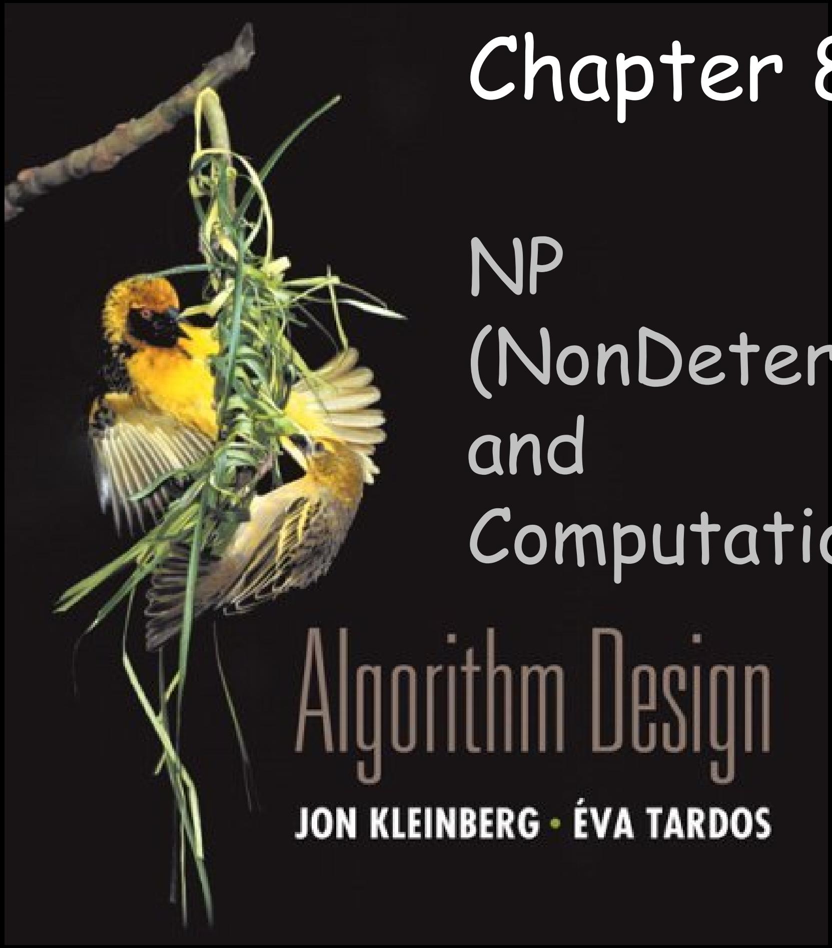


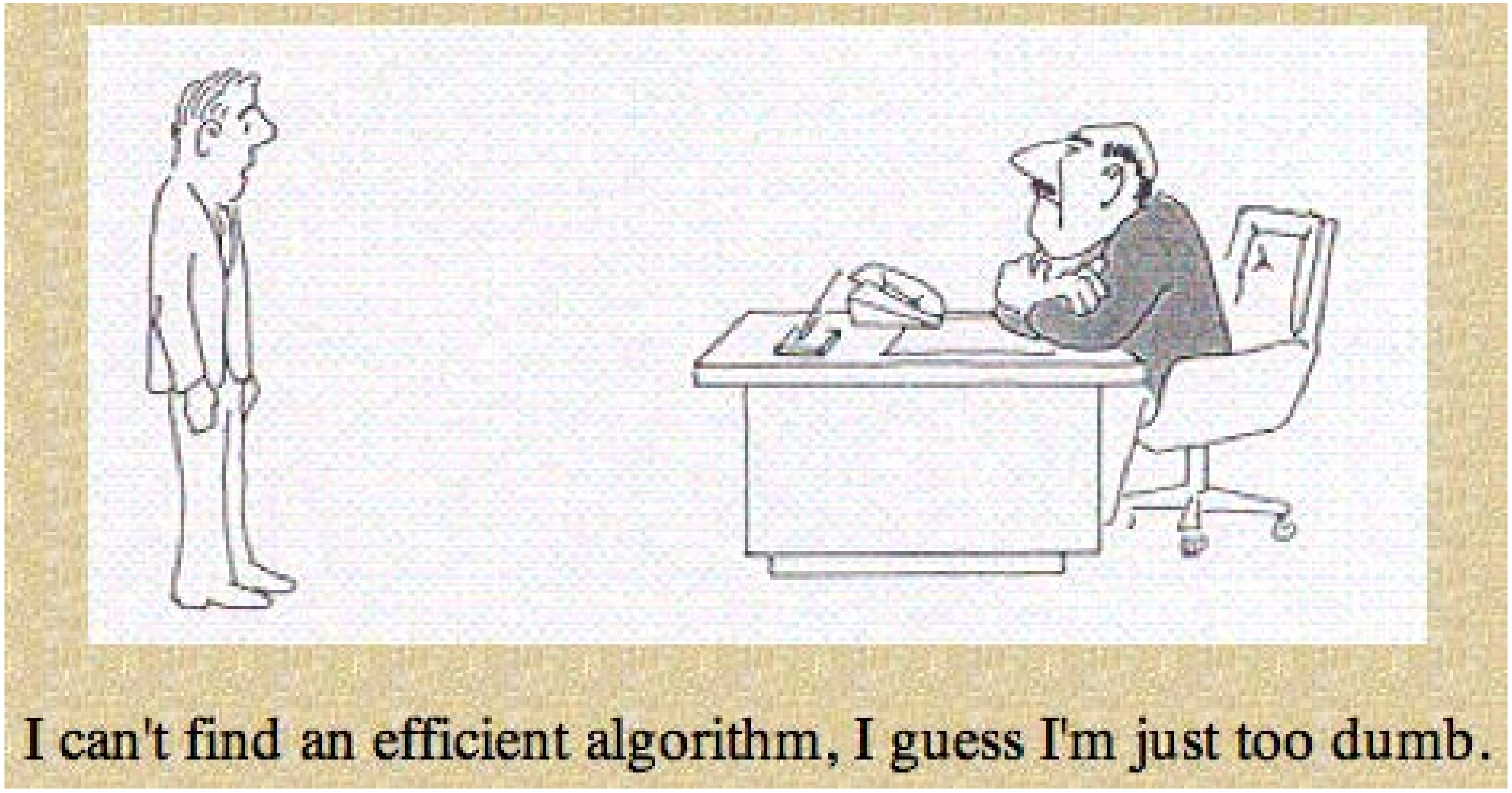
Chapter 8

NP (NonDeterministic Polynomial) and Computational Intractability

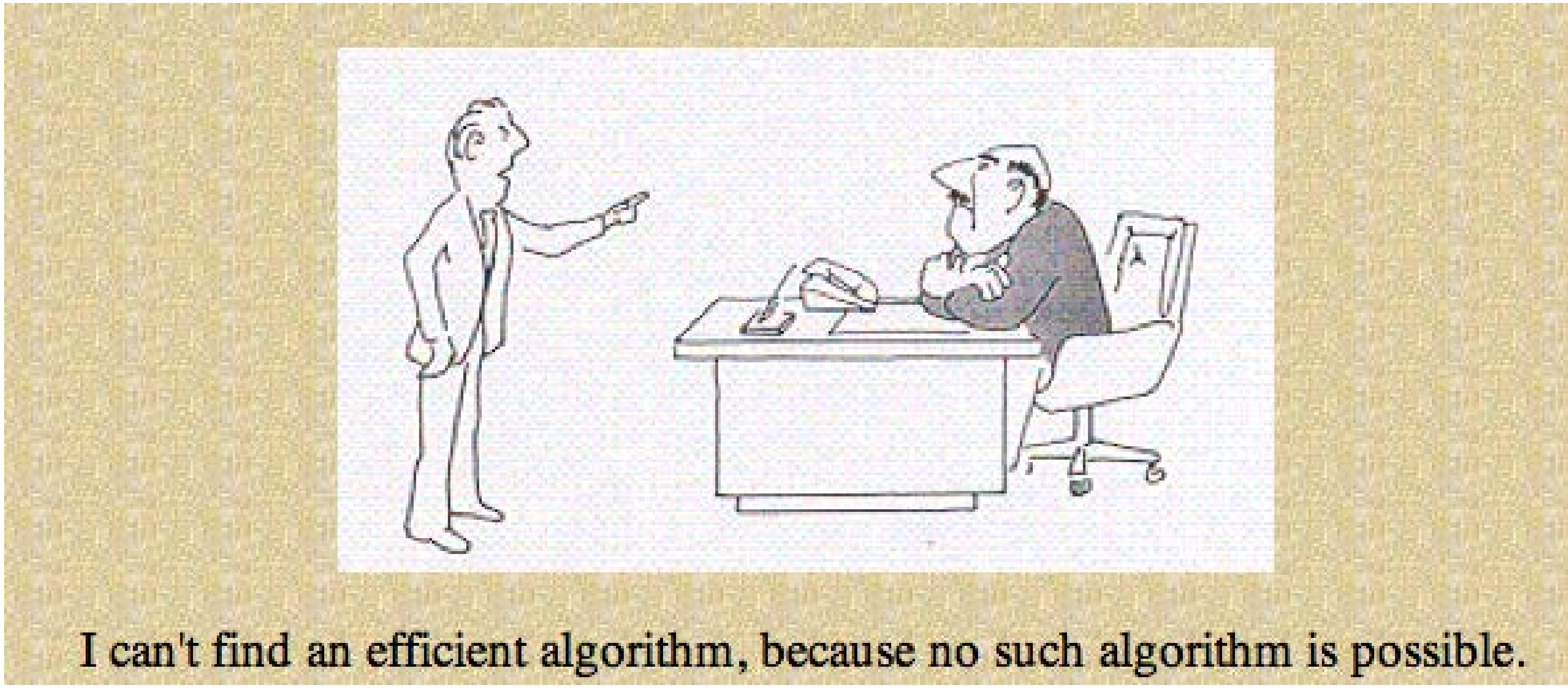
Algorithm Design Patterns and Anti-Patterns

Algorithm design patterns.	Ex.
▪ Greed.	$O(n \log n)$ interval scheduling.
▪ Divide-and-conquer.	$O(n \log n)$ mergesort, FFT.
▪ Dynamic programming.	$\Theta(n W)$ Knapsack, $O(n^2)$ edit distance.
▪ Duality.	$O(n^3)$ bipartite matching.
▪ Reductions.	
▪ Local search.	
▪ Randomization.	

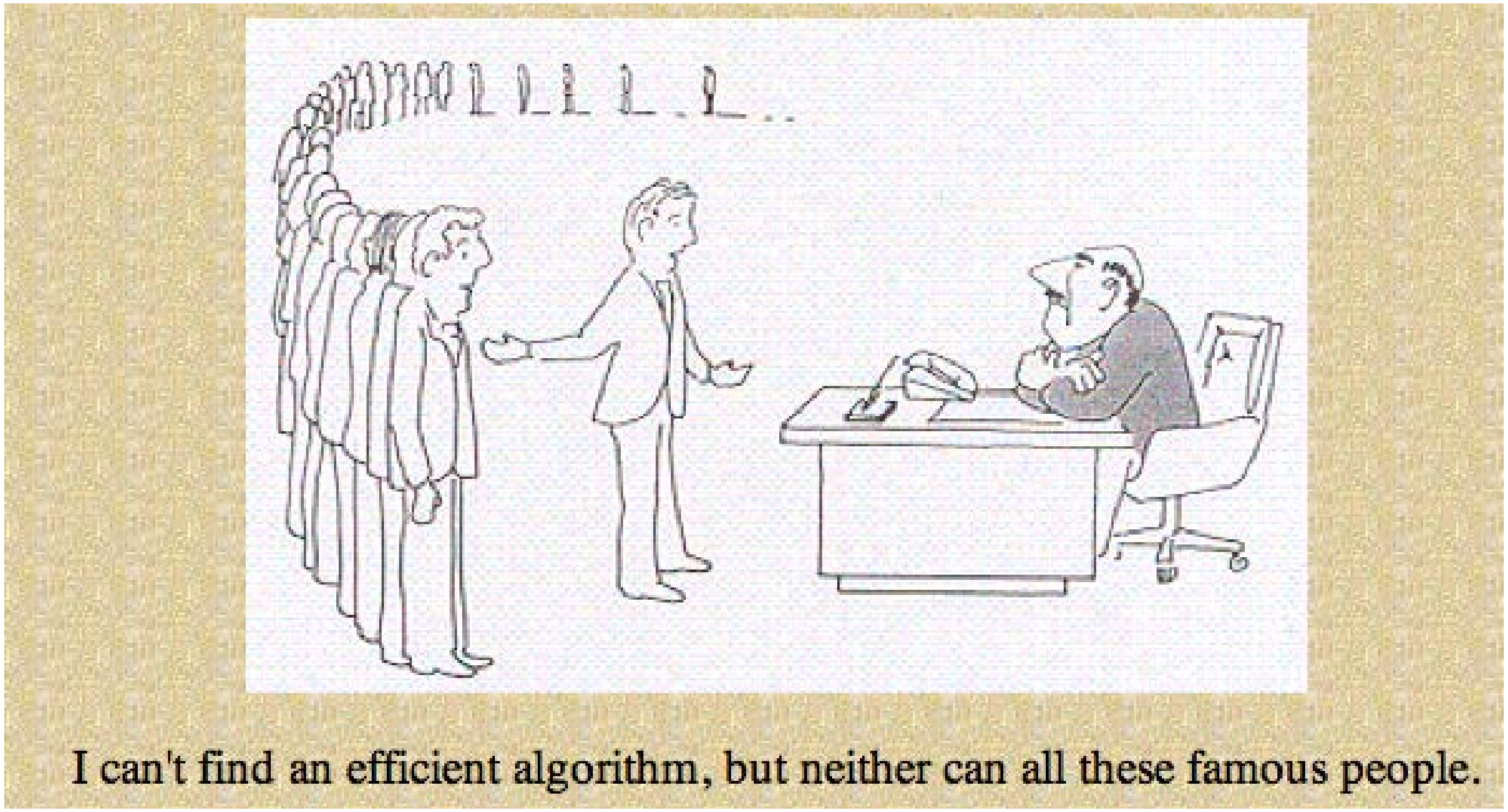
Algorithm design anti-patterns.	
▪ NP-completeness.	$O(n^k)$ algorithm unlikely.
▪ PSPACE-completeness.	$O(n^k)$ certification algorithm unlikely.
▪ Undecidability.	No algorithm possible.


I can't find an efficient algorithm, I guess I'm just too dumb.

http://max.cs.kzoo.edu/~kschultz/CS510/ClassPresentations/NPCartoons.html


I can't find an efficient algorithm, because no such algorithm is possible.

http://max.cs.kzoo.edu/~kschultz/CS510/ClassPresentations/NPCartoons.html

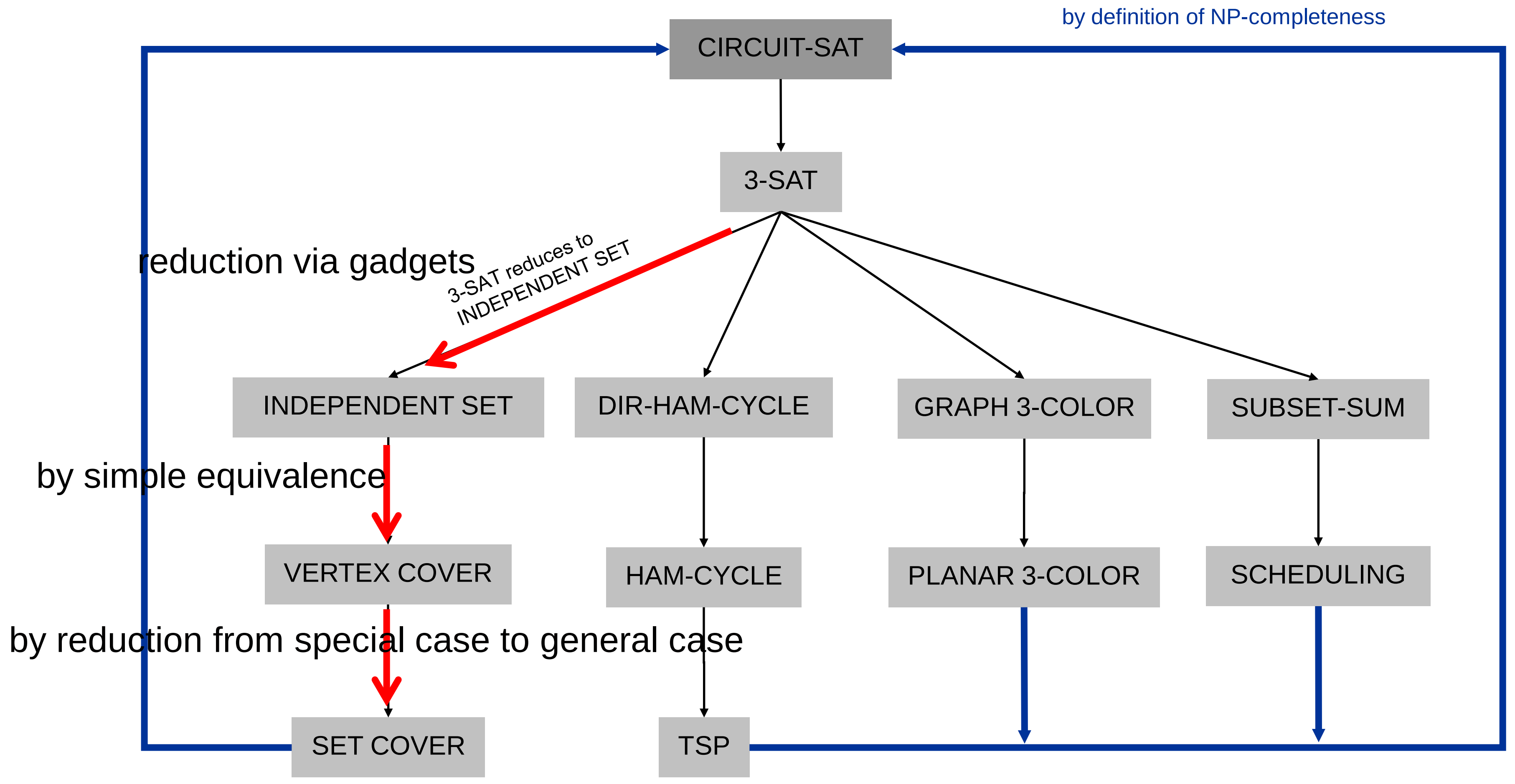

I can't find an efficient algorithm, but neither can all these famous people.

http://max.cs.kzoo.edu/~kschultz/CS510/ClassPresentations/NPCartoons.html

8.1 Polynomial-Time Reductions

Why Poly Time Reductions? Needed for NP-Completeness Results

- **Observation.** All problems below are NP-complete and polynomial reduce to one another!



Classify Problems According to Computational Requirements

Q. Which problems will we be able to solve in practice?

A working definition. [Cobham 1964, Edmonds 1965, Rabin 1966]
Those with polynomial-time algorithms.

Yes	Probably no
Shortest path	Longest path
Matching	3D-matching
Min cut	Max cut
2-SAT	3-SAT
Planar 4-color	Planar 3-color
Bipartite vertex cover	Vertex cover

Primality testing	Factoring

Classify Problems

Desiderata. Classify problems according to those that can be solved in polynomial-time and those that cannot.

Provably requires exponential-time.

- Given a Turing machine, does it halt in at most k 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.

This chapter. Show that these fundamental problems are "computationally equivalent" and appear to be different manifestations of one really hard problem.

Polynomial-Time Reduction

Desiderata'. Suppose we could solve X in polynomial-time. What else could we solve in polynomial time?

don't confuse with reduces from

Reduction. Problem X polynomial reduces to problem Y if arbitrary instances of problem X can be solved using:

- Polynomial number of standard computational steps, plus
- Polynomial number of calls to oracle that solves problem Y.

computational model supplemented by special piece of hardware that solves instances of Y in a single step

Notation. $X \leq_P Y$.

Remarks.

- We pay for time to write down instances sent to black box $\Rightarrow$ instances of Y must be of polynomial size.
- Note: Cook reducibility. $\longleftarrow$ in contrast to Karp reductions

Wikipedia:
polynomial-time reduction is a reduction which is computable by a deterministic Turing machine in polynomial time. If it is a many-one reduction, it is called a **polynomial-time many-one reduction**, **polynomial transformation**, or **Karp reduction**. I
f it is a Turing reduction, it is called a **polynomial-time Turing reduction** or **Cook reduction.**

Polynomial-Time Reduction

Purpose. Classify problems according to relative difficulty.

Design algorithms. If $X \le_P Y$ and Y can be solved in polynomial-time, then X can also be solved in polynomial time.

Establish intractability. If $X \le_P Y$ and X cannot be solved in polynomial-time, then Y cannot be solved in polynomial time.

Establish $\underset{\substack{\uparrow \\ \text{up to cost of reduction}}}{\text{equivalence}}$. If $X \le_P Y$ and $Y \le_P X$, we use notation $X \equiv_P Y$.

Reduction By Simple Equivalence

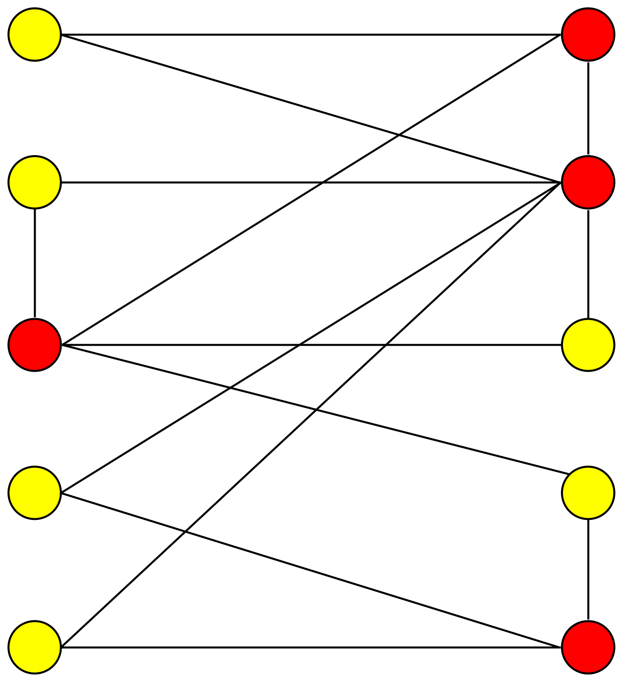
Basic reduction strategies.

- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction by encoding with gadgets.

Independent Set

INDEPENDENT SET: Given a graph $G = (V, E)$ and an integer k, is there a subset of vertices $S \subseteq V$ such that $|S| \geq k$, and for each edge at most one of its endpoints is in S? (No two vertices in S joined by an edge)

Ex. Is there an independent set of size ≥ 6? Yes.
Ex. Is there an independent set of size ≥ 7? No.

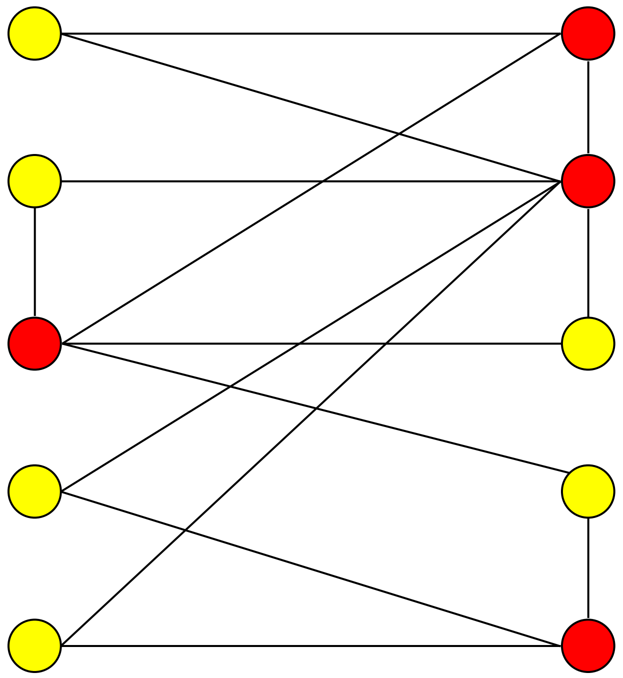


🟡 independent set

Vertex Cover

VERTEX COVER: Given a graph $G = (V, E)$ and an integer k, is there a subset of vertices $S \subseteq V$ such that $|S| \leq k$, and for each edge, at least one of its endpoints is in S?

Ex. Is there a vertex cover of size ≤ 4? Yes.
Ex. Is there a vertex cover of size ≤ 3? No.

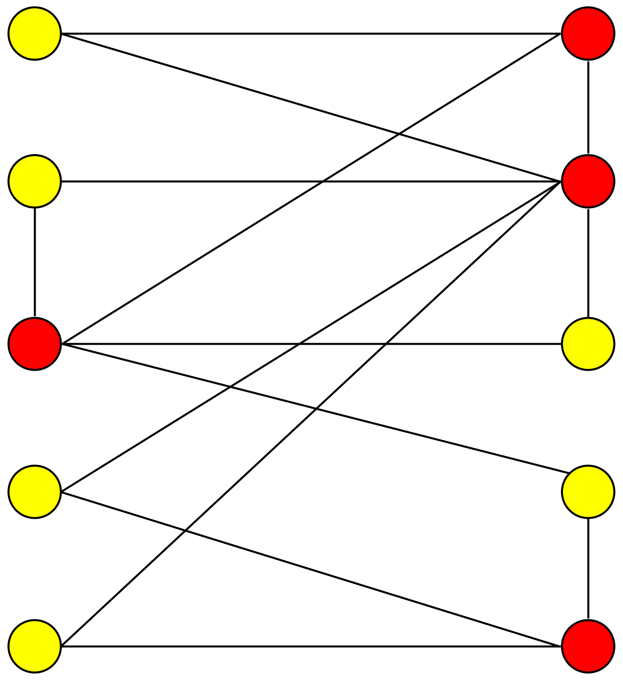


🟥 vertex cover

Vertex Cover and Independent Set

Claim. VERTEX-COVER $\equiv_P$ INDEPENDENT-SET.

Pf. We show S is an independent set iff V – S is a vertex cover.



🟡 independent set

🔴 vertex cover

Vertex Cover and Independent Set

Claim. VERTEX-COVER $\equiv_P$ INDEPENDENT-SET.

Pf. We show S is an independent set iff V – S is a vertex cover.

$\Rightarrow$

- Let S be any independent set.
- Consider an arbitrary edge (u, v).
- S independent $\Rightarrow$ u $\notin$ S or v $\notin$ S $\Rightarrow$ u $\in$ V – S or v $\in$ V – S.
- Thus, V – S covers (u, v).

$\Leftarrow$

- Let V – S be any vertex cover.
- Consider two nodes u $\in$ S and v $\in$ S.
- Observe that (u, v) $\notin$ E since V – S is a vertex cover.
- Thus, no two nodes in S are joined by an edge $\Rightarrow$ S independent set. ▪

Reduction from Special Case to General Case

Basic reduction strategies.

- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction by encoding with gadgets.

Set Cover

SET COVER: Given a set U of elements, a collection $S_1, S_2, \ldots, S_m$ of subsets of U, and an integer k, does there exist a collection of $\leq$ k of these sets whose union is equal to U?

Sample application.
- m available pieces of software.
- Set U of n capabilities that we would like our system to have.
- The ith piece of software provides the set $S_i \subseteq U$ of capabilities.
- Goal: achieve all n capabilities using fewest pieces of software.

Ex:

$U = \{ 1, 2, 3, 4, 5, 6, 7 \}$

$k = 2$

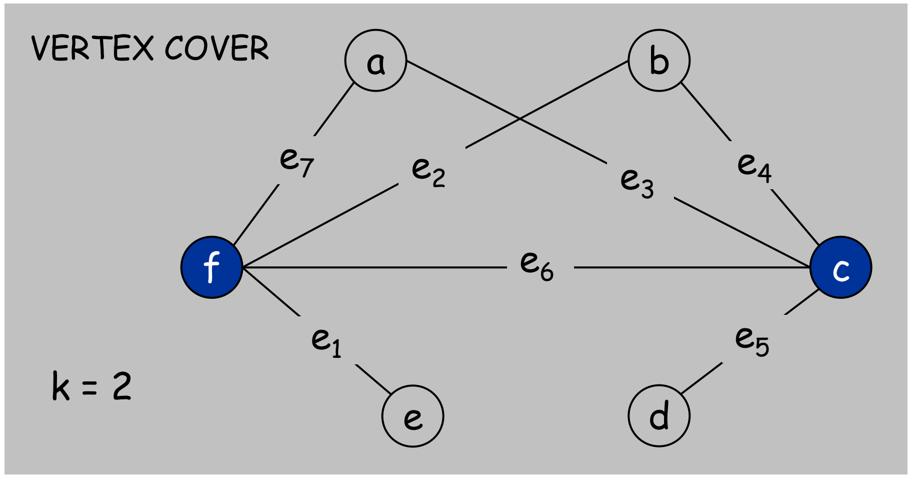
$S_1 = \{3, 7\}$	$S_4 = \{2, 4\}$
$S_2 = \{3, 4, 5, 6\}$	$S_5 = \{5\}$
$S_3 = \{1\}$	$S_6 = \{1, 2, 6, 7\}$

Vertex Cover Reduces to Set Cover

Claim. VERTEX-COVER $\le_P$ SET-COVER.
Pf. Given a VERTEX-COVER instance $G = (V, E)$, k, we construct a set cover instance whose size equals the size of the vertex cover instance.

Construction.

- Create SET-COVER instance:
  - $k = k$, $U = E$, $S_v = \{e \in E : e \text{ incident to } v\}$
- Set-cover of size $\le k$ iff vertex cover of size $\le k$. ▪



SET COVER

$U = \{ 1, 2, 3, 4, 5, 6, 7 \}$
$k = 2$

$S_a = \{3, 7\}$	$S_b = \{2, 4\}$
$S_c = \{3, 4, 5, 6\}$	$S_d = \{5\}$
$S_e = \{1\}$	$S_f = \{1, 2, 6, 7\}$

Polynomial-Time Reduction

Basic strategies.
- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction by encoding with gadgets.

8.2 Reductions via "Gadgets"

Basic reduction strategies.

- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction via "gadgets."

Satisfiability

Literal: A Boolean variable or its negation. x_i or $\overline{x_i}$

Clause: A disjunction of literals. $C_j = x_1 \vee \overline{x_2} \vee x_3$

Conjunctive normal form: A propositional formula Φ that is the conjunction of clauses. $\Phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$

SAT: Given CNF formula Φ, does it have a satisfying truth assignment?

3-SAT: SAT where each clause contains exactly 3 literals.

$\uparrow$ each corresponds to a different variable

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

Yes: x_1 = true, x_2 = true x_3 = false.

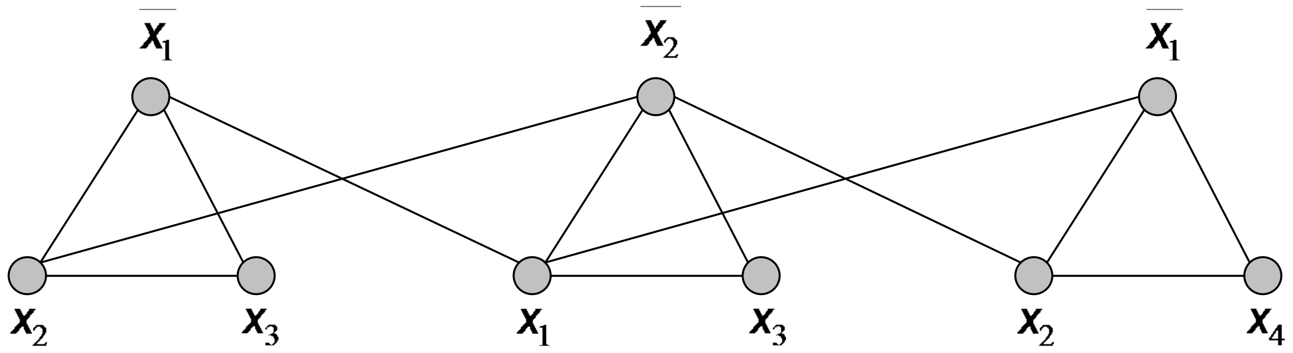
3 Satisfiability Reduces to Independent Set

Claim. 3-SAT $\le_P$ INDEPENDENT-SET.
Pf. Given an instance Φ of 3-SAT, we construct an instance (G, k) of INDEPENDENT-SET that has an independent set of size k iff Φ is satisfiable.

Construction.

- G contains 3 vertices for each clause, one for each literal.
- Connect 3 literals in a clause in a triangle.
- Connect literal to each of its negations.

G



k = 3

$$\Phi = \left(\overline{x_1} \vee x_2 \vee x_3\right) \wedge \left(x_1 \vee \overline{x_2} \vee x_3\right) \wedge \left(\overline{x_1} \vee x_2 \vee x_4\right)$$

3 Satisfiability Reduces to Independent Set

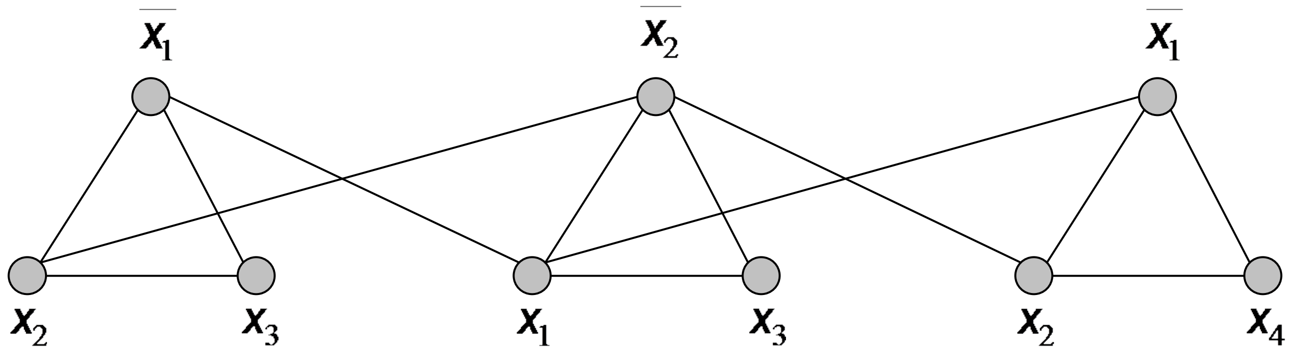
Claim. G contains independent set of size k = $|\Phi|$ iff Φ is satisfiable.

Pf. $\Rightarrow$ Let S be independent set of size k.

- S must contain exactly one vertex in each triangle.
- Set these literals to true. $\longleftarrow$ and any other variables in a consistent way
- Truth assignment is consistent and all clauses are satisfied.

Pf $\Leftarrow$ Given satisfying assignment, select one true literal from each triangle. This is an independent set of size k. ▪

G



k = 3

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

Review

Basic reduction strategies.

- Simple equivalence: INDEPENDENT-SET $\equiv_P$ VERTEX-COVER.
- Special case to general case: VERTEX-COVER $\le_P$ SET-COVER.
- Encoding with gadgets: 3-SAT $\le_P$ INDEPENDENT-SET.

Transitivity. If $X \le_P Y$ and $Y \le_P Z$, then $X \le_P Z$.
Pf idea. Compose the two algorithms.

Ex: 3-SAT $\le_P$ INDEPENDENT-SET $\le_P$ VERTEX-COVER $\le_P$ SET-COVER.

8.3 Definition of NP

8.3 Efficient Certification and Definition of NP

How should we formalize that a solution to a problem can be checked efficiently independently of whether it can be solved efficiently?

Contrast between finding a solution and checking a proposed solution.

If a 3-SAT instance is satisfiable, and if someone gives us a solution, we can plug it into the clauses and check that they are all satisfied.

Decision Problems

- Decision problem.
  - X is a set of strings.
  - Instance: string s.
  - Algorithm A solves problem X: A(s) = yes iff $s \in X$.

- Polynomial time. Algorithm A runs in poly-time if for every string s, A(s) terminates in at most $p(|s|)$ "steps", where $p(\cdot)$ is some polynomial.

  $\uparrow$ length of s

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

  (Decision problem: decide yes or no, whether x is a multiple of y.

  Optimization problem: find the solution: *given* x, find its factor y)

NP

- Certification algorithm intuition.
  - Certifier views things from "managerial" viewpoint.
  - Certifier doesn't determine whether $s \in X$ on its own; rather, it checks a proposed proof t that $s \in X$.

- Def. Algorithm $B(s, t)$ is a certifier for problem X if for every string s, $s \in X$ iff there exists a string t such that $B(s, t) =$ yes.

  "certificate" or "witness"

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

  C(s, t) is a poly-time algorithm and $|t| \le p(|s|)$ for some polynomial $p(\cdot)$.

- Remark. NP stands for nondeterministic polynomial-time.

Certifiers and Certificates: 3-Satisfiability

SAT. Given a CNF formula Φ, is there a satisfying assignment?

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

- Certifier. Check that each clause in Φ has at least one true literal.

- Ex.

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

instance s

$$x_1 = 1,\ x_2 = 1,\ x_3 = 0,\ x_4 = 1$$

certificate t

- Conclusion. SAT is in NP.

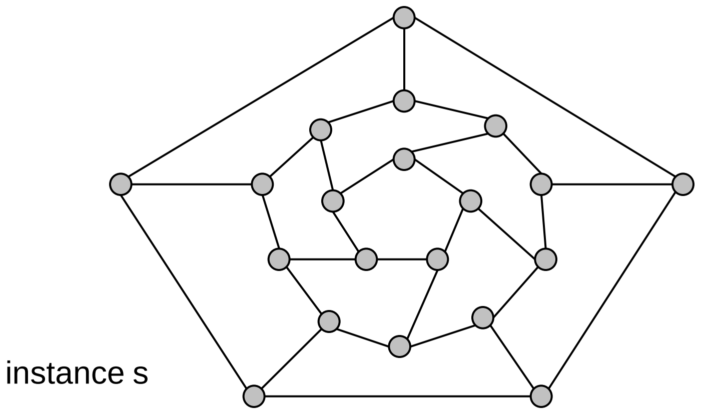
Certifiers and Certificates: Hamiltonian Cycle

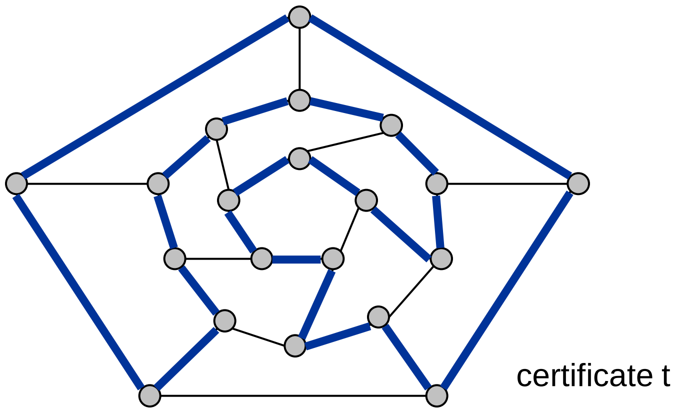
HAM-CYCLE. Given an undirected graph G = (V, E), does there exist a simple cycle C that visits every node?

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.





NP-Complete

- NP-complete. A problem Y in NP with the property that for every problem X in NP, $X \leq_p Y$.

- Theorem. Suppose Y is an NP-complete problem. Then Y is solvable in poly-time iff P = NP.
- Pf. $\Leftarrow$ If P = NP then Y can be solved in poly-time since Y is in NP.
- Pf. $\Rightarrow$ Suppose Y can be solved in poly-time.
  - Let X be any problem in NP. Since $X \leq_p Y$, we can solve X in poly-time. This implies NP $\subseteq$ P.
  - We already know P $\subseteq$ NP. Thus P = NP. ▪

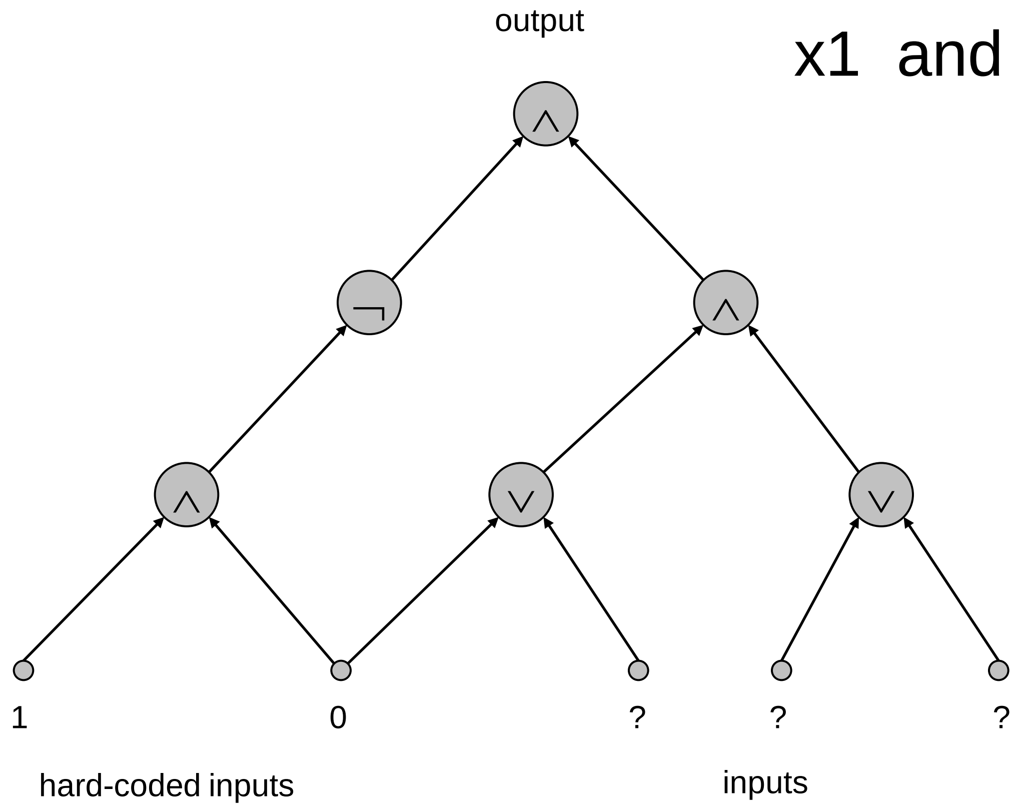
- Fundamental question. Do there exist "natural" NP-complete problems?

Circuit Satisfiability

- **CIRCUIT-SAT.** Given a combinational circuit built out of AND, OR, and NOT gates, is there a way to set the circuit inputs so that the output is 1? (The circuit below has two sources on the left already assigned truth values.)

x1 and (x2 or x3)

yes: 1 0 1



The "First" NP-Complete Problem

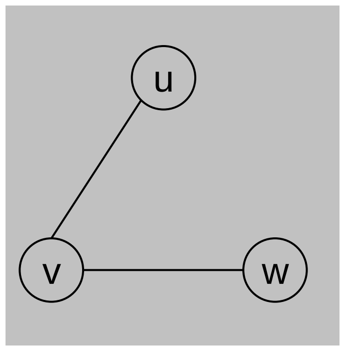
- Theorem. CIRCUIT-SAT is NP-complete. [Cook 1971, Levin 1973]
- Pf. (sketch)
  - Any algorithm that takes a fixed number of bits n as input and produces a yes/no answer can be represented by such a circuit. Moreover, if algorithm takes poly-time, then circuit is of poly-size.

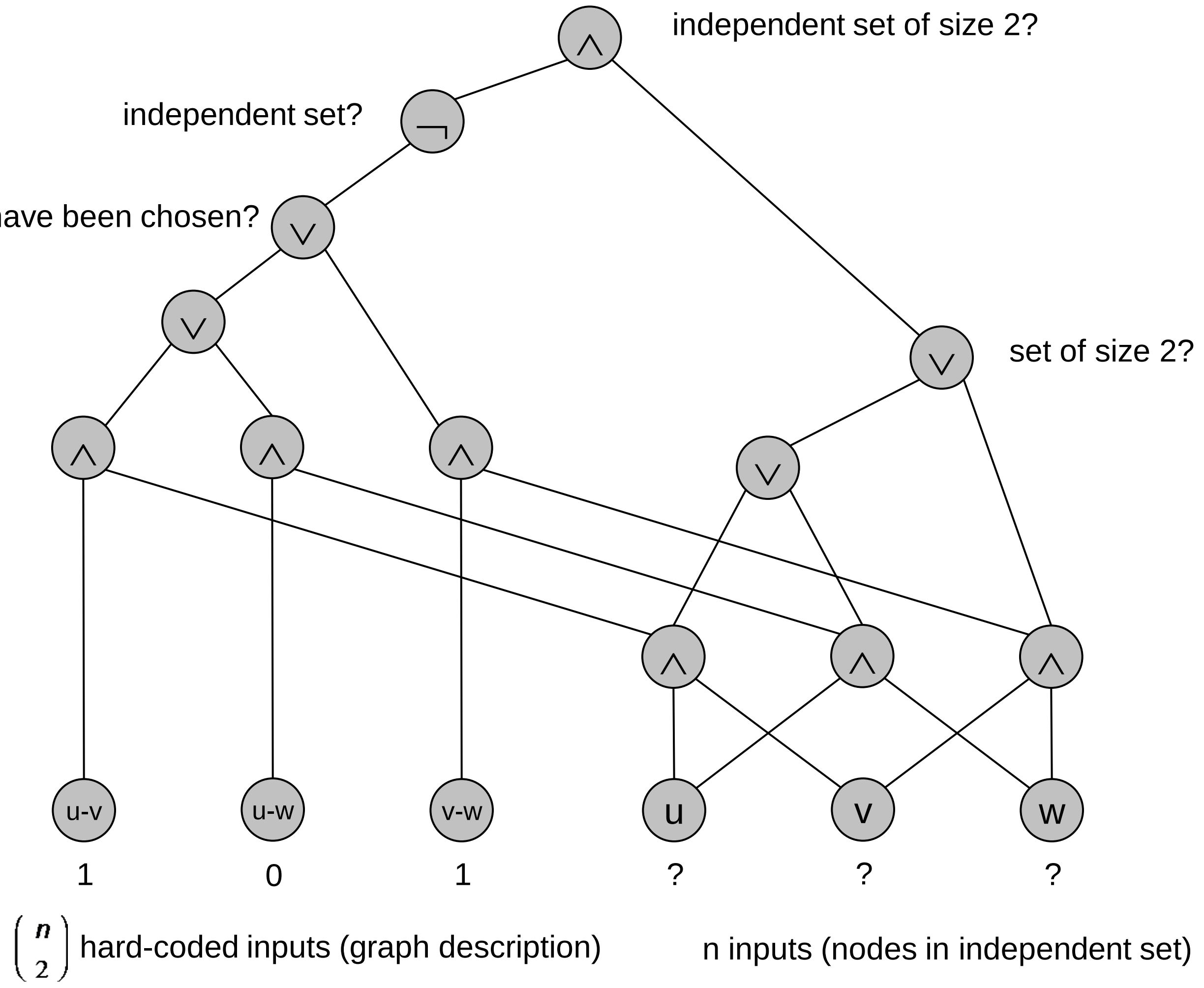
sketchy part of proof; fixing the number of bits is important,
and reflects basic distinction between algorithms and circuits

  - Consider some problem X in NP. It has a poly-time certifier B(s, t). To determine whether s is in X, need to know if there exists a certificate t of length p(|s|) such that B(s, t) = yes.
  - View B(s, t) as an algorithm on |s| + p(|s|) bits (input s, certificate t) and convert it into a poly-size circuit K.
    - first |s| bits are hard-coded with s
    - remaining p(|s|) bits represent bits of t
  - Circuit K is satisfiable iff B(s, t) = yes.

Example

- **Ex.** Construction below creates a circuit K whose inputs can be set so that K outputs true iff graph G has an independent set of size 2.


G = (V, E), n = 3



independent set of size 2?
independent set?
both endpoints of some edge have been chosen?
set of size 2?
u-v
u-w
v-w
u
v
w
1
0
1
?
?
?
$\binom{n}{2}$ hard-coded inputs (graph description)
n inputs (nodes in independent set)

Establishing NP-Completeness

- **Remark.** Once we establish first "natural" NP-complete problem, others fall like dominoes.

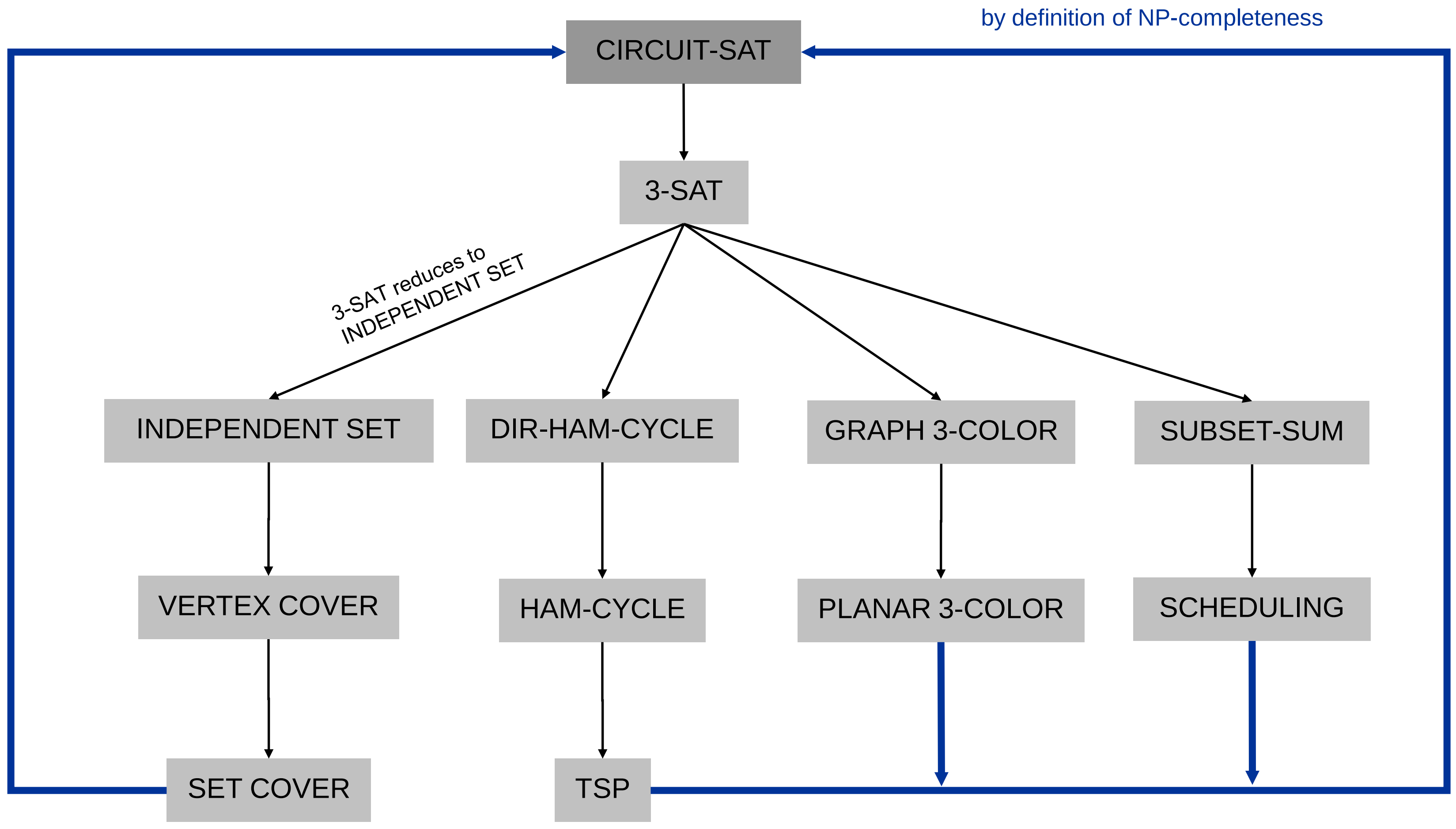
- **Recipe to establish NP-completeness of problem Y.**
  - Step 1. Show that Y is in NP.
  - Step 2. Choose an NP-complete problem X.
  - Step 3. Prove that $X \le_p Y$.

- **Justification.** If X is an NP-complete problem, and Y is a problem in NP with the property that $X \le_P Y$ then Y is NP-complete.

- **Pf.** Let W be any problem in NP. Then $W \underset{\substack{\uparrow \\ \text{by definition of} \\ \text{NP-complete}}}{\le_P} X \underset{\substack{\uparrow \\ \text{by assumption}}}{\le_P} Y.$
  - By transitivity, $W \le_P Y$.
  - Hence Y is NP-complete. ▪

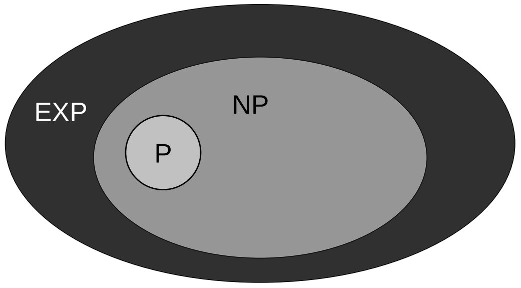
NP-Completeness

- Observation. All problems below are NP-complete and polynomial reduce to one another!

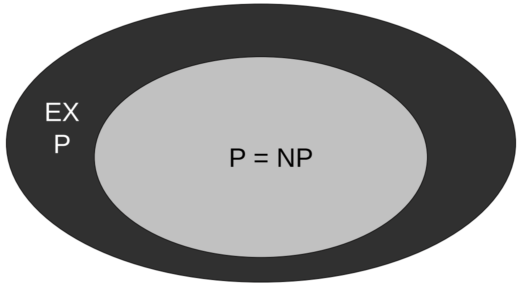


The Main Question: P Versus NP
A Major Unsolved Problem in Computer Science

- Does P = NP? [Cook 1971, Edmonds, Levin, Yablonski, Gödel]
  - Is the decision problem as easy as the certification problem?
  - Clay Mathematics Inst. $1 million prize.



If P ≠ NP



If P = NP

would break RSA cryptography
(and potentially collapse economy)

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

- Consensus opinion on P = NP? Probably no.

See also: Gerhard J. Woeginger, "The P-versus-NP page".

- Some more for the interested students

Some NP-Complete Problems

- Six basic genres of NP-complete problems and paradigmatic examples.
  - Packing problems: SET-PACKING, INDEPENDENT SET.
  - Covering problems: SET-COVER, VERTEX-COVER.
  - Constraint satisfaction problems: SAT, 3-SAT.
  - Sequencing problems: HAMILTONIAN-CYCLE, TSP.
  - Partitioning problems: 3D-MATCHING 3-COLOR.
  - Numerical problems: SUBSET-SUM, KNAPSACK.

- Practice. Most NP problems are either known to be in P or NP-complete.

- Notable exceptions. Factoring, graph isomorphism, Nash equilibrium.

MAJOR UNSOLVED PROBIEMS IN COMPUTER SCIENCE

Computational complexity theory
P = NP problem. This is one of the seven Millennium Prize Problems.
NC = P problem
NP = co-NP problem
P = BPP problem
P = PSPACE problem
What is the relationship between BQP and NP?
Unique games conjecture
Is the exponential time hypothesis true?
Do one-way functions exist?

Algorithms
What is the fastest algorithm for multiplication of two n-digit numbers?
What is the fastest algorithm for matrix multiplication?
Can integer factorization be done in polynomial time?
Can the discrete logarithm be computed in polynomial time?
Can the graph isomorphism problem be solved in polynomial time?
Can parity games be solved in polynomial time?
Does linear programming admit a strongly polynomial-time algorithm? This is problem #9 in Smale's list of problems.
What is the lower bound on the complexity of fast Fourier transform algorithms? Can they be faster than Θ (N log N)?
Can the 3SUM problem be solved in subquadratic time?
Dynamic optimality conjecture for splay trees
K-server problem