CSE 421

Polynomial Time Reductions
NP Completeness

Shayan Oveis Gharan

Decision Problems

A decision problem is a computational problem where the answer is just yes/no

Here, we study computational complexity of decision Problems.

Why?

- much simpler to deal with
- Decision version is not harder than Search version, so it is easier to lower bound Decision version
- Less important, usually, you can use decider multiple times to find an answer .

Polynomial Time

Define P (polynomial-time) to be the set of all decision problems solvable by algorithms whose worst-case running time is bounded by some polynomial in the input size.

Do we well understand P?

- We can prove that a problem is in P by exhibiting a polynomial time algorithm
- It is in most cases very hard to prove a problem is not in P.

Beyond P?

We have seen many problems that seem hard

- Independent Set
- 3-coloring
- Min Vertex Cover
- 3-SAT

The independent set S

The 3-coloring

The vertex cover S

The T/F assignment

Given a 3-CNF $(x_1 \vee \overline{x_2} \vee x_9) \wedge (\overline{x_2} \vee x_3 \vee x_7) \wedge \cdots$ is there a satisfying assignment?

Common Property: If the answer is yes, there is a "short" proof (a.k.a., certificate), that allows you to verify (in polynomial-time) that the answer is yes.

- The proof may be hard to find

NP

Certifier: algorithm C(x, t) is a certifier for problem A if for every string x, the answer is “yes” iff there exists a string t such that C(x, t) = `yes`.

Intuition: Certifier doesn't determine whether answer is “yes” on its own; rather, it checks a proposed proof that answer is “yes”.

NP: Decision problems for which there exists a poly-time certifier.

Remark. NP stands for nondeterministic polynomial-time.

Example: 3SAT is in NP

Given a 3-CNF formula, is there a satisfying assignment?

Certificate: An assignment of truth values to the n boolean variables.

Verifier: Check that each clause has at least one true literal.

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

Certificate: $x_1 = T, x_2 = F, x_3 = T, x_4 = F$

Conclusion: 3-SAT is in NP

Example: Hamil-Cycle is in NP

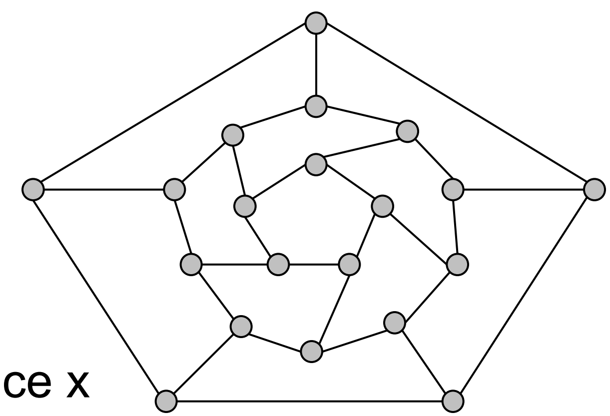
HAM-CYCLE. Given an undirected graph G = (V, E), does there exist a simple cycle C that visits every node?

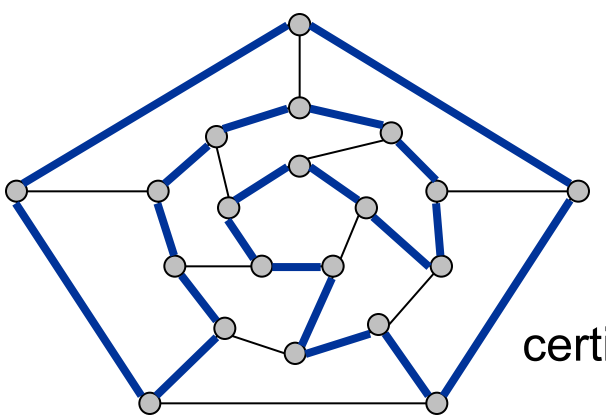
Certificate. A permutation of the n nodes.

Certifier. Check that the permutation contains each node in V exactly once, and that there is an edge between each pair of adjacent nodes in the permutation.

Conclusion. HAM-CYCLE is in NP.

Instance x





certificate t

Example: Min s,t-cut in in NP

MIN-CUT. Given a flow network, and a number k, does there exist a min-cut of capacity at most k?

Certificate. A s-t min-cut (A,B).

Certifier. Check that the capacity of the min-cut is at most k.

Conclusion. MIN-CUT is in NP.

P, NP, EXP

P. Decision problems for which there is a poly-time algorithm.

EXP. Decision problems for which there is an exponential-time algorithm.

NP. Decision problems for which there is a poly-time certifier.

Claim. P $\subseteq$ NP.

Pf. Consider any problem X in P.

By definition, there exists a poly-time algorithm A(x) that solves X.

Certificate: t = empty string, certifier C(x, ∅) = A(x). ▪

Claim. NP $\subseteq$ EXP.

Pf. Consider any problem X in NP.

By definition, there exists a poly-time certifier C(x, t) for X.

To solve input x, run C(x, t) on all strings t with of length polyn in |x|

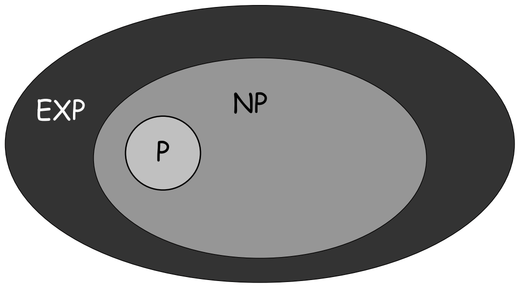
Return `yes`, if C(x, t) returns `yes` for any of these.

The main question: P vs NP

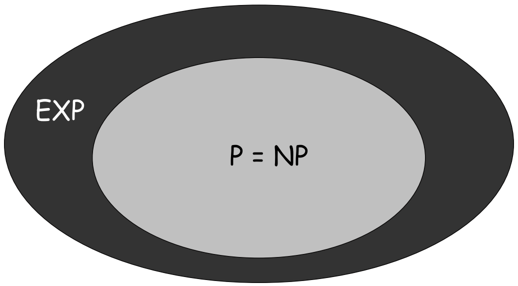
Does P = NP? [Cook 1971, Edmonds, Levin, Yablonski, Gödel]

Is the decision problem as easy as the certification problem?

Clay $1 million prize.



If P ≠ NP



If P = NP

If yes: Efficient algorithms for 3-COLOR, TSP, FACTOR, SAT, …

If no: No efficient algorithms possible for 3-COLOR, TSP, SAT, …

What do we know about NP?

- Nobody knows if all problems in NP can be done in polynomial time, i.e. does P=NP?
  - one of the most important open questions in all of science.
  - Huge practical implications specially if answer is yes

- To show Hamil-cycle $\notin P$ we have to prove that there is no poly-time algorithm for it even using all mathematical theorem that will be discovered in future!

NP Completeness

Complexity Theorists Approach: We don't know how to prove any problem in NP is hard. So, let's find hardest problems in NP.

NP-hard: A problem B is NP-hard iff for any problem $A \in NP$, we have $A \leq_p B$

NP-Completeness: A problem B is NP-complete iff B is NP-hard and $B \in NP$.

Motivations:

- If $P \neq NP$, then every NP-Complete problems is not in P. So, we shouldn't try to design Polytime algorithms
- To show $P = NP$, it is enough to design a polynomial time algorithm for just one NP-complete problem.

Cook-Levin Theorem

Theorem (Cook 71, Levin 73): 3-SAT is NP-complete, i.e., for all problems $A \in NP$, $A \leq_p$ 3-SAT.

- So, 3-SAT is the hardest problem in NP.

What does this say about other problems of interest? Like Independent set, Vertex Cover, …

Fact: If $A \leq_p B$ and $B \leq_p C$ then, $A \leq_p C$
Pf idea: Just compose the reductions from A to B and B to C

So, if we prove 3-SAT $\leq_p$ Independent set, then Independent Set, Clique, Vertex cover, Set cover are all NP-complete

3-SAT $\leq_p$ Independent Set $\leq_p$ Vertex Cover $\leq_p$ Set Cover

Summary

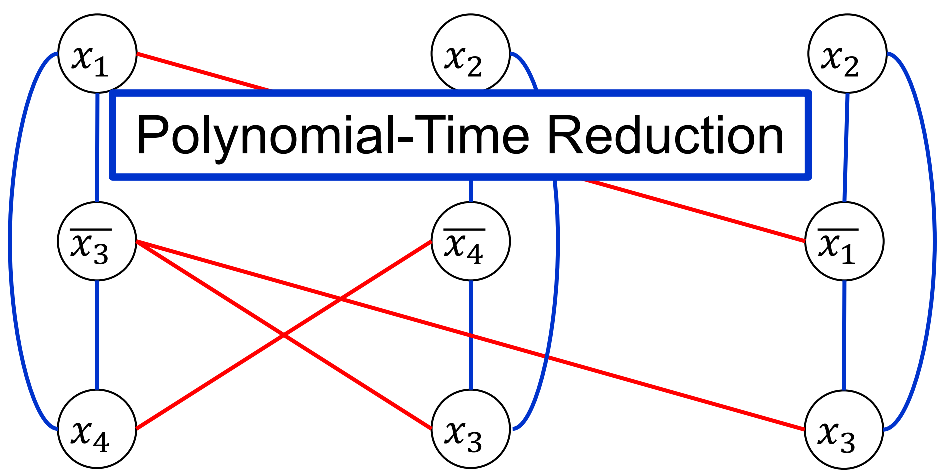
- If a problem is NP-hard it does not mean that all instances are hard, e.g., Vertex-cover has a polynomial-time algorithm on trees or bipartite graphs
- We learned the crucial idea of polynomial-time reduction. This can be even used in algorithm design, e.g., we know how to solve max-flow so we reduce image segmentation to max-flow
- NP-Complete problems are the hardest problem in NP
- NP-hard problems may not necessarily belong to NP.
- Polynomial-time reductions are transitive relations

3-SAT $\leq_p$ Independent Set

Map a 3-CNF to (G,k). Say m is number of clauses

- Create a vertex for each literal
- Join two literals if
  - They belong to the same clause (blue edges)
  - The literals are negations, e.g., $x_i, \overline{x_i}$ (red edges)
- Set k=m

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



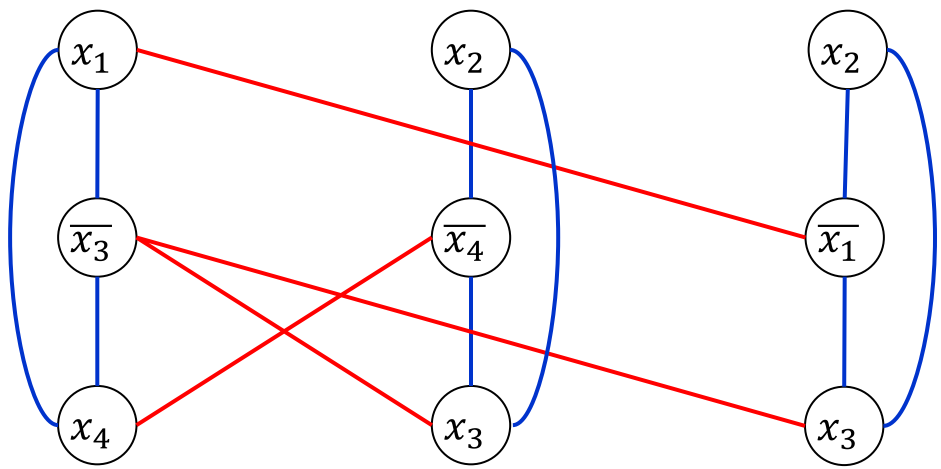
Correctness of 3-SAT $\leq_p$ Indep Set

F satisfiable => An independent of size m

Given a satisfying assignment, Choose one node from each clause where the literal is satisfied

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

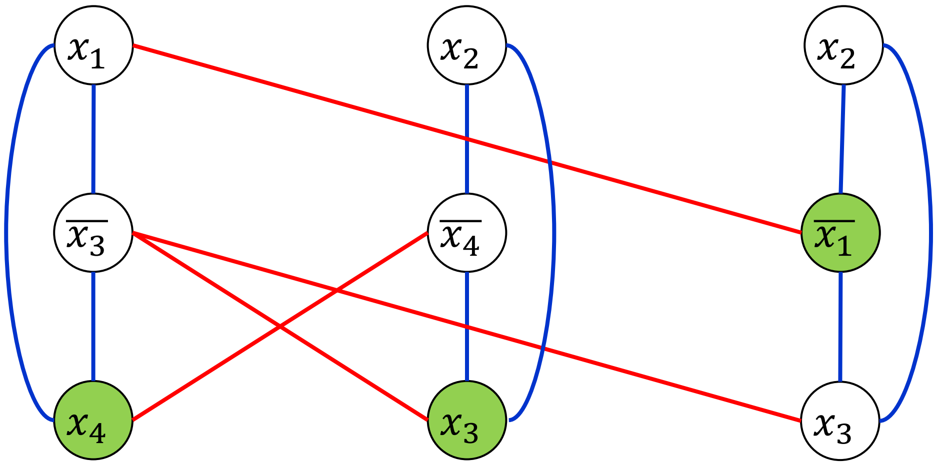
Satisfying assignment: $x_1 = T, x_2 = F, x_3 = T, x_4 = F$



- S has exactly one node per clause => No blue edges between S
- S follows a truth-assignment => No red edges between S
- S has one node per clause => |S|=m

Correctness of 3-SAT $\leq_p$ Indep Set

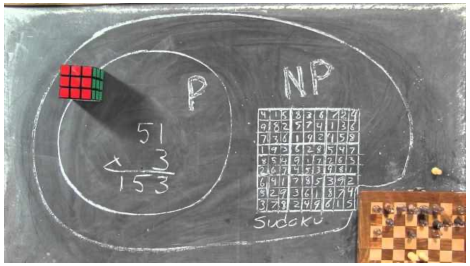
An independent set of size m => A satisfying assignment
Given an independent set S of size m.
S has exactly one vertex per clause (because of blue edges)
S does not have $x_i, \overline{x_i}$ (because of red edges)
So, S gives a satisfying assignment



Satisfying assignment: $x_1 = F, x_2 = ?, x_3 = T, x_4 = T$

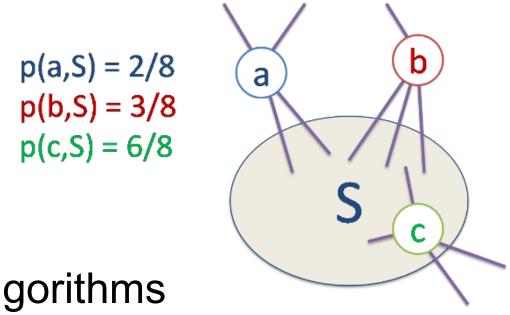
$$(x_1 \vee \overline{x_3} \vee x_4) \wedge (x_2 \vee \overline{x_4} \vee x_3) \wedge (x_2 \vee \overline{x_1} \vee x_3)$$

What is next?



- CSE 431 (Complexity Course)
  - How to prove lower bounds on algorithms?
- CSE 422 (Advanced Toolkit for Modern Alg)
  - SVD, Data structures, many programming tasks
- CSE 521 (Graduate Algorithms Course)

  Prereq: 312, Math 308
  - How to design streaming algorithms?
  - How to design algorithms for high dimensional data?
  - How to use matrices/eigenvalues/eigenvectors to design algorithms
  - How to use LPs to design algorithms?



- CSE 525 (Graduate Randomized Algorithms Course)

  Prereq: CSE 521
  - How to use randomization to design algorithms?
  - How to use Markov Chains to design algorithms?

Course Evaluations

- How can we improve this course?
- Did you like topics related to linear programming? Did you like to see more of that?
- Which topic was most/least interesting to you?
- Which problem sets did you like more?