COMP 446 / 546 ALGORITHM DESIGN AND ANALYSIS

LECTURE 14 P-NP

ALPTEKİN KÜPÇÜ

Based on slides of Shafi Goldwasser, Xukai Zou, Michael Goodrich, and Roberto Tamassia

TRACTABLE PROBLEMS

- **Our focus so far: Problems that can be solved efficiently**
  - Matrix Multiplication
  - Is N a prime or composite
  - Max Flow Min Cut
  - Linear Programming
- **Tractable: Problems that can be solved by algorithms which run in time $O(n^k)$ for some constant $k > 0$ where n is the input size.**

INTRACTABLE PROBLEMS

- **Are all problems tractable? Far from it.**
- **No known polynomial-time algorithm for:**
  - Traveling Salesman Problem: On a weighted graph G, find minimum-cost cycle that visits each vertex exactly once.
  - Factoring: On input N, find its prime factorization
- **No known (any complexity) algorithm for:**
  - Halting Problem: Given a program P (e.g., written in Java) and an input x, does P(x) terminate ? UNDECIDABLE!!

CLASSIFY PROBLEMS

- **Provably un-deciable: Halting**
- **Provably requires exponential-time:**
  - Given a Turing machine, does it halt in at most n steps?
  - Given a board position in an n-by-n generalization of chess, can black guarantee a win?
- **Frustrating news: Huge number of fundamental problems have defied classification for decades.**

DECISION, SEARCH, OPTIMIZATION PROBLEMS

- **Decision problem:**
  - Given a matrix A, vectors b and c, and a value k, is there a real-valued vector x, such that $Ax \leq b$ and $cx \geq k$.
- **Search Problem:**
  - Given a matrix A, and vectors b and c, and a value k, find a real-valued vector x such that $Ax \leq b$ and $cx \geq k$.
- **Optimization problem:**
  - Given a matrix A, and vectors b and c, find a real-valued vector x, such that $Ax \leq b$ that maximizes the value of cx.
- **Optimization is harder than Search, which is harder than Decision**
  - If one can solve Optimization, then one can easily solve Search
  - If one can solve Search, then one can easily solve Decision

DECISION, SEARCH, OPTIMIZATION PROBLEMS

- **Decision problem:**
  - Given a weighted graph $G=(V,E)$, a weight function w, and a bound k, is there a spanning tree of G of total weight less than k.
- **Search Problem:**
  - Given a weighted graph $G=(V,E)$, a weight function w, and a bound k, find a spanning tree of G of total weight less than k.
- **Optimization problem:**
  - Given a weighted graph $G=(V,E)$, and a weight function w, find a spanning tree of G of minimum possible total weight.

DECISION PROBLEMS

- **Problems whose answer is either Yes or No**
- **Complexity theory usually deals with decision version of problems**
  - We are interested in classifying how hard problems are.
  - If one can show decision version is hard, then we immediately know its search and optimization versions are also hard.
- **A decision problem is a function D from the set all binary strings (all possible inputs represented in binary) to the set of {1,0}.**
  - An input x to D is a yes-input if $D(x)=1$
  - An input x to D is a no-input if $D(x)=0$
  - We write $x \in D$ if and only if $D(x)=1$

COMPLEXITY CLASS P

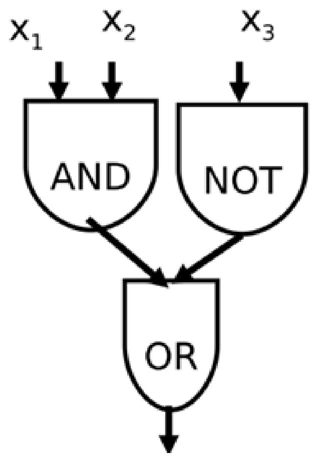
- **Complexity Class P = { All decision problems D such that there exits a polynomial-time algorithm A such that A(x)=D(x) for all inputs x }**
  - A is an algorithm that solves the decision problem D
  - We say that A decides D
- **Examples**
  - ***Existence of a Spanning Tree with cost at most k ?***
  - ***Feasibility of a Linear Program ?***
  - ***Is N a prime number ?***

PROBLEMS NOT KNOWN TO BE IN P: TSP

- **Decision version of Traveling Sales Person (TSP):**
  - Given a complete weighted graph G and an integer k, is there a simple cycle that visits all nodes exactly once whose total weight is less than or equal to k.
- **Search version: Find such a cycle**
- **Optimization version: Find minimum-weight simple cycle**
- **Best known algorithm: Solve decision version by solving the search version**
  - Find cycle in time $O(2^V)$ for a graph with V vertices.
  - Given cycle, it is easy to verify that its weight is $\leq k$
  - Given a cycle, one cannot verify its optimality in polynomial time
    - Search problem does not let us solve optimization problem easily

PROBLEMS NOT KNOWN TO BE IN P: SAT & CIRCUIT-SAT

- **Circuit Satisfiability (CIRCUIT-SAT): Given a Boolean circuit C made of gates and wires (an acyclic diagraph whose vertices can be AND/OR/NOT gates) with n inputs and one output, is there a way to assign 0-1 values to the inputs so that the output value is 1?**
- **General Satisfiability (SAT): Given a Boolean formula over n variables, is there an assignment of TRUE/FALSE to the variables that make the formula TRUE.**
- **For this lecture, we will not distinguish between SAT and cSAT.**
- **Best Known algorithm: Try out all 2^n assignments.**
  - EXPONENTIAL!!



PROBLEMS NOT KNOWN TO BE IN P: FACTORING

- **Decision version of Factoring: Given an n-bit integer N and bound B, is there a divisor d of N such that 1<d<B**
- **Search Version: Given an n-bit integer N, find a divisor d of N such that 1<d<N**
- **Best known algorithm: O(e^K) where K =~ $n^{1/3} \log_n^{2/3}$**
  - Sub-exponential
  - Super-polynomial
- **Interesting note: A quantum algorithm by Shor solves Factoring in polynomial-time on a quantum computer.**

PROBLEMS NOT KNOWN TO BE IN P: COMMON OBSERVATIONS

- **Solve search to solve decision:**
  - Finding the solution to the search version of these problems is hard
  - But once you found it, the solution has polynomial size and can be verified to be correct in polynomial time
- **For a decision problem D**
  - For yes-inputs x of D, there is a short and easy to verify certificate that $x \in D$
  - The certificate is simply the solution to the search variant of the problem
  - Short means polynomial-size
  - Easy to verify means polynomial-time verification
- **ONLY FOR YES-INPUTS !!!!!!!!!!!!!**

COMPLEXITY CLASS NP

- **Complexity Class NP: All decision problems D for which there exist polynomial-time verification algorithm V and constant $c > 0$ such that**
  - If D(x) = 1 (x is a yes-input of D) then ∃ certificate y s.t. $|y| < |x|^c$ and V(x,y) = **ACCEPT**
  - If D(x) = 0 (x is a no-input of D) then ∀ certificates y V(x,y) = **REJECT**
- **NO-inputs cannot have certificates that cause the verification algorithm to accept.**
- **Verification algorithm never accepts NO-inputs regardless of the certificate provided.**

COMPLEXITY CLASS NP

- **Theoretical Non-Deterministic Turing Machine**
  - Can “guess” the answer
  - Or, can be thought as running many copies of the program in parallel
  - Accepts an input x if there exists some sequence of operations that outputs “yes” on input x.
- **NP = The class of problems that can be solved in polynomial-time on a Non-Deterministic Turing Machine.**
  - Polynomial-time verification definition is equivalent!

REMARKS

- For every yes-input for an NP problem, there exists at least one certificate, but possibly many
- P means decision problems that can be solved in polynomial time
- NP does not mean decision problems that are "not solvable in polynomial time"
- NP means "decision problems with polynomial-time verification algorithms" (non-deterministic polynomial time)

REMARKS

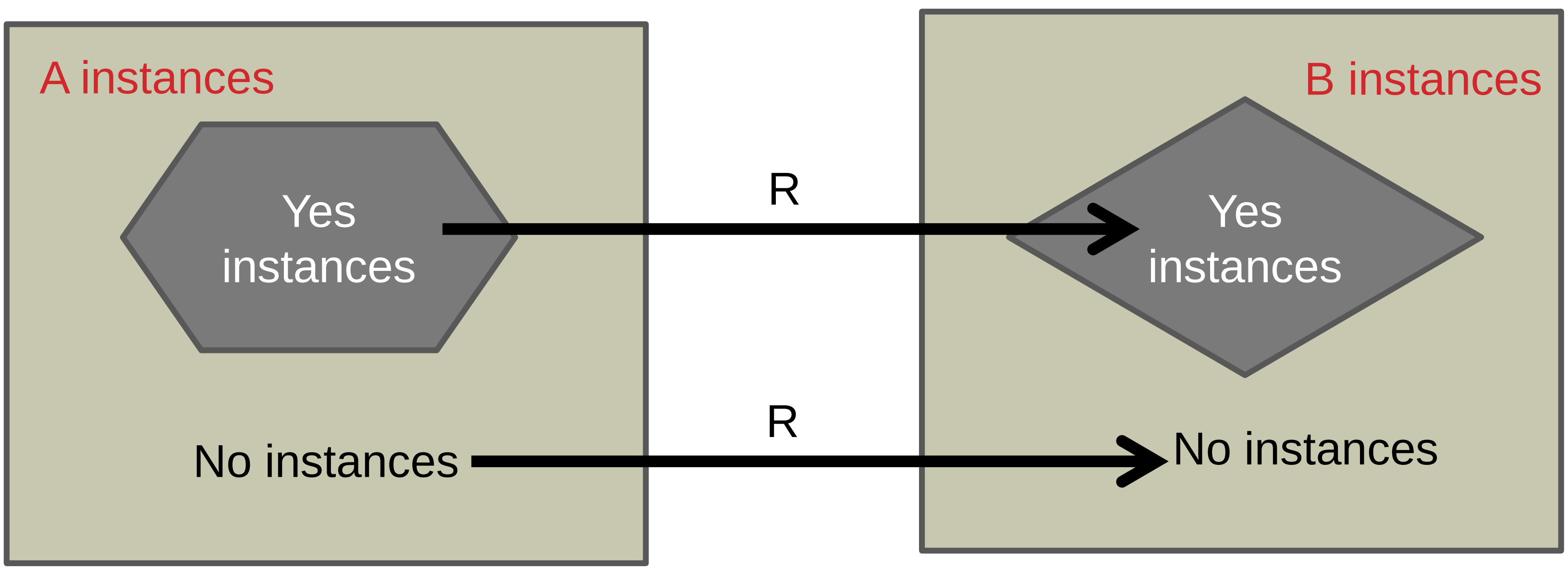
- **For every yes-input for an NP problem, there exists at least one certificate, but possibly many**
- **P means decision problems that can be solved in polynomial time**
- **NP does not mean decision problems that are "not solvable in polynomial time"**
- **NP means "decision problems with polynomial-time verification algorithms" (non-deterministic polynomial time)**
- **P $\subseteq$ NP (every problem in P is also in NP)**
  - Since every deterministic TM is also non-deterministic
- **Not all decision problems are in NP.**
  - Ex: Given circuit C decide if all assignments to the input variables make the output 0 (FALSE).
  - No short certificate is known that can be verified in polynomial-time

P vs. NP

- **P =? NP**
  - Is it as easy to solve as to verify?
- **Main problem: P and NP classify problems, not solutions**
  - But, classification depends on solution algorithms
- **No proof exists that shows a polynomial-time solution to an NP problem cannot exist.**
  - But highly unlikely
  - The question has been open since 1970s
  - No such algorithm has been found
- **How can we make progress on this problem?**
  - Find the "hardest" problems in NP and just work on them
  - The **NP-Complete** problems

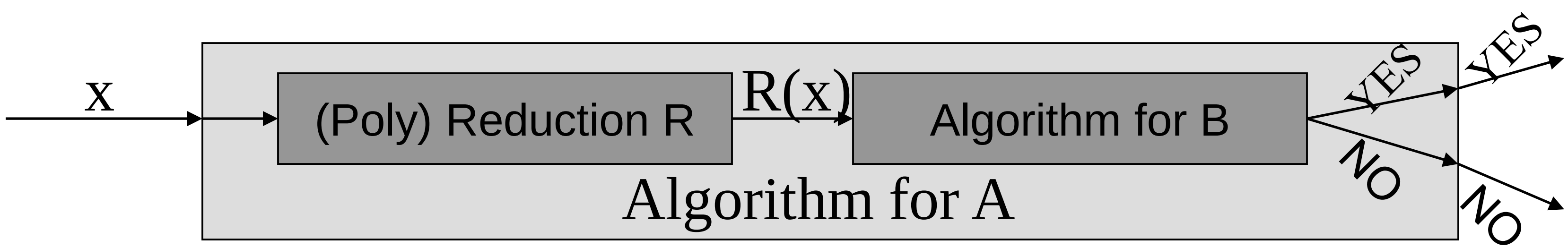
POLYNOMIAL-TIME REDUCTIONS

- **Let A and B be decision problems**
- **A is polynomial-time reducible to B if there exists a polynomial-time algorithm R s.t.**
  - R transforms input x to problem A into input R(x) to problem B so that x is a **yes-input** for A **if and only if** R(x) is **yes-input** for B



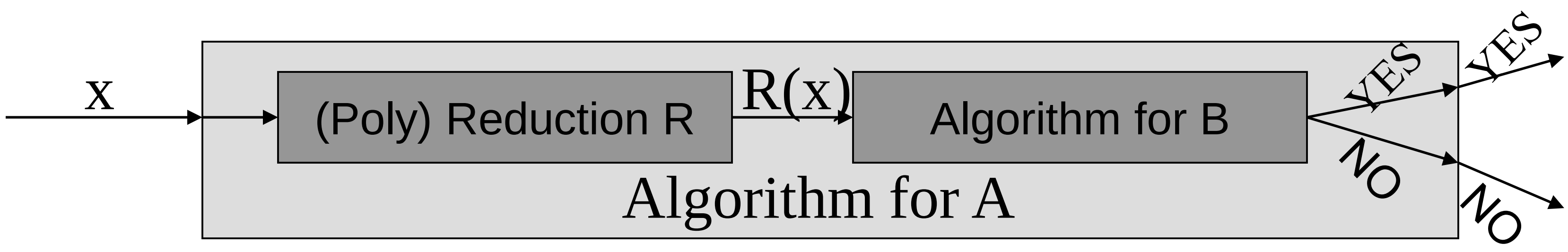
POLYNOMIAL-TIME REDUCTIONS

- **Claim: If A is polynomial-time reducible to B and if ∃ polynomial-time algorithm for B then ∃ polynomial-time algorithm for A**



- **Proof:**
  - Assume R takes p(|x|) time and the algorithm for B takes q(|y|) time, where p and q are polynomial functions.
  - Further assume the output of R is r(|x|) bits where r is a polynomial.
  - Then on input x the running time of A is p(|x|) + q(r(|x|)) = poly(|x|)

POLYNOMIAL-TIME REDUCTIONS



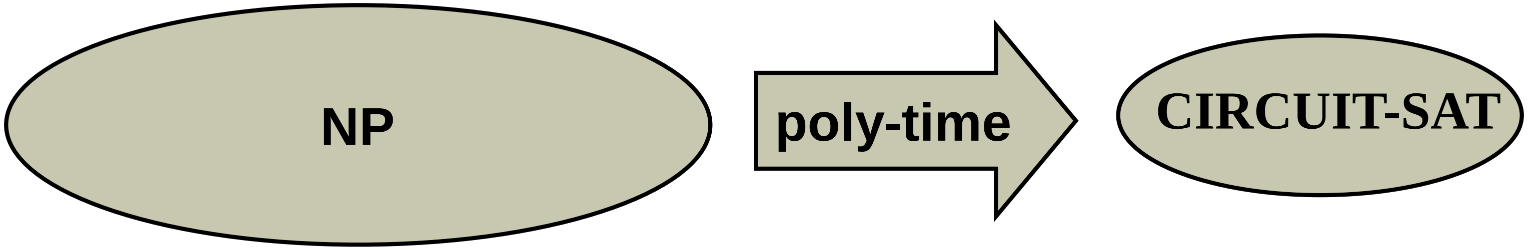
- **Main uses of reductions:**
  - Solving problem A using algorithm for B
  - Showing B is harder than (at least as hard as) A
    - Because if we can solve B, we can easily solve A
    - But even if we can solve A, it may be hard to solve B
  - Cryptographic security proofs

POLYNOMIAL-TIME REDUCTIONS

- **Let $B \geq A$ (or $A \leq B$) denote that A is polynomial-time reducible to B**
  - We use $\geq$ because such a reduction means B is at least as hard as A.
- **Reductions are transitive: If $A \leq B$ and $B \leq C$, then $A \leq C$.**
  - An input x for A can be converted to x' for B, such that $x \in A$ iff $x' \in B$.
  - Likewise, we can convert x' into x'' for C such that $x' \in B$ iff $x'' \in C$.
  - Hence, if $x \in A$, then $x' \in B$ and $x'' \in C$.
  - Likewise, if $x'' \in C$, then $x' \in B$ and $x \in A$.
  - Thus, $A \leq C$ since polynomials are closed under composition and addition. (required for sizes of certificates and verification times)

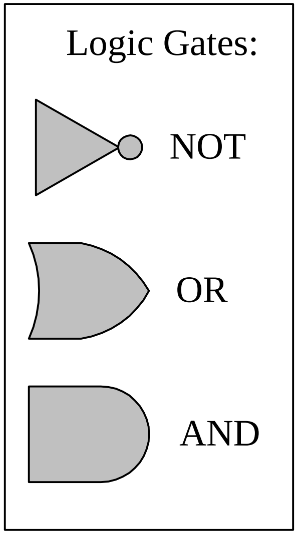
NP-COMPLETENESS

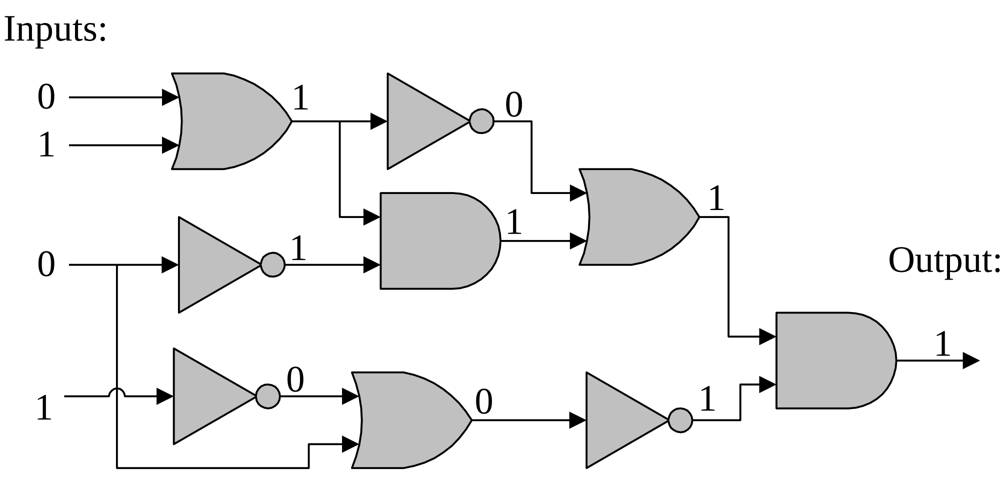
- **A problem L is NP-hard if every problem in NP can be reduced to L in polynomial time.**
- **L is NP-complete if it is in NP and is NP-hard.**
- **Cook-Levin Theorem: SATISFIABILITY is NP-complete.**
  - This includes SAT, CIRCUIT-SAT, 3-SAT



SATISFIABILITY IS IN NP

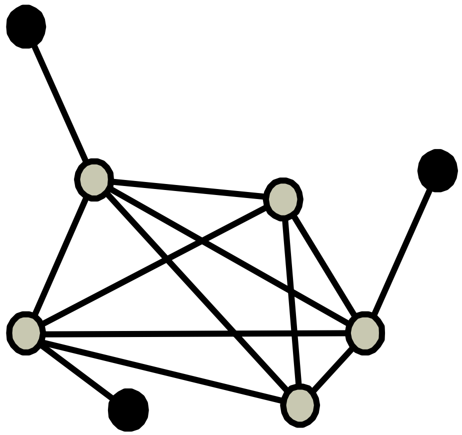
- Consider a Boolean circuit composed of AND/OR/NOT gates.
- The CIRCUIT-SAT problem is to determine if there is an assignment of 0/1 to inputs so that the circuit outputs 1.
- Polynomial-time verification algorithm: Given a set of satisfying inputs (certificate), run each gate and check if the result is 1.





CLIQUE

- **Input: An undirected graph G=(V,E) be and K > 0**
- **Problem: Is there a subset C of V with |C| ≥ K such that every pair of vertices in C has an edge between them?**
  - Does the graph have a clique of size K?
  - Clique of size 3 are easy to detect
- **Theorem: CLIQUE is NP-Complete.**
- **Proof:**
  - First, obviously CLIQUE is in NP
    - ***Verification algorithm ?***
  - For NP-hardness proof
    - Reduce 3-SAT to CLIQUE
    - Reduce inputs so that solving CLIQUE solves 3-SAT

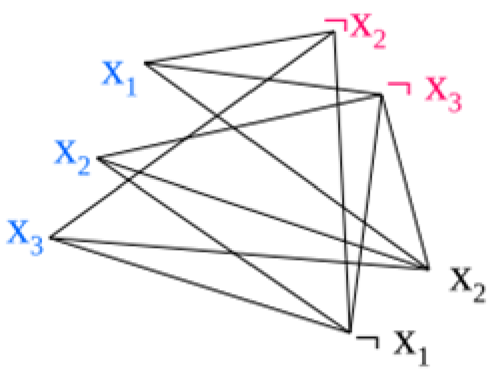


3SAT ≤ CLIQUE REDUCTION

- **Given a 3-SAT input 3-CNF $\varphi=C_1,\ldots,C_m$ over $x_1,\ldots,x_n$, construct a CLIQUE input $G=(V,E)$ and K s.t φ satisfiable iff G has a clique of size $\geq K$**
  - Notation: a literal is either x_i or $\neg x_i$
- **Reduction:**

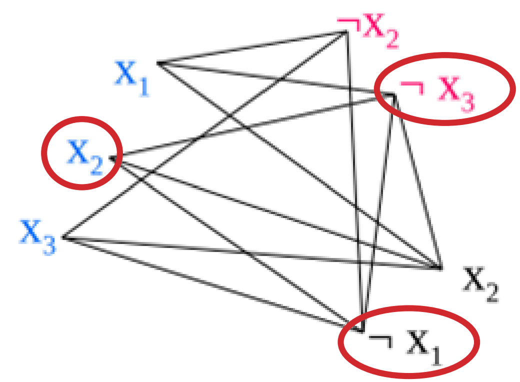
O(n+m)

  - create a vertex for each literal t occuring in a clause
  - create an edge between V_t and $V_{t'}$ unless
    - t and t' are in the same clause, or
    - t is the negation of t'
  - set K=m (#clauses)
- **Example:**
  - Formula: $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2)$



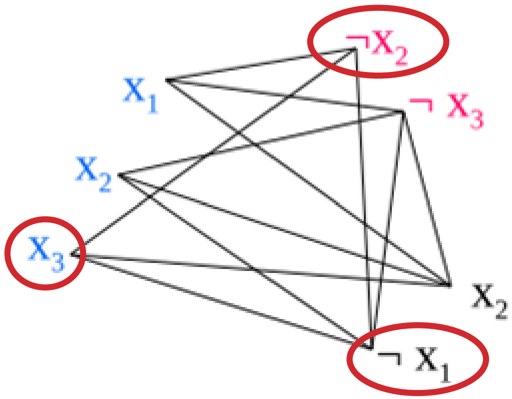
3SAT ≤ CLIQUE REDUCTION

- **Claim: φ satisfiable iff G has a clique of size ≥ m**
- **φ satisfiable => G has a clique of size m**
  - Take any assignment that satisfies φ
    - e.g., x_1=F, x_3=F, x_2=T
  - Let the set of vertices C contain one literal for each clause that made the clause true
    - x_2 & $\neg x_3$ & $\neg x_1$
  - C is a clique of size m
    - All vertices in C are from different clauses
    - No two vertices' labels are negation of each other
      - Otherwise, not a valid assignment
        - No i with x_i = T and x_i = F



3SAT ≤ CLIQUE REDUCTION

- **Claim: φ satisfiable iff G has a clique of size ≥ m**
- **φ satisfiable <= G has a clique of size m**
  - Take any clique C of size m
    - All vertices in C are from different clauses
    - No two vertices' labels are negation of each other
  - Set all literals in the clique to evaluate to true in φ
    - e.g., x_3=T, x_2=F, x_1=F
  - This is a legal assignment which satisfies φ

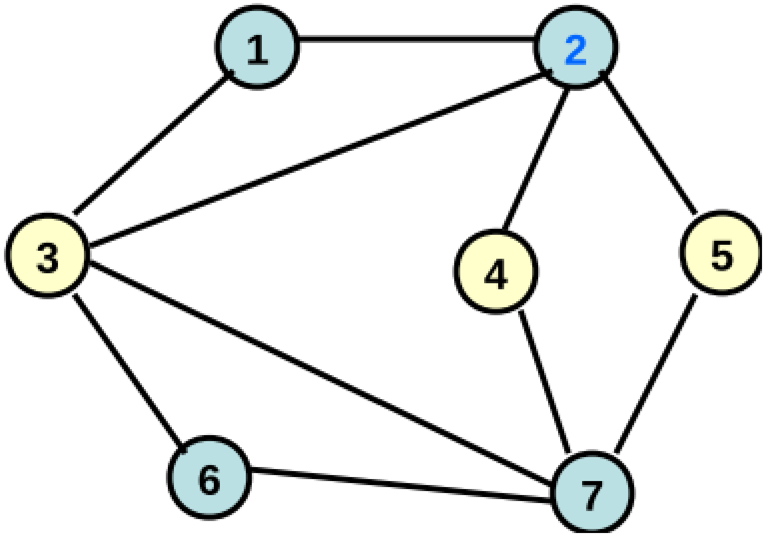


3SAT ≤ CLIQUE REDUCTION SUMMARY

- We constructed a reduction that maps:
  - YES-inputs of 3-SAT to YES-inputs of CLIQUE
  - NO-inputs of 3-SAT to NO-inputs of CLIQUE
- The reduction works in polynomial time
- Therefore, SAT ≤ CLIQUE and hence CLIQUE is NP-hard
- CLIQUE ∈ NP and CLIQUE is NP-hard
  - => CLIQUE is NP-complete

INDEPENDENT SET

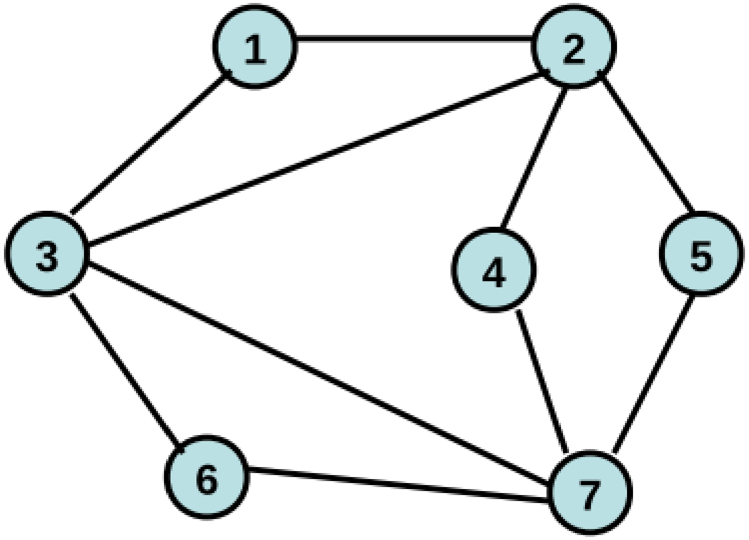
- **Input:** **An undirected graph G=(V,E) and integer K > 0**
- **Problem:** **Is there a subset S of V of size K that is an independent set (that is there are no edges between vertices in S)**

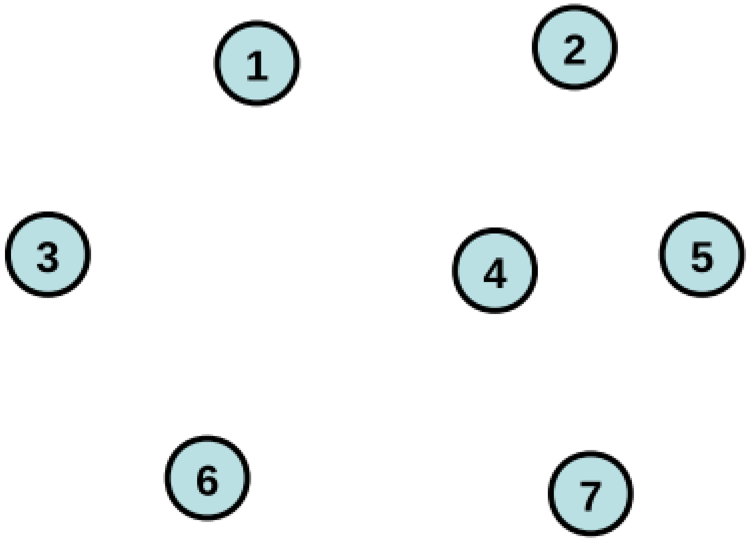


- **Theorem:** **IS is NP-complete**
- **Proof:**
  - First, obviously IS is in NP
    - ***Verification algorithm ?***
  - For NP-hardness proof
    - Reduce CLIQUE to IS
    - Reduce inputs so that solving IS solves CLIQUE

COMPLEMENT OF A GRAPH

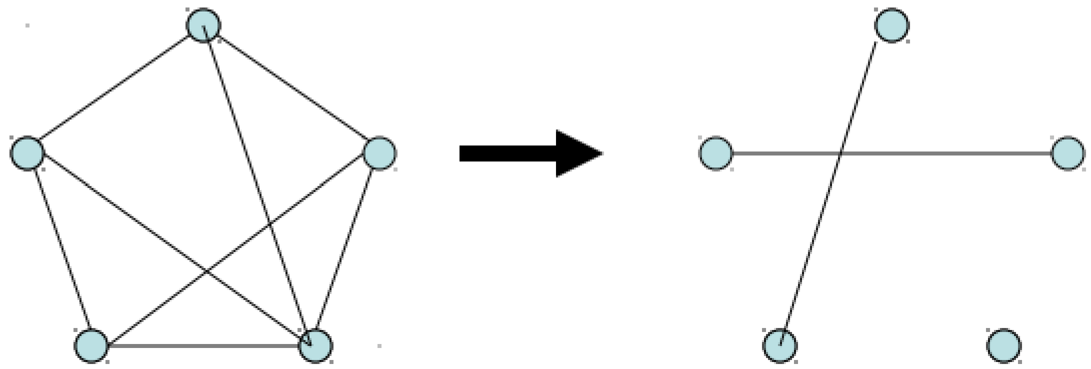
- **G'=(V,E') is the complement of G=(V,E) if edge e ∈ E' iff e ∉ E**
- ***Example: Construct the complement on board***





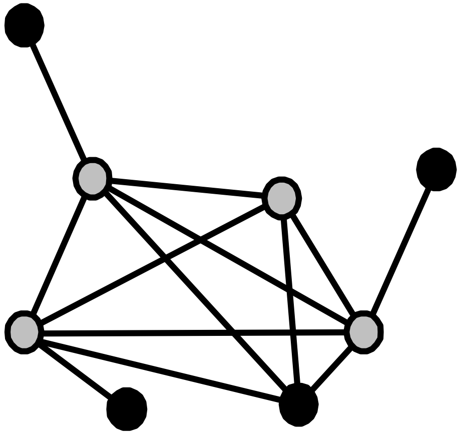
CLIQUE ≤ INDEPENDENT SET

- **Claim: S is a clique in G iff S is an independent set in the complement of G**
- **Reduction:** **$O(|V|^2)$**
  - Compute the complement of the graph G'.
  - The complement graph has an independent set of size K iff the orginal graph had a clique of size K.
  - (G,K) is reduced to (G',K)
- **Duality again!**



VERTEX COVER

- **Input:** **An undirected graph $G=(V,E)$ and an integer $K > 0$**
- **Problem:** **Is there a subset S of V that is a vertex cover of size at most K (every edge in E has at least one end point in S)**
  - For all $(a,b) \in E$, $a \in S$ or $b \in S$
- **Theorem:** **Vertex Cover is NP-Complete**
- **Proof:**
  - First, obviously VC is in NP
    - ***Verification algorithm ?***
  - For NP-hardness proof
    - Reduce IS to VC
    - Reduce inputs so that solving VC solves IS



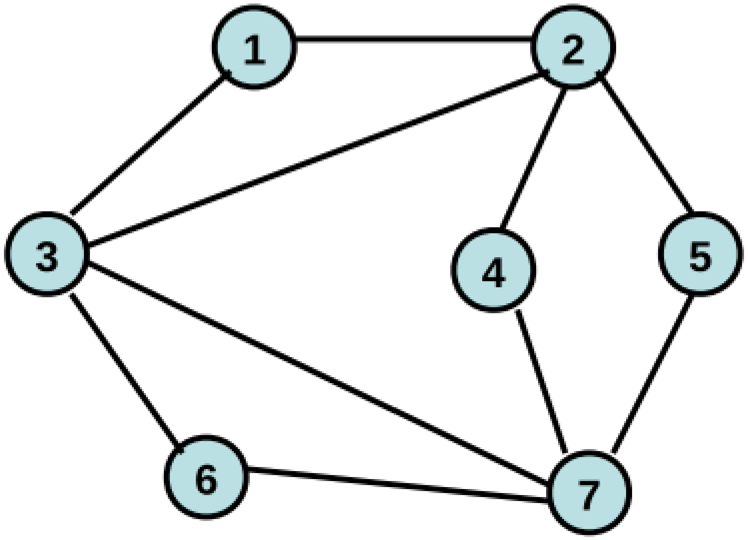
INDEPENDENT SET ≤ VERTEX COVER

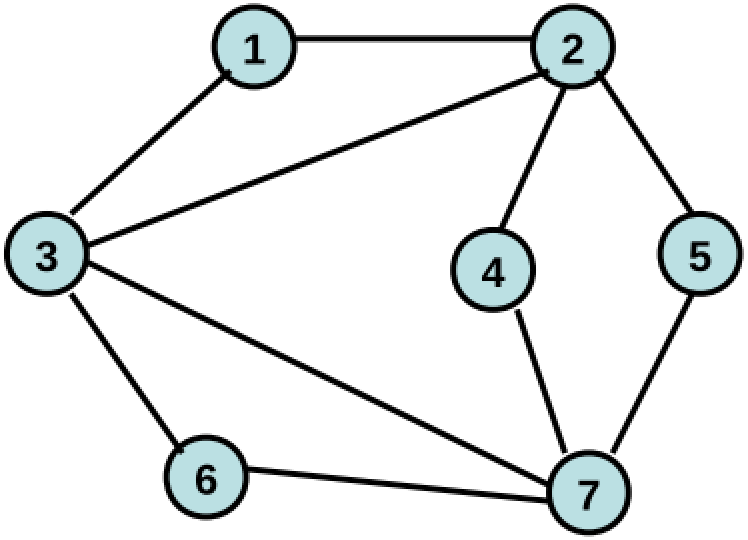
- **Lemma: S is independent set in graph G iff V-S is a vertex cover in G**
- **Proof:**
  - S independent set => V-S vertex cover
    - If S in independent set in G, there are no edges between vertices in S.
    - So all edges must have an end point in a vertex outside of S.
    - Thus, V-S is a vertex cover.
  - S independent set <= V-S vertex cover
    - If V-S is a vertex cover in G, then all edges must have at least one end point in V-S.
    - So any pair of vertices out of V-S cannot have an edge between them otherwise that edge wouldn’t be covered.
    - Thus, S is independent set.

INDEPENDENT SET ≤ VERTEX COVER

EXERCISE:
Find maximum independent set S

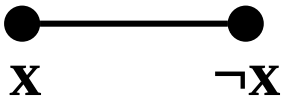
Show that V-S is a vertex cover



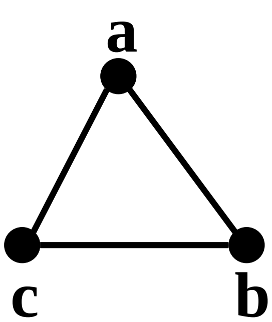


3-SAT ≤ VERTEX COVER

- **Reduce 3SAT to VERTEX-COVER:**
  - We have a Boolean formula in CNF with each clause having 3 literals.
  - For each variable x, create a node for x and ¬x, and connect them with an edge:

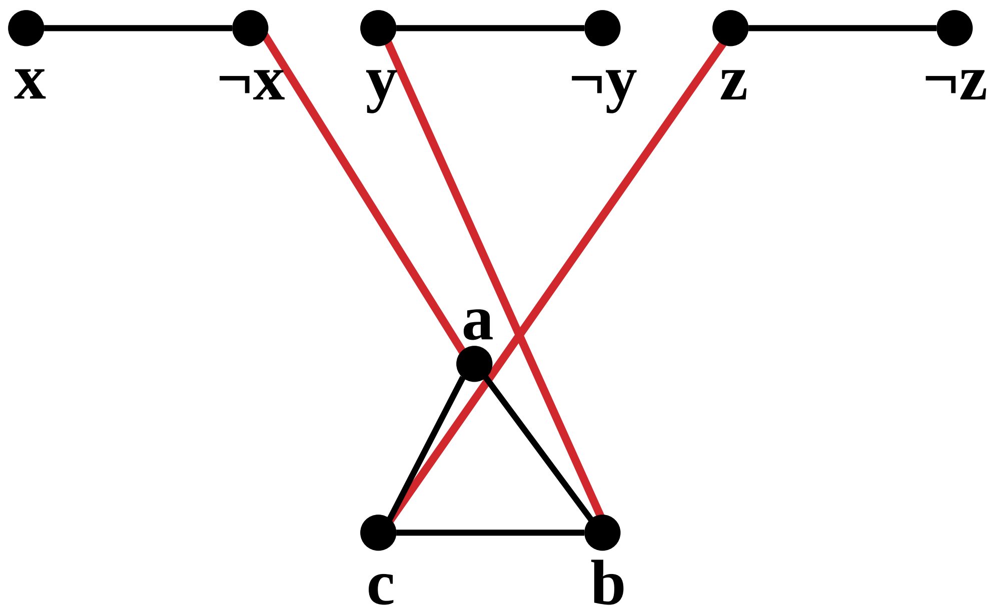


  - For each clause (a ∨ b ∨ c), create three nodes and connect them as a triangle.



3-SAT ≤ VERTEX COVER

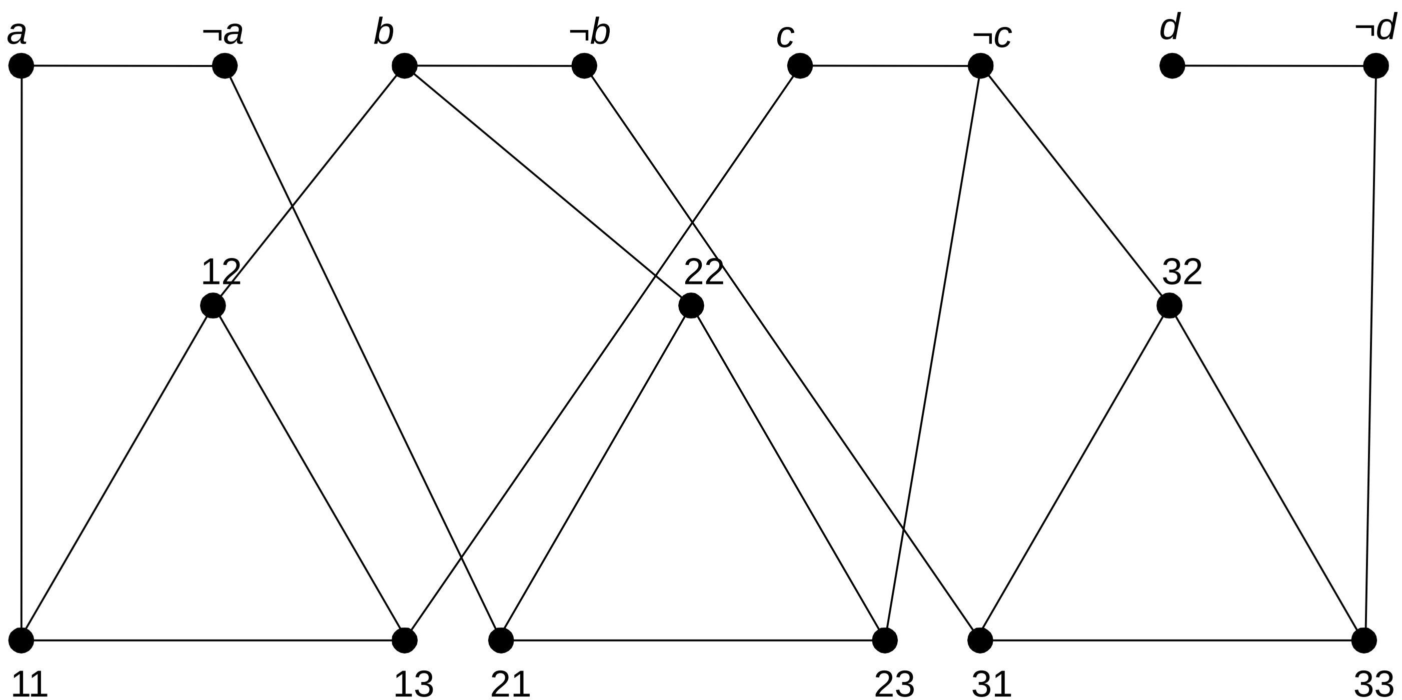
- **Reduction (continued):**
  - Connect each literal in a clause triangle to its copy in a variable pair.
    - e.g., a clause (¬x ∨ y ∨ z)



  - Let n = # variables
  - Let m = # clauses
  - Set K=n+2m

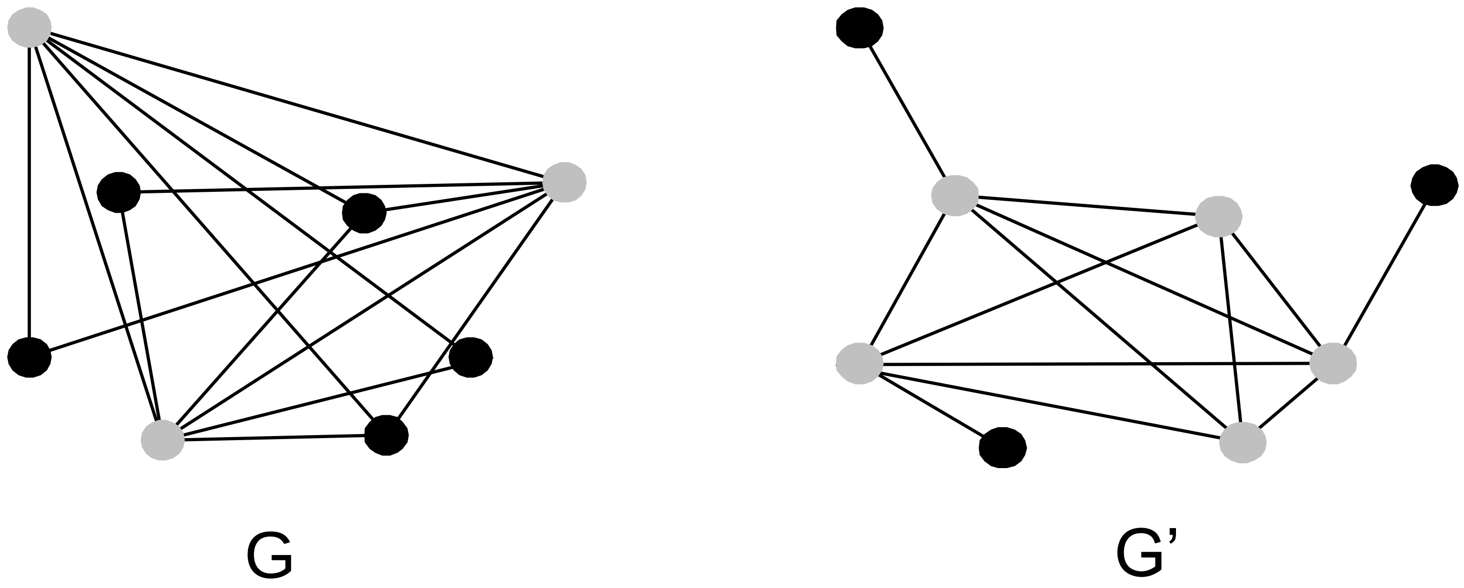
3-SAT ≤ VERTEX COVER

- **Example: (a ∨ b ∨ c) ∧ (¬a ∨ b ∨ ¬c) ∧ (¬b ∨ ¬c ∨ ¬d)**
- **Graph has vertex cover of size K=4+6=10 iff formula is satisfiable.**



VERTEX COVER ≤ CLIQUE

- Reduce VERTEX-COVER to CLIQUE.
- A graph G has a vertex cover of size K if and only if its complement graph G' has a clique of size n-K.



OTHER NP-COMPLETE PROBLEMS

- **SET-COVER: Given a collection of m sets, are there K of these sets whose union is the same as the whole collection of m sets?**
  - NP-complete by reduction from VERTEX-COVER
- **SUBSET-SUM: Given a set of integers and a distinguished integer K, is there a subset of the integers that sums to K?**
  - NP-complete by reduction from VERTEX-COVER

OTHER NP-COMPLETE PROBLEMS

- **0-1 Knapsack: Given a collection of items with weights and benefits, is there a subset of weight at most W and benefit at least K?**
  - NP-complete by reduction from SUBSET-SUM
- **Hamiltonian-Cycle: Given an graph G, is there a cycle in G that visits each vertex exactly once?**
  - NP-complete by reduction from VERTEX-COVER
- **TSP: Given a complete weighted graph G, is there a simple cycle that visits each vertex and has total cost at most K?**
  - NP-complete by reduction from Hamiltonian-Cycle.

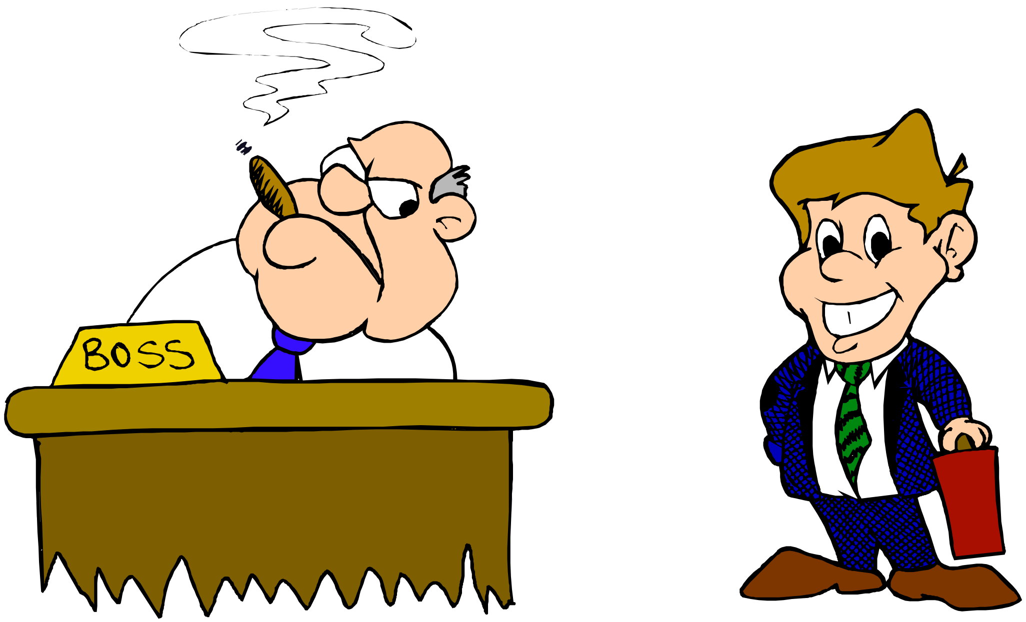
DEALING WITH NP-COMPLETENESS

I couldn't find a polynomial-time algorithm.
I guess I'm not smart enough.

(cartoon inspired by [Garey-Johnson, 79])

DEALING WITH NP-COMPLETENESS

I couldn't find a polynomial-time algorithm, because no such algorithm exists!

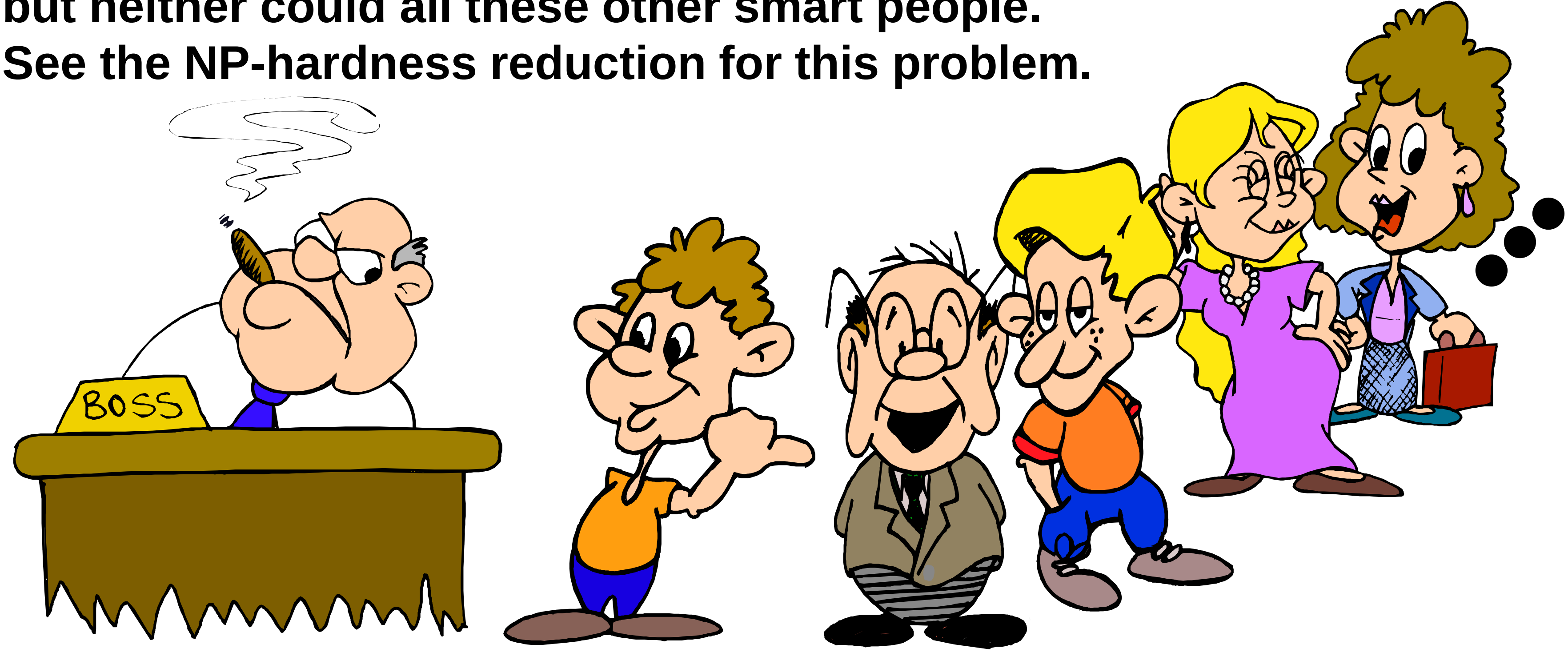


(cartoon inspired by [Garey-Johnson, 79])

Sometimes we can prove a strong lower bound… (but not usually)

DEALING WITH NP-COMPLETENESS

I couldn't find a polynomial-time algorithm, but neither could all these other smart people. See the NP-hardness reduction for this problem.



(cartoon inspired by [Garey-Johnson, 79])

DEALING WITH NP-COMPLETENESS

- **If you recognize your problem is NP-complete, do not waste your time trying to solve it in polynomial time.**
- **Options:**
  - Use slow algorithm
  - Settle for an approximation of the solution
    - This applies to the search and optimization versions of the NP-complete decision problem
  - Change your problem formulation so this updated version is in P.
    - Sometimes special cases are surprisingly easy.

REMARKS

- **Special cases of a problem may be in P even if the general case is not known to be.**
  - 2-SAT (P) vs. 3-SAT (NP-complete)
- **Representation is important: Sometimes if inputs to a problem (or even just part of the input) is specified not in binary but say in unary, then it becomes much easier.**
  - Clique when K is in unary can be solved in polynomial n^K time.
  - Length of input K when represented in unary is K, instead of log K in binary.
- **Fixed vs. Growing:** Sometimes a portion of the input is really the same for all inputs (i.e., fixed), and that can make the problem tractable.
  - Integer programming with a fixed number of variables is in P
  - Clique with k=3 (i.e., deciding if there exists a triangle in the graph) is in P

NP VS. NP-COMPLETENESS

- **Graph Isomorphism: Given two graphs $G_1=(V_1,E_1)$ and $G_2=(V_2,E_2)$, is there a mapping $f: V_1 \rightarrow V_2$ such that $(u,v) \in E_1$ iff $(f(u) , f(v)) \in E_2$**
- **Important Problem!**
- **Best known algorithm: $O(2^{\sqrt{V}})$**
- **Obviously in NP**
  - ***Verification algorithm ?***
- **But probably not NP-complete**
  - Definitely not known to be NP-complete
  - Similarly not known to be NP-complete: Factoring Problem

BEYOND NP

- Can you verify quickly that a 3-CNF formula is not satisfiable?
  - *What would be a certificate?*
- Can you verify quickly that a graph G does not have large clique?
  - *What would be a certificate?*
- Can you verify quickly that a pair of graphs are not isomorphic?
  - *What would be a certificate?*
- Best we know: exponential-size certificates for all these

http://introcs.cs.princeton.edu/java/76computability/

CO-NP

- **NP: Decision problems that have a polynomial-size certificate that can be verified in polynomial-time for yes-instances.**
  - Ex: SAT, HAM-CYCLE, COMPOSITES
- **Complexity Class co-NP: Set containing complements of decision problems in NP. co-NP problems have a polynomial-size certificate that can be verified in polynomial-time for no-instances.**
  - Ex: TAUTOLOGY, NO-HAM-CYCLE, PRIMES

POSSIBLE OPTIONS (NONE PROVEN YET)