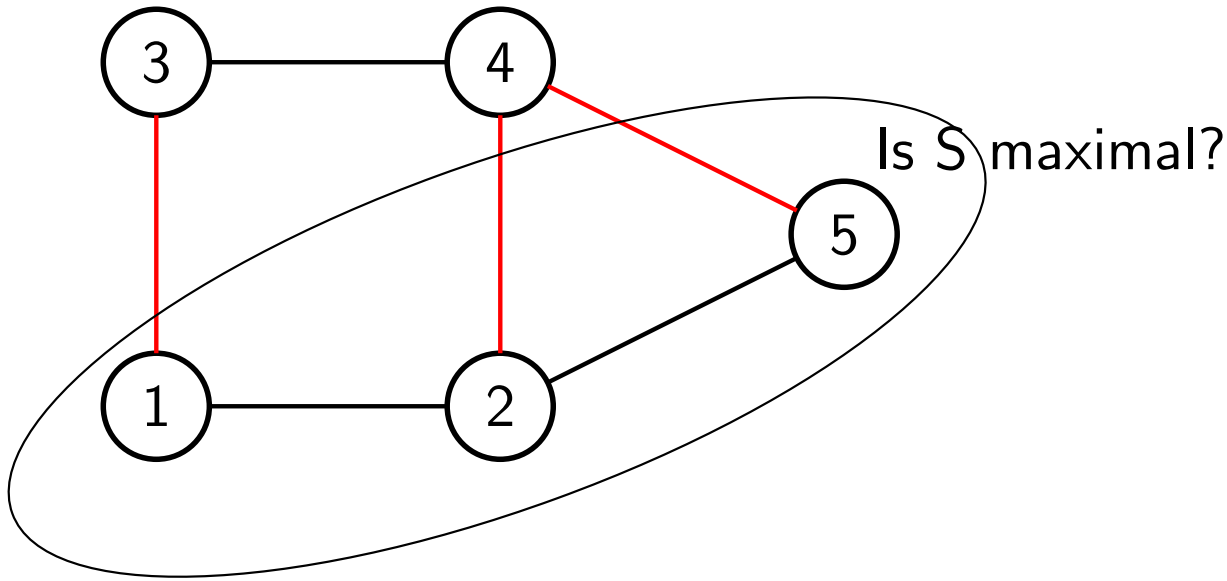
(The same as combinatorial optimization problems.)
Comes in two flavors:
Maximization Problems; Minimization Problems.

Let P now be instead of a property, some integer-valued function, computable in polynomial time.

Find y such that $P^y(x)$ is maximal (for fixed input x). Call this maximal value $\mathrm{OPT}(x)$, and call y an optimal witness.

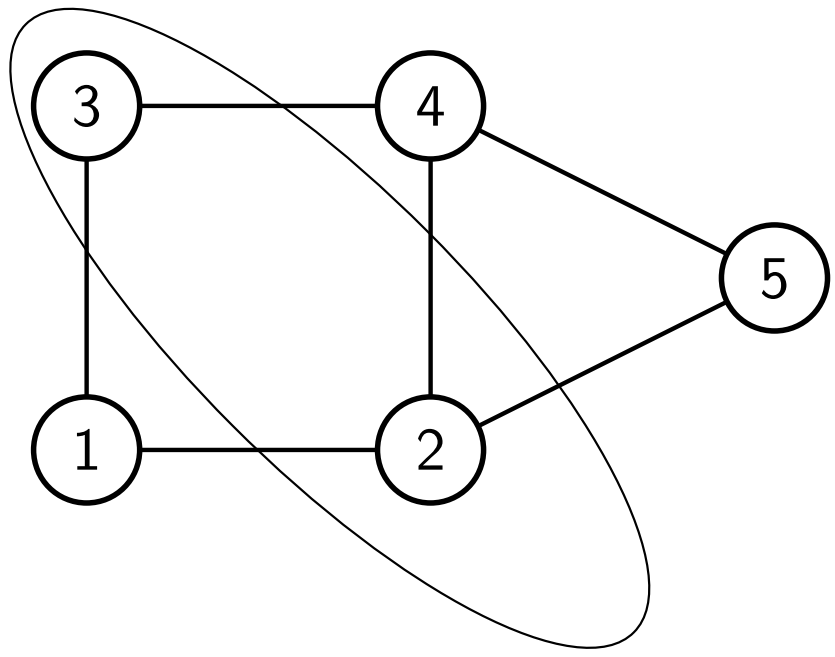
Example: MAX-CUT

Find a subset $S \subseteq V(G)$ of a graph G such that the size of the cut $(S, \overline{S})$ is maximized.



INPUT $= G$ OUTPUT $=$ largest cut

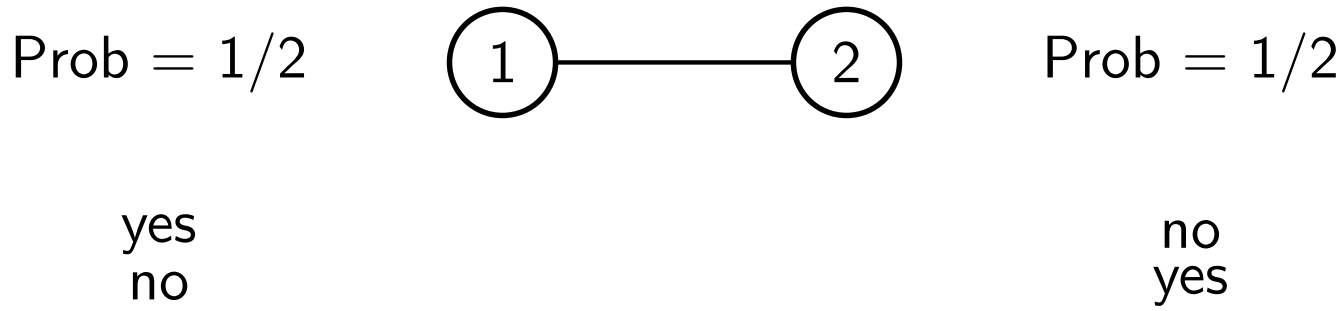
No, you can actually cut 5 edges



$\frac{1}{2}$-approximation algorithm for MAX CUT

- Select every node into S, independently with probability 0.5.
- On expectation we get a cut of size $|E(G)|/2$.

PROOF:



Prob(yes,no) = Prob(no, yes) = 1/4, so the probability that a fixed edge is cut is 1/4 + 1/4 = 1/2.

Approximation Ratio

Let P be a maximization problem with optimal value $OPT(x)$ for input x.

Definition: If for every input x an algorithm finds a witness y such that

$$P^y(x) \geq \alpha \times OPT(x),$$

then α is called the **approximation ratio** of the algorithm.

Best Approximation Ratio

Assume we cannot solve a polynomial time. Then WE LOOK FOR THE POLYNOMIAL TIME ALGORITHM THAT ACHIEVES THE BEST APPROXIMATION RATIO POSSIBLE.

Puzzle: Assume that for a problem every $\epsilon > 0$ there is an algorithm that achieves approximation ratio $1 - \epsilon$. Does it follow that then there exists a single algorithm with approximation ratio 1?

Note: such sequence of algorithms is called *Polynomial Time Approximation Scheme (PTAS)*.

Best Approximation Ratio for MAX-CUT

Is it true that 1/2 the best approximation ratio for MAX-CUT?
No.

Goemans-Williamson: A semidefinite programming based algorithm achieves an approximation ratio of 0.87856....

Subhash Khot, Guy Kindler, Elchanan Mossel, Ryan ODonnell: The Unique Games Conjecture implies that no better approximation ratio than 0.87856... can be achieved.

Apropos, what is the Unique Games Conjecure?

It is a fantastic new hypothesis you will learn about. Here is the history of approximation in a nutshell:

- Until PCP theory, they had tried to prove in-approximability via approximation preserving reductions starting from some basic in-approximability assumptions, with little success.
- In PCP theory we use $P \neq NP$ as the basic hardness assumption, and Levin reduction (in a certain novel way). This was a breakthrough.
- In post-PCP theory we use Unique Games Conjecture as the as the basic hardness assumption, and Levin reduction. — we get a large number of exact bounds, like that the GW algorithm achieves the best approximation ratio for MAX-CUT.