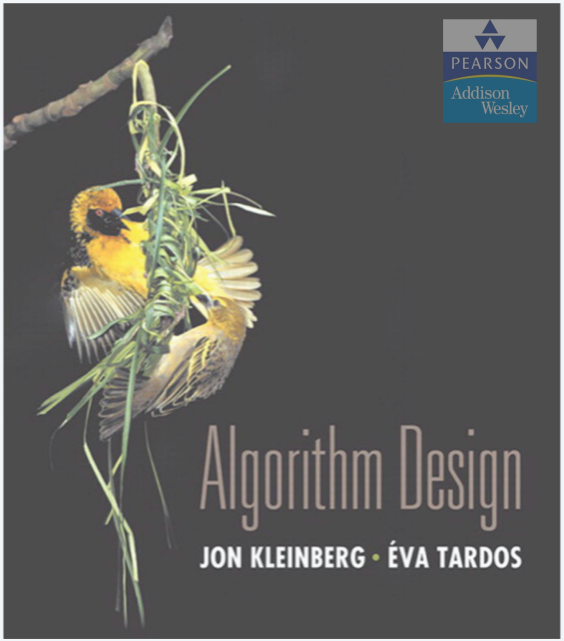
CS101 Algorithms and Data Structures

Reductions, P and NP



SECTION 8.1

REDUCTIONS, P AND NP

- *poly-time reductions*
- *packing and covering problems*
- *constraint satisfaction problems*
- *graph coloring*
- *P vs. NP*
- *NP-complete*

Algorithm design patterns and antipatterns

Algorithm design patterns.

- Greedy.
- Divide and conquer.
- Dynamic programming.
- Reductions.
- Duality.
- Local search.
- Randomization.

Algorithm design antipatterns.

• **NP**-completeness.	$O(n^k)$ algorithm unlikely.
• **PSPACE**-completeness.	$O(n^k)$ certification algorithm unlikely.
• Undecidability.	No algorithm possible.

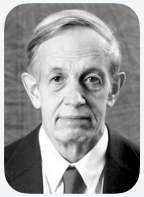
Classify problems according to computational requirements

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

A working definition. Those with poly-time algorithms.

polynomial time.

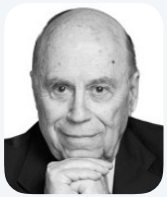

von Neumann
(1953)


Nash
(1955)


Gödel
(1956)


Cobham
(1964)


Edmonds
(1965)


Rabin
(1966)

Theory. Definition is broad and robust.

Turing machine, word RAM, uniform circuits, ...

穷举

Practice. Poly-time algorithms scale to huge problems.

constants tend to be small, e.g., $3n^2$

Classify problems

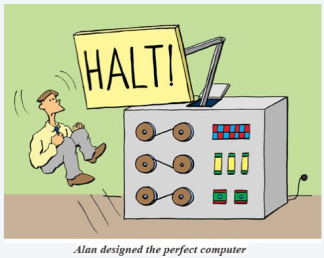
Desiderata. Classify problems according to those that can be solved in polynomial time and those that cannot.

Provably requires exponential time.

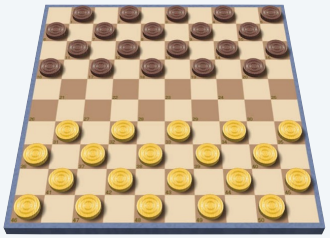
- Given a constant-size program, does it halt in at most k steps?
- Given a board position in an n-by-n generalization of checkers, can black guarantee a win?

input size = $c + \log k$

using forced capture rule



Alan designed the perfect computer



Frustrating news. Huge number of fundamental problems have defied classification for decades.

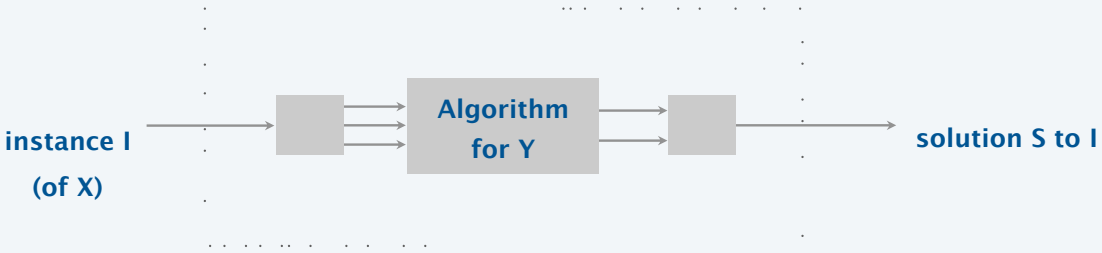
Poly-time reductions

Desiderata'. Suppose we could solve problem Y in polynomial time. What else could we solve in polynomial time?

Reduction. Problem X polynomial-time (Cook) 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



Algorithm for X

Poly-time reductions

Desiderata'. Suppose we could solve problem Y in polynomial time. What else could we solve in polynomial time?

Reduction. Problem X polynomial-time (Cook) 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.

Notation. $X \leq_P Y$.

poly

Note. We pay for time to write down instances of Y sent to oracle $\Rightarrow$ instances of Y must be of polynomial size.

Novice mistake. Confusing $X \leq_P Y$ with $Y \leq_P X$.

Intractability: quiz 1

Suppose that $X \leq_P Y$. Which of the following can we infer?

A. If X can be solved in polynomial time, then so can Y.

B. X can be solved in poly time iff Y can be solved in poly time.

C. If X cannot be solved in polynomial time, then neither can Y.

D. If Y cannot be solved in polynomial time, then neither can X.

Solve Y. Solve X.

Y is harder $X \leq_P Y$

Reductions for Hardness

means X is at least as hard as Y

Theorem

If $Y \leq_P X$ and Y cannot be solved in polynomial time, then X cannot be solved in polynomial time.

means reduce Y to X

Why? If we *could* solve X in polynomial time, then we'd be able to solve Y in polynomial time using the reduction, contradicting the assumption.

So: If we could find one hard problem Y, we could prove that another problem X is hard by reducing Y to X.

Reductions as tool for hardness

$Y \leq_P X$.

We want prove some problems are computationally difficult.

As a first step, we settle for relative judgements:

Problem X is at least as hard as problem Y

To prove such a statement, we reduce problem Y to problem X:

> *If you had a black box that can solve instances of problem X, how can you solve any instance of Y using polynomial number of steps, plus a polynomial number of calls to the black box that solves X?*

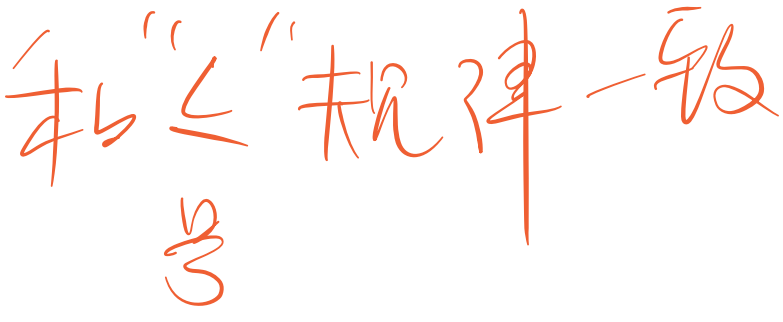
Polynomial Problems

Suppose:

- $Y \leq_P X$, and
- there is an polynomial time algorithm for X.

Then, there is a polynomial time algorithm for Y.

Why?



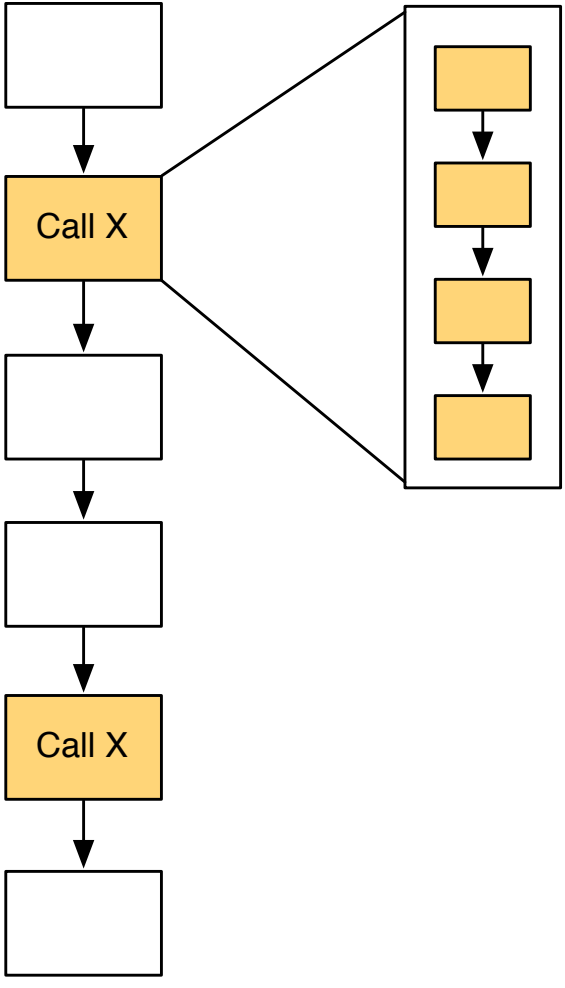
Polynomial Problems

Suppose:

- $Y \leq_P X$, and
- there is an polynomial time algorithm for X.

Then, there is a polynomial time algorithm for Y.

Why? Because polynomials compose.



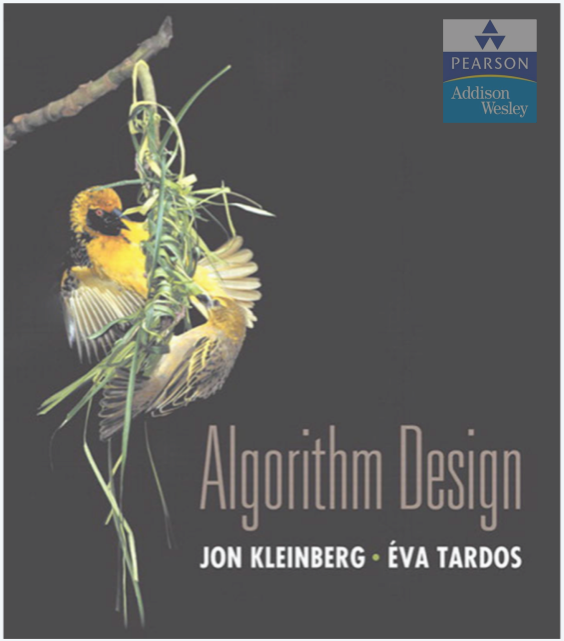
Poly-time reductions

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

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

Establish equivalence. If both $X \leq_P Y$ and $Y \leq_P X$, we use notation $X \equiv_P Y$. In this case, X can be solved in polynomial time iff Y can be.

Bottom line. Reductions classify problems according to relative difficulty.



SECTION 8.1

REDUCTIONS, P AND NP

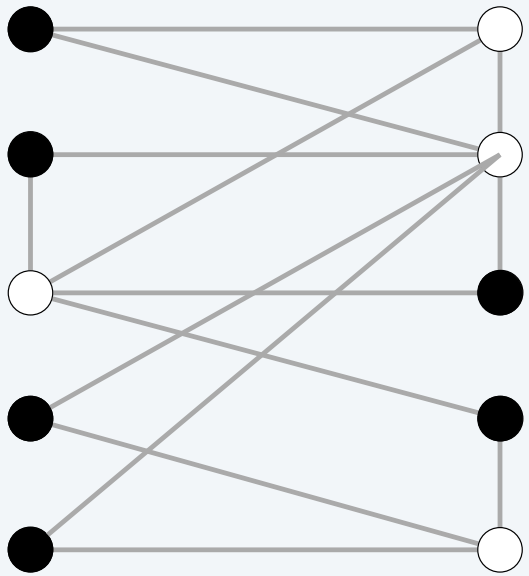
- *poly-time reductions*
- *packing and covering problems*
- *constraint satisfaction problems*
- *graph coloring*
- *P vs. NP*
- *NP-complete*

Independent set

INDEPENDENT-SET. Given a graph $G = (V, E)$ and an integer k, is there a subset of k (or more) vertices such that no two are adjacent?

Ex. Is there an independent set of size ≥ 6? ✓

Ex. Is there an independent set of size ≥ 7? no



independent set of size 6

Vertex Cover

Def. A vertex cover of a graph is a set S of nodes such that every edge has at least one endpoint in S.

In other words, we try to "cover" each of the edges by choosing at least one of its vertices.

Vertex Cover

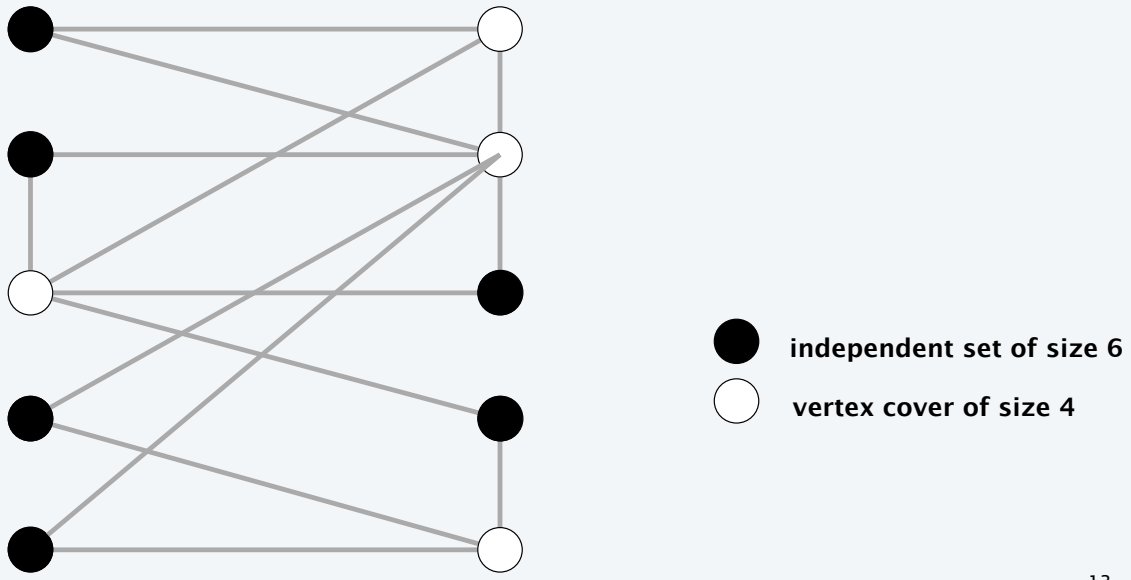
Given a graph G and a number k, does G contain a vertex cover of size at most k.

Vertex cover

VERTEX-COVER. Given a graph $G = (V, E)$ and an integer k, is there a subset of k (or fewer) vertices such that each edge is incident to at least one vertex in the subset?

Ex. Is there a vertex cover of size ≤ 4 ?

Ex. Is there a vertex cover of size ≤ 3 ?



Independent Set to Vertex Cover

Independent Set

Given graph G and a number k, does G contain a set of at least k independent vertices?

Can we reduce independent set to vertex cover?

Vertex Cover

Given a graph G and a number k, does G contain a vertex cover of size at most k.

Relation btw Vertex Cover and Indep. Set

(1) S V-S.

Theorem

If $G = (V, E)$ is a graph, then S is an independent set $\iff$ $V - S$ is a vertex cover.

1 1 2对应

Proof. $\implies$ Suppose S is an independent set, and let $e = (u, v)$ be some edge. Only one of u, v can be in S. Hence, at least one of u, v is in $V - S$. So, $V - S$ is a vertex cover.

$\impliedby$ Suppose $V - S$ is a vertex cover, and let $u, v \in S$. There can't be an edge between u and v (otherwise, that edge wouldn't be covered in $V - S$). So, S is an independent set. $\square$

How to proof

Independent Set $\leq_P$ Vertex Cover

Independent Set $\leq_P$ Vertex Cover

To show this, we change any instance of Independent Set into an instance of Vertex Cover:

- Given an instance of Independent Set $\langle G, k \rangle$,
- We ask our Vertex Cover black box if there is a vertex cover $V - S$ of size $\leq |V| - k$.

By our previous theorem, S is an independent set iff $V - S$ is a vertex cover. If the Vertex Cover black box said:

yes: *then S must be an independent set of size $\geq k$.*
no: *then there is no vertex cover $V - S$ of size $\leq |V| - k$, hence there is no independent set of size $\geq k$.*

Vertex Cover $\leq_P$ Independent Set

Actually, we also have:

$$\text{Vertex Cover} \leq_P \text{Independent Set}$$

Proof. To decide if G has an vertex cover of size k, we ask if it has an independent set of size $n - k$. $\square$

So: Vertex Cover and Independent Set are equivalently difficult.

How to proof equivalently (diff).

Intractability: quiz 2

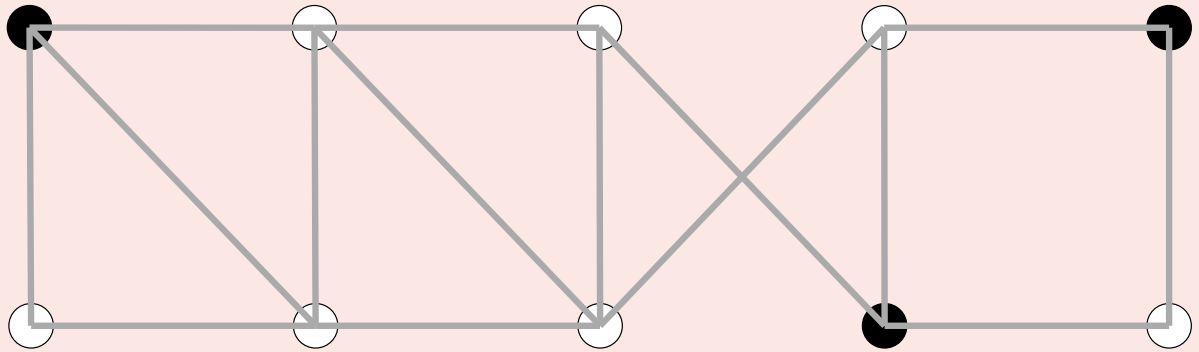
Consider the following graph G. Which are true?

A. The white vertices are a vertex cover of size 7.

B. The black vertices are an independent set of size 3.

C. Both A and B.

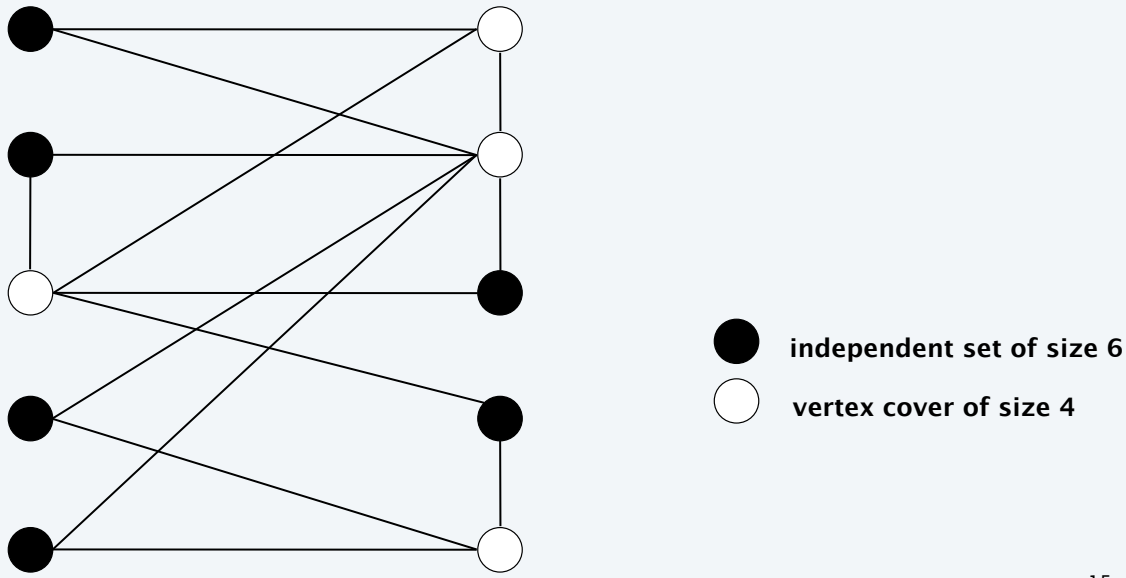
D. Neither A nor B.



Vertex cover and independent set reduce to one another

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

Pf. We show S is an independent set of size k iff $V - S$ is a vertex cover of size $n - k$.



Vertex cover and independent set reduce to one another

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

Pf. We show S is an independent set of size k iff $V - S$ is a vertex cover of size $n - k$.

$\Rightarrow$

- Let S be any independent set of size k.
- $V - S$ is of size $n - k$.
- Consider an arbitrary edge $(u, v) \in E$.
- S independent $\Rightarrow$ either $u \notin S$, or $v \notin S$, or both.
  $\Rightarrow$ either $u \in V - S$, or $v \in V - S$, or both.
- Thus, $V - S$ covers (u, v). ▪

Vertex cover and independent set reduce to one another

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

Pf. We show S is an independent set of size k iff $V - S$ is a vertex cover of size $n - k$.

$\Leftarrow$

- Let $V - S$ be any vertex cover of size $n - k$.
- S is of size k.
- Consider an arbitrary edge $(u, v) \in E$.
- $V - S$ is a vertex cover $\Rightarrow$ either $u \in V - S$, or $v \in V - S$, or both.
  $\Rightarrow$ either $u \notin S$, or $v \notin S$, or both.
- Thus, S is an independent set. ▪

Set cover

SET-COVER. Given a set U of elements, a collection S of subsets of U, and an integer k, are there $\leq k$ of these subsets whose union is equal to U?

set number

Sample application.

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

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

$S_a = \{ 3, 7 \}$ $S_b = \{ 2, 4 \}$

$S_c = \{ 3, 4, 5, 6 \}$ $S_d = \{ 5 \}$

$S_e = \{ 1 \}$ $S_f = \{ 1, 2, 6, 7 \}$

$k = 2$

a set cover instance

Intractability: quiz 3

Given the universe U = { 1, 2, 3, 4, 5, 6, 7 } and the following sets, which is the minimum size of a set cover?

A. 1

B. 2

C. 3

D. None of the above.

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

$S_a = \{ 1, 4, 6 \}$ $S_b = \{ 1, 6, 7 \}$

$S_c = \{ 1, 2, 3, 6 \}$ $S_d = \{ 1, 3, 5, 7 \}$

$S_e = \{ 2, 6, 7 \}$ $S_f = \{ 3, 4, 5 \}$

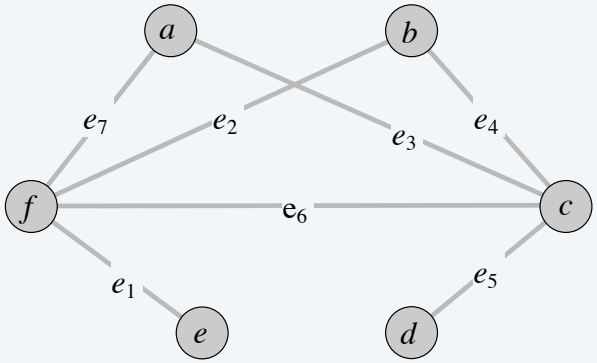
Vertex cover reduces to set cover

Theorem. VERTEX-COVER $\leq_P$ SET-COVER.

Pf. Given a VERTEX-COVER instance $G = (V, E)$ and k, we construct a SET-COVER instance (U, S, k) that has a set cover of size k iff G has a vertex cover of size k.

Construction.

- Universe $U = E$.
- Include one subset for each node $v \in V$: $S_v = \{e \in E : e \text{ incident to } v\}$.



vertex cover instance
(k = 2)

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

$S_a = \{3, 7\}$ $S_b = \{2, 4\}$

$S_c = \{3, 4, 5, 6\}$ $S_d = \{5\}$

$S_e = \{1\}$ $S_f = \{1, 2, 6, 7\}$

set cover instance
(k = 2)

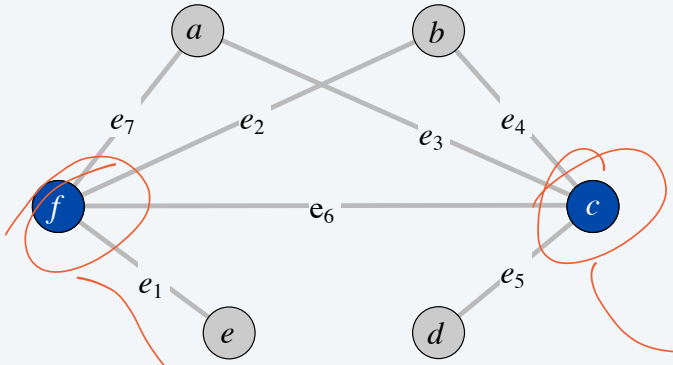
Vertex cover reduces to set cover

Lemma. $G = (V, E)$ contains a vertex cover of size k iff (U, S, k) contains a set cover of size k.

Pf. $\Rightarrow$ Let $X \subseteq V$ be a vertex cover of size k in G.

- Then $Y = \{ S_v : v \in X \}$ is a set cover of size k. ▪

"yes" instances of VERTEX-COVER are solved correctly



vertex cover instance
(k = 2)

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

$S_a = \{ 3, 7 \}$

$S_b = \{ 2, 4 \}$

$S_c = \{ 3, 4, 5, 6 \}$

$S_d = \{ 5 \}$

$S_e = \{ 1 \}$

$S_f = \{ 1, 2, 6, 7 \}$

set cover instance
(k = 2)

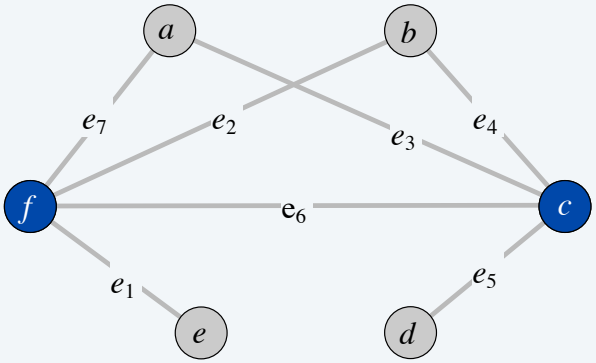
Vertex cover reduces to set cover

Lemma. $G = (V, E)$ contains a vertex cover of size k iff (U, S, k) contains a set cover of size k.

Pf. $\Leftarrow$ Let $Y \subseteq S$ be a set cover of size k in (U, S, k).

- Then $X = \{ v : S_v \in Y \}$ is a vertex cover of size k in G. ▪

"no" instances of VERTEX-COVER are solved correctly



vertex cover instance
(k = 2)

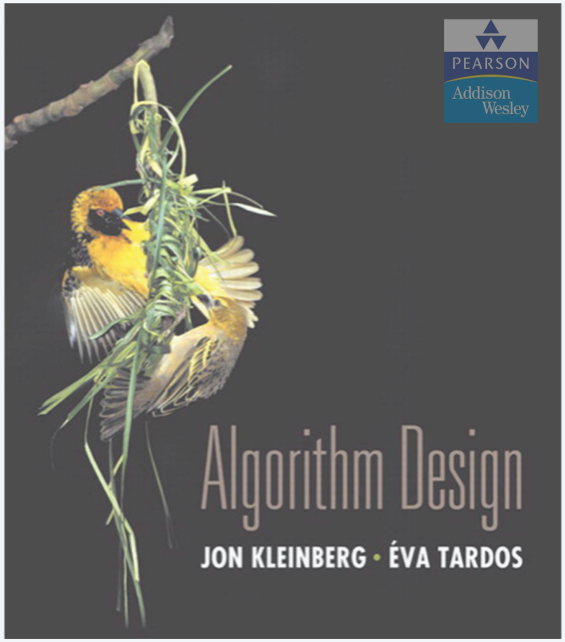
$U = \{ 1, 2, 3, 4, 5, 6, 7 \}$

$S_a = \{ 3, 7 \}$ $S_b = \{ 2, 4 \}$

$S_c = \{ 3, 4, 5, 6 \}$ $S_d = \{ 5 \}$

$S_e = \{ 1 \}$ $S_f = \{ 1, 2, 6, 7 \}$

set cover instance
(k = 2)



SECTION 8.2

REDUCTIONS, P AND NP

- *poly-time reductions*
- *packing and covering problems*
- *constraint satisfaction problems*
- *graph coloring*
- *P vs. NP*
- *NP-complete*

受限满足 / 的来补

Satisfiability

先年再交.

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 (CNF). A propositional formula Φ that is a conjunction of clauses. $\Phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$

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

3-SAT. SAT where each clause contains exactly 3 literals (and each literal corresponds to a different variable).

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

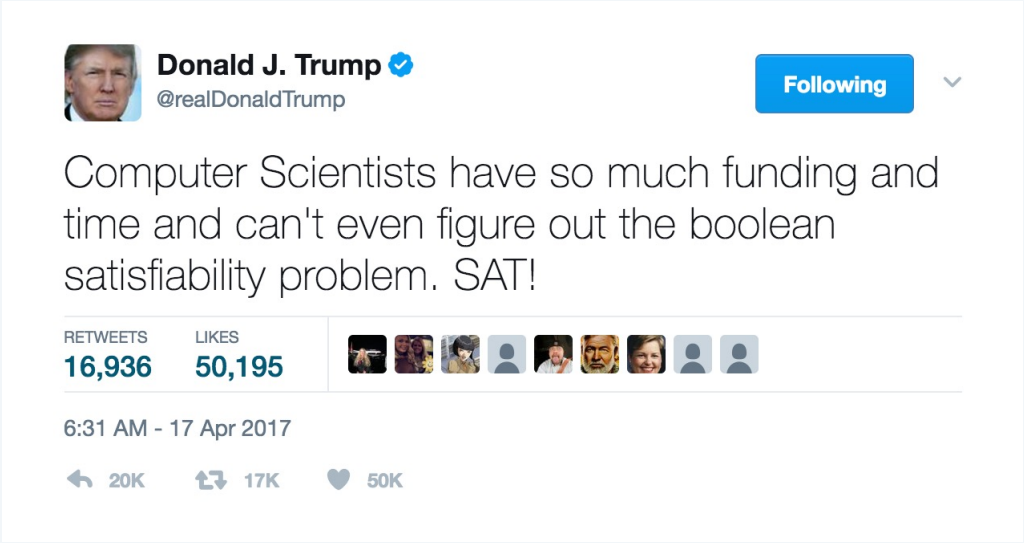
yes instance: x_1 = true, x_2 = true, x_3 = false, x_4 = false

Key application. Electronic design automation (EDA).

Satisfiability is hard

Scientific hypothesis. There does not exists a poly-time algorithm for 3-SAT.

P vs. NP. This hypothesis is equivalent to P ≠ NP conjecture.



https://www.facebook.com/pg/npcompleteteens

3-satisfiability reduces to independent set

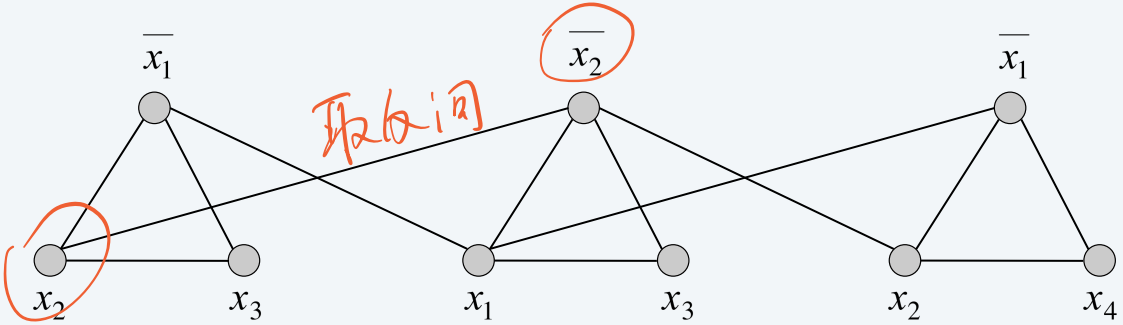
Theorem. 3-SAT $\leq_P$ INDEPENDENT-SET.

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

Construction.

- G contains 3 nodes 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

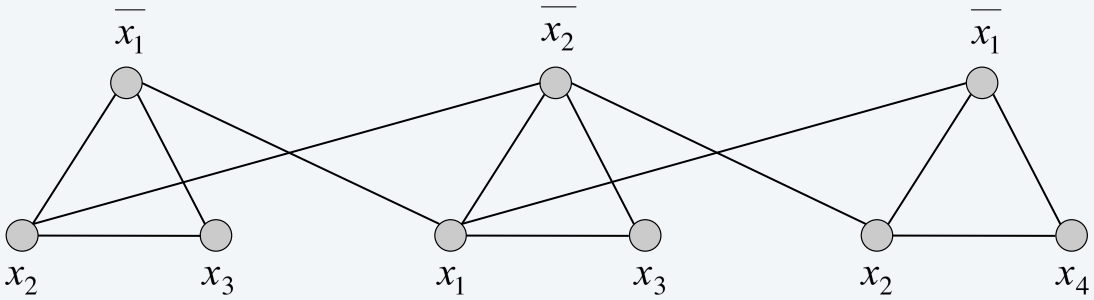
Lemma. Φ is satisfiable iff G contains an independent set of size $k = |\Phi|$.

Pf. $\Rightarrow$ Consider any satisfying assignment for Φ.

- Select one true literal from each clause/triangle.
- This is an independent set of size $k = |\Phi|$. ▪

"yes" instances of 3-SAT are solved correctly

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

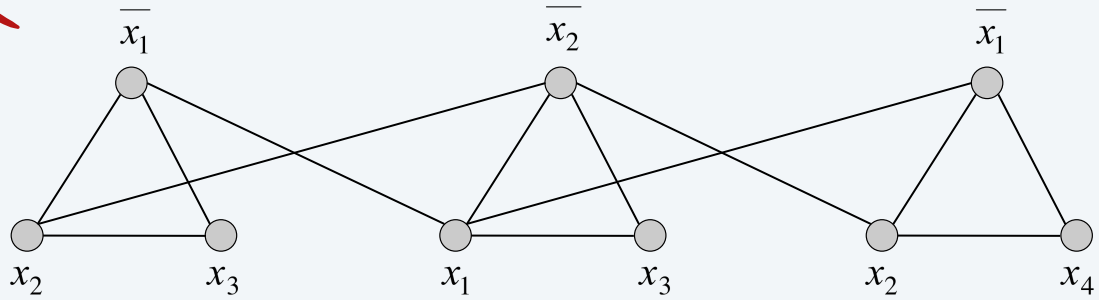
Lemma. Φ is satisfiable iff G contains an independent set of size $k = |\Phi|$.

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

- S must contain exactly one node in each triangle.
- Set these literals to *true* (and remaining literals consistently).
- All clauses in Φ are satisfied. ▪

"no" instances of 3-SAT are solved correctly

G



k = 3

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

Review

Basic reduction strategies.

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

Transitivity. If $X \leq_P Y$ and $Y \leq_P Z$, then $X \leq_P Z$.

Pf idea. Compose the two algorithms.

Ex. 3-SAT $\leq_P$ INDEPENDENT-SET $\leq_P$ VERTEX-COVER $\leq_P$ SET-COVER.

Decision, search, and optimization problems

Decision problem. Does there exist a vertex cover of size $\leq k$?

Search problem. Find a vertex cover of size $\leq k$.

Optimization problem. Find a vertex cover of minimum size.

Goal. Show that all three problems poly-time reduce to one another.

general

SEARCH PROBLEMS VS. DECISION PROBLEMS

VERTEX-COVER. Does there exist a vertex cover of size $\leq k$?

FIND-VERTEX-COVER. Find a vertex cover of size $\leq k$.

Exist

Theorem. VERTEX-COVER $\equiv_P$ FIND-VERTEX-COVER.

Pf. $\leq_P$ Decision problem is a special case of search problem. ▪

Pf. $\geq_P$

Find

To find a vertex cover of size $\leq k$:

- Determine if there exists a vertex cover of size $\leq k$. (判断exist)
- Find a vertex v such that $G - \{ v \}$ has a vertex cover of size $\leq k - 1$.
  (any vertex in any vertex cover of size $\leq k$ will have this property)
- Include v in the vertex cover.
- Recursively find a vertex cover of size $\leq k - 1$ in $G - \{ v \}$. ▪

delete v and all incident edges

Optimization problems vs. search problems

Find-Vertex-Cover. Find a vertex cover of size $\leq k$.

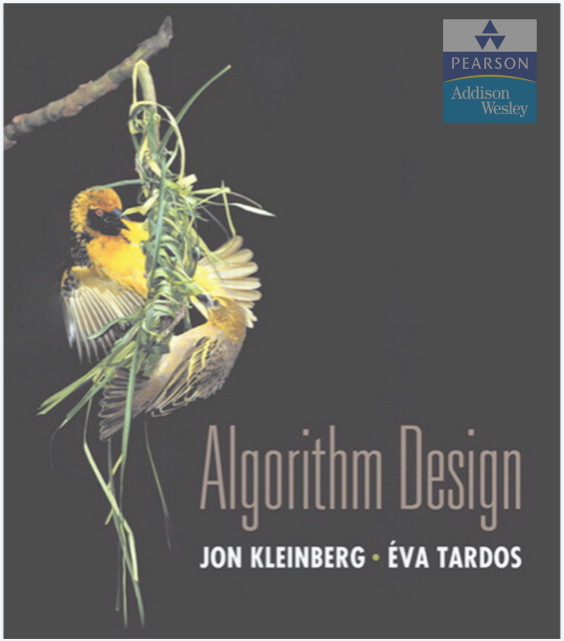
Find-Min-Vertex-Cover. Find a vertex cover of minimum size.

Theorem. Find-Vertex-Cover $\equiv_P$ Find-Min-Vertex-Cover.

Pf. $\leq_P$ Search problem is a special case of optimization problem. ▪

Pf. $\geq_P$ To find vertex cover of minimum size:

- Binary search (or linear search) for size k^* of min vertex cover.
- Solve search problem for given k^*. ▪



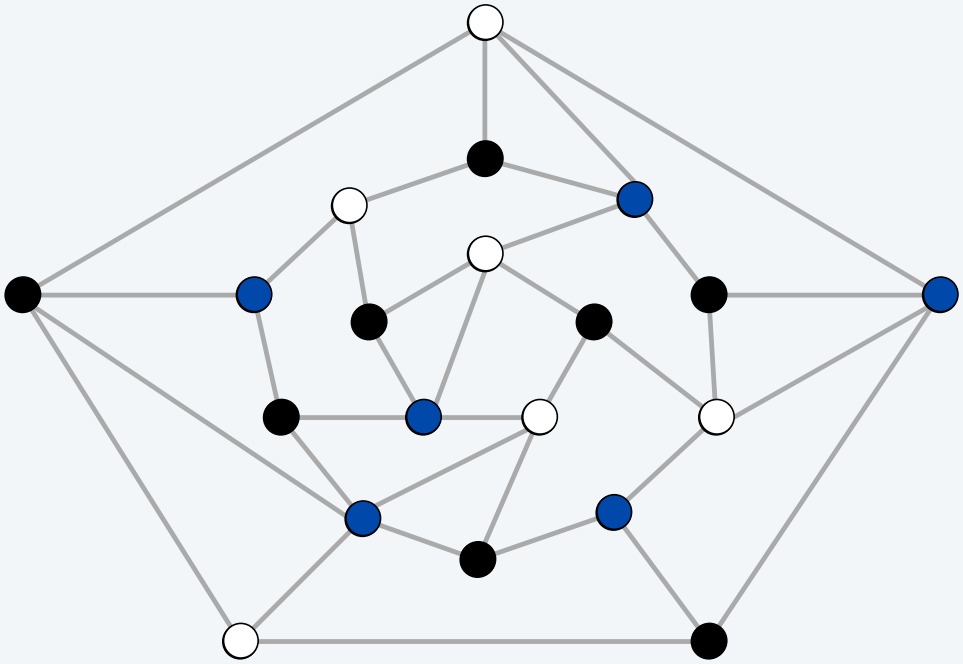
SECTION 8.7

REDUCTIONS, P AND NP

- *poly-time reductions*
- *packing and covering problems*
- *constraint satisfaction problems*
- *graph coloring*
- *P vs. NP*
- *NP-complete*

3-colorability

3-COLOR. Given an undirected graph G, can the nodes be colored black, white, and blue so that no adjacent nodes have the same color?



yes instance

Intractability: quiz 4

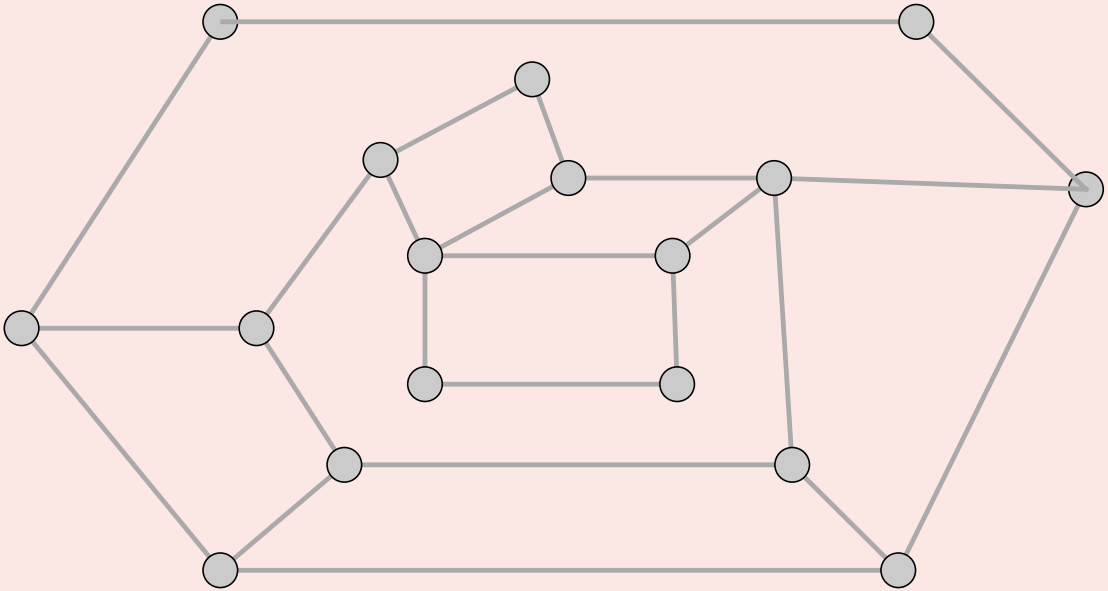
How difficult to solve 2-COLOR?

A. $O(m + n)$ using BFS or DFS. ✓

B. $O(mn)$ using maximum flow.

C. $\Omega(2^n)$ using brute force.

D. Not even Tarjan knows.



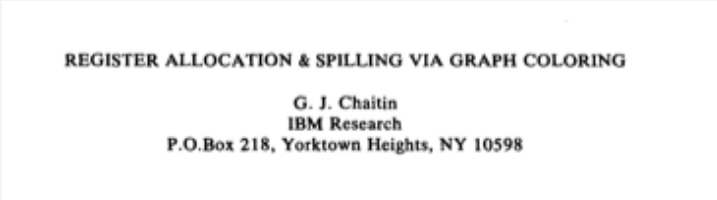
Application: register allocation

Register allocation. Assign program variables to machine registers so that no more than k registers are used and no two program variables that are needed at the same time are assigned to the same register.

Interference graph. Nodes are program variables; edge between u and v if there exists an operation where both u and v are "live" at the same time.

Observation. [Chaitin 1982] Can solve register allocation problem iff interference graph is k-colorable.

Fact. 3-COLOR $\leq_P$ K-REGISTER-ALLOCATION for any constant $k \geq 3$.



REGISTER ALLOCATION & SPILLING VIA GRAPH COLORING

G. J. Chaitin
IBM Research
P.O.Box 218, Yorktown Heights, NY 10598

3-satisfiability reduces to 3-colorability

Theorem. 3-SAT $\leq_P$ 3-COLOR.

Pf. Given 3-SAT instance Φ, we construct an instance of 3-COLOR that is 3-colorable iff Φ is satisfiable.

Equal to.

How to prove 3SAT

3-satisfiability reduces to 3-colorability

Construction.

(i) Create a graph G with a node for each literal.

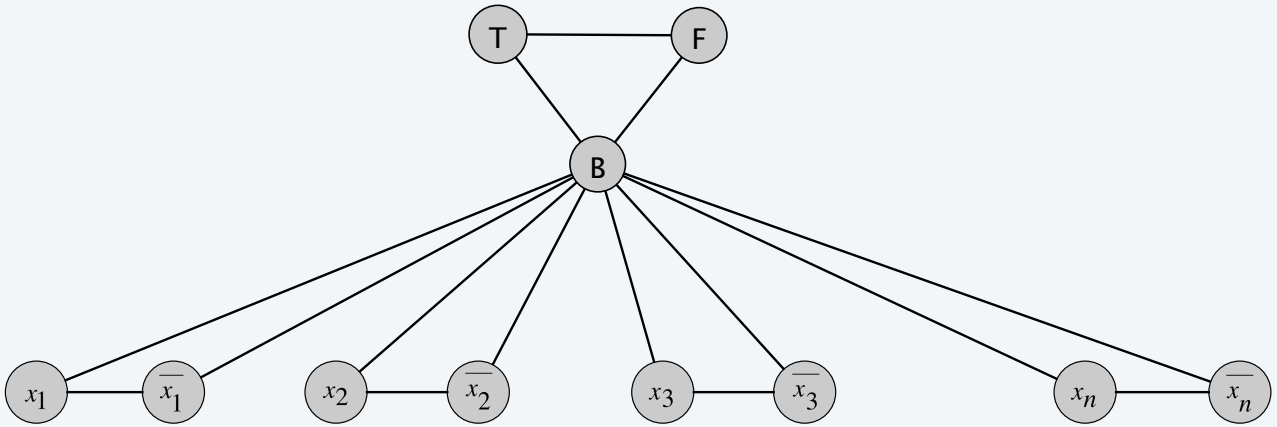
(ii) Connect each literal to its negation.

(iii) Create 3 new nodes T, F, and B; connect them in a triangle.

(iv) Connect each literal to B.

(v) For each clause C_j, add a gadget of 6 nodes and 13 edges.

to be described later

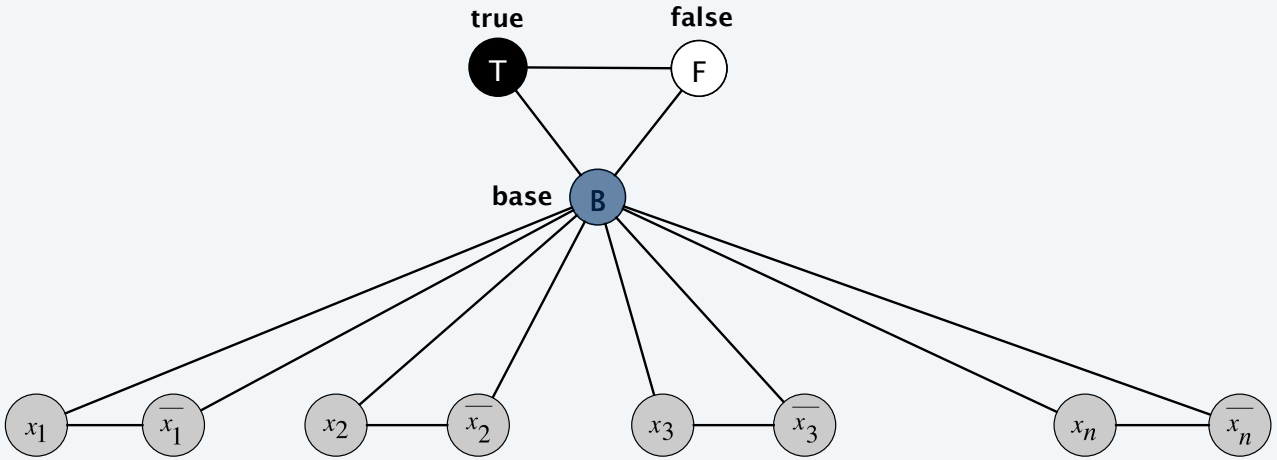


3-satisfiability reduces to 3-colorability

Lemma. Graph G is 3-colorable iff Φ is satisfiable.

Pf. $\Rightarrow$ Suppose graph G is 3-colorable.

- WLOG, assume that node T is colored *black*, F is *white*, and B is *blue*.
- Consider assignment that sets all *black* literals to *true* (and *white* to *false*).
- (iv) ensures each literal is colored either *black* or *white*.
- (ii) ensures that each literal is *white* if its negation is *black* (and vice versa).

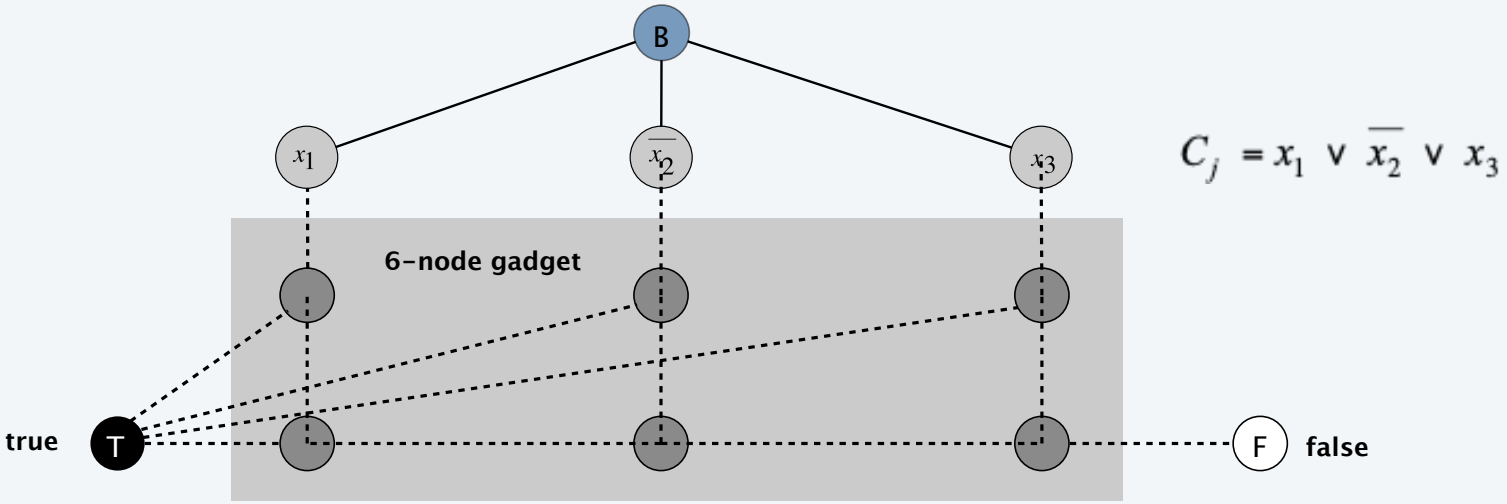


3-satisfiability reduces to 3-colorability

Lemma. Graph G is 3-colorable iff Φ is satisfiable.

Pf. $\Rightarrow$ Suppose graph G is 3-colorable.

- WLOG, assume that node T is colored *black*, F is *white*, and B is *blue*.
- Consider assignment that sets all *black* literals to *true* (and *white* to *false*).
- (iv) ensures each literal is colored either *black* or *white*.
- (ii) ensures that each literal is *white* if its negation is *black* (and vice versa).
- (v) ensures at least one literal in each clause is *black*.

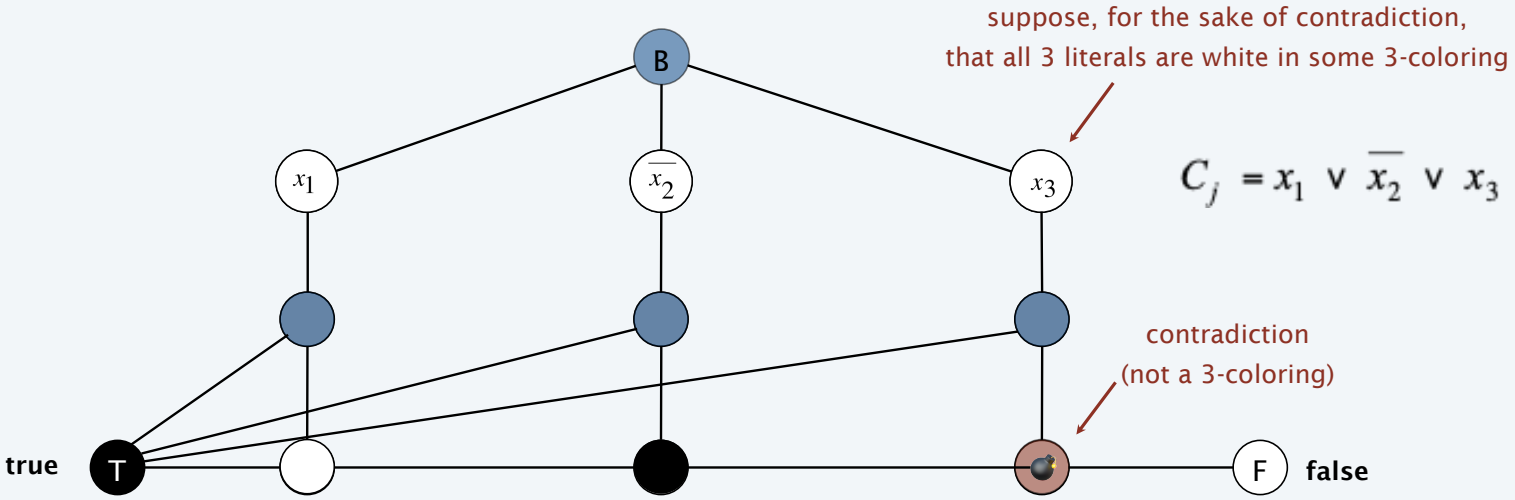


3-satisfiability reduces to 3-colorability

Lemma. Graph G is 3-colorable iff Φ is satisfiable.

Pf. $\Rightarrow$ Suppose graph G is 3-colorable.

- WLOG, assume that node T is colored *black*, F is *white*, and B is *blue*.
- Consider assignment that sets all *black* literals to *true* (and *white* to *false*).
- (iv) ensures each literal is colored either *black* or *white*.
- (ii) ensures that each literal is *white* if its negation is *black* (and vice versa).
- (v) ensures at least one literal in each clause is *black*. ▪

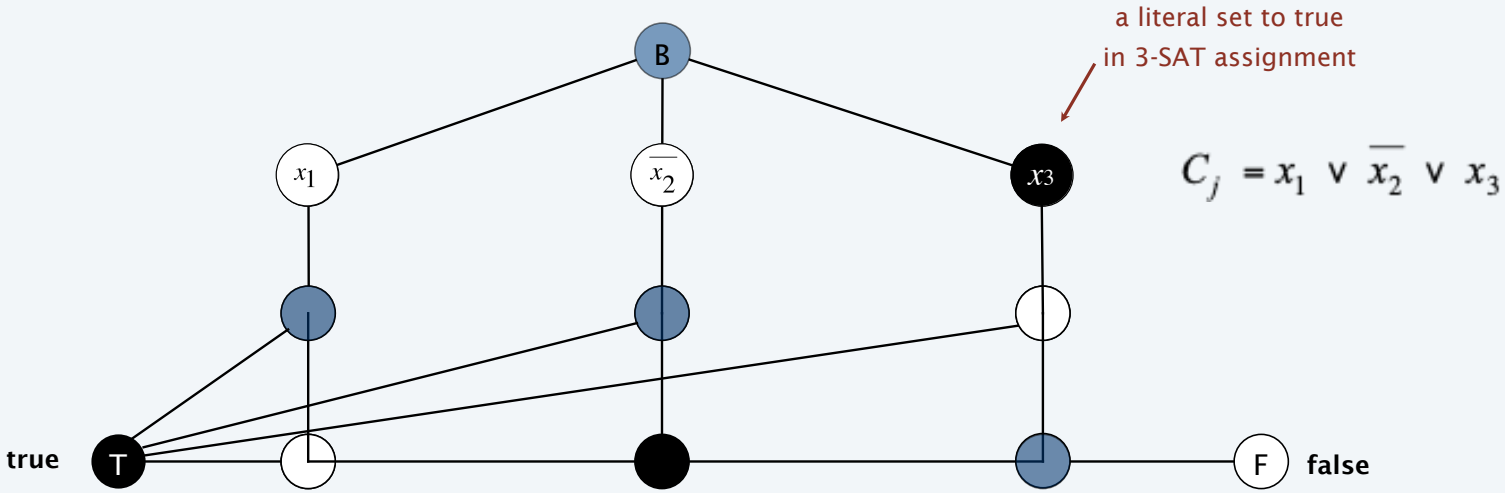


3-satisfiability reduces to 3-colorability

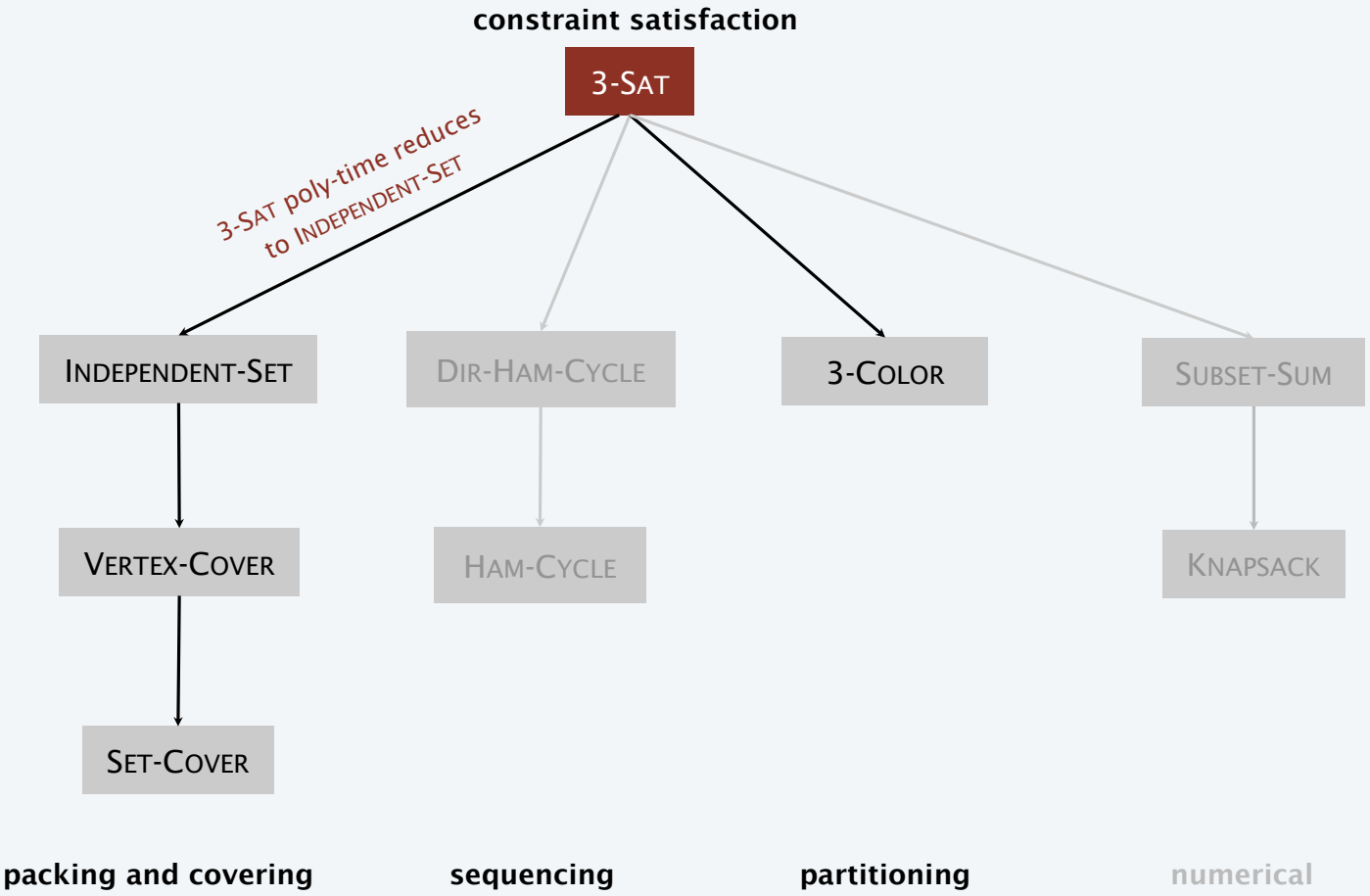
Lemma. Graph G is 3-colorable iff Φ is satisfiable.

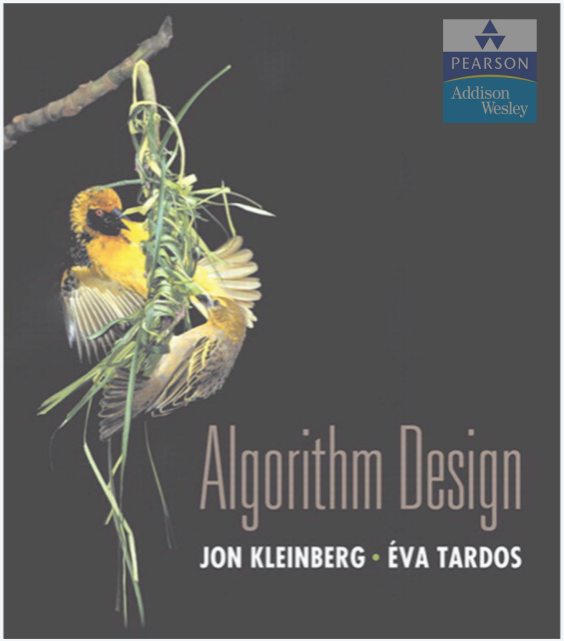
Pf. $\Leftarrow$ Suppose 3-SAT instance Φ is satisfiable.

- Color all *true* literals *black* and all *false* literals *white*.
- Pick one *true* literal; color node below that node *white*, and node below that *blue*.
- Color remaining middle row nodes *blue*.
- Color remaining bottom nodes *black* or *white*, as forced. ▪



Poly-time reductions





SECTION 8.3

REDUCTIONS, P AND NP

- *poly-time reductions*
- *packing and covering problems*
- *constraint satisfaction problems*
- *graph coloring*
- *P vs. NP*
- *NP-complete*

P

Decision problem.

- Problem X is a set of strings.
- Instance s is one string.
- Algorithm A solves problem X: $A(s) = \begin{cases} yes & \text{if } s \in X \\ no & \text{if } s \notin X \end{cases}$

Def. Algorithm A runs in polynomial time if for every string s, $A(s)$ terminates in $\leq p(|s|)$ "steps," where $p(\cdot)$ is some polynomial function.

length of s

Def. **P** = set of decision problems for which there exists a poly-time algorithm.

on a deterministic Turing machine

problem PRIMES:	$\{ 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, \ldots \}$
instance s:	592335744548702854681
algorithm:	Agrawal–Kayal–Saxena (2002)

NP

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

Def. **NP** = set of decision problems for which there exists a poly-time certifier.

- $C(s, t)$ is a poly-time algorithm.
- Certificate t is of polynomial size: $|t| \leq p(|s|)$ for some polynomial $p(\cdot)$.

"certificate" or "witness"

problem COMPOSITES:	{ 4, 6, 8, 9, 10, 12, 14, 15, 16, 18, 20, …. }
instance s:	437669
certificate t:	541 ← 437,669 = 541 × 809
certifier C(s, t):	grade-school division

Definition 2. *A decision problem is in the complexity class* P *(polynomial time) if there exists a polynomial time algorithm deciding (answering) it.*

It turns out that many interesting problems have not (yet) been proved to be in P. Most of these, however, are in a different class called NP.

Definition 3. *A decision problem is in the complexity class* NP *(nondeterministic polynomial time) it has a polynomial time verifier. A* ***verifier*** *for a decision problem takes a (decision problem) input S, an answer (yes or no), some additional information C commonly called a certificate or proof or witness.*

- *When the true answer is "yes", there must exist some certificate for which the verifier can prove the answer is "yes". If given the wrong certificate, the verifier is allowed to answer "no" incorrectly.*
- *When the true answer is "no", there must be no certificate for which the verifier proves (incorrectly) the answer is "yes".*

Intuitively, problems in NP are ones whose solutions are naturally easy to check. Often, the definition is phrased as a game between a *prover* and a *verifier*, where the omniscient prover wants to convince the

computationally limited verifier that its answer to the decision problem is correct. The prover hands the verifier the input, the claimed solution, and the certificate.

Example 1. Consider the boolean satisfiability (SAT) problem.

The input to SAT is a boolean formula of conjunctions (AND) between disjunctive (OR) clauses.

$$(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_2 \vee x_3) \wedge \cdots$$

The decision problem for SAT is whether a given formula has a satifying assignment. A polynomial time verifier for this problem takes as a certificate a satisfying assignment, and checks satisfiability clause by clause. On the other hand, if the formula is unsatisfiable, no assignment will pass this verification scheme. We conclude that SAT has a polynomial time verifier, and is therefore a problem in NP. Many common verifiers for search/optimization problems do indeed use the search "solution" as the certificate.

Interestingly, there are also problems not known to be in NP.

Example 2. Let NON-SAT be the problem of boolean unsatisfiability. That is, given a boolean formula, report whether the formula has no satisfying assignment. In other words, the set of formulas in NON-SAT is the *complement* of SAT. It is not known if NON-SAT is in NP.

1. *The problem itself is in NP class.*
2. *All other problems in NP class can be polynomial-time reducible to that.*
   (B is polynomial-time reducible to C is denoted as $\leq P^C$)

SAT is NPC

If the 2nd condition is only satisfied then the problem is called **NP-Hard**.

But it is not possible to reduce every NP problem into another NP problem to show its NP-Completeness all the time i.e., to show a problem is NP-complete then prove that the problem is in NP and any NP-Complete problem is reducible to that i.e. if B is NP-Complete and $B \leq P^C$ For C in NP, then C is NP-Complete. Thus, it can be verified that the **SAT Problem** is NP-Complete using the following propositions:

SAT is in NP:
It any problem is in NP, then given a 'certificate', which is a solution to the problem and an instance of the problem (a boolean formula ***f***) we will be able to check (identify if the solution is correct or not) certificate in polynomial time. This can be done by checking if the given assignment of variables satisfies the boolean formula.

SAT is NP-Hard:
In order to prove that this problem is NP-Hard then reduce a known problem, Circuit-SAT in this case to our problem. The boolean circuit **C** can be corrected into a boolean formula as:

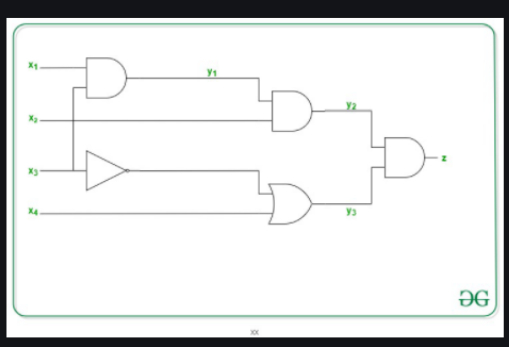
- For every input wire, add a new variable y_i.
- For every output wire, add a new variable **Z**.
- An equation is prepared for each gate.
- These sets of equations are separated by ∩ values and adding ∩**Z** at the end.

This transformation can be done in linear time. The following propositions now hold:

- If there is a set of input, variable values satisfying the circuit then it can derive an assignment for the formula ***f*** that satisfies the formula. This can be simulated by computing the output of every gate in the circuit.
- If there is a satisfying assignment for the formula ***f***, this can satisfy the boolean circuit after the removal of the newly added variables.

For Example: If below is the circuit then:

$y_1 = x_1 \cap x_3$
$y_2 = y_1 \cap x_2$
$y_3 = x_3 \cup x_4$
$f = y_1 \cap y_2 \cap y_3 \cap z$



Certifiers and certificates: satisfiability

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

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

Certificate. An assignment of truth values to the Boolean variables.

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

instance s	$\Phi = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$
certificate t	$x_1 = true,\ x_2 = true,\ x_3 = false,\ x_4 = false$

Conclusions. SAT ∈ **NP**, 3-SAT ∈ **NP**.

Intractability: quiz 5

Which of the following graph problems are known to be in NP?

A. Is the length of the longest simple path $\leq k$?

B. Is the length of the longest simple path $\geq k$?

C. Is the length of the longest simple path $= k$?

D. Find the length of the longest simple path.

E. All of the above.

Intractability: quiz 6

In complexity theory, the abbreviation NP stands for...

A. Nope.

B. No problem.

C. Not polynomial time.

D. Not polynomial space.

E. Nondeterministic polynomial time.

Significance of NP

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

> *" NP captures vast domains of computational, scientific, and mathematical endeavors, and seems to roughly delimit what mathematicians and scientists have been aspiring to compute feasibly. "* — *Christos Papadimitriou*

> *" In an ideal world it would be renamed P vs VP. "* — *Clyde Kruskal*

P, NP, and EXP

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

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

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

Proposition. **P** $\subseteq$ **NP**.

Pf. Consider any problem $X \in$ **P**.

- By definition, there exists a poly-time algorithm $A(s)$ that solves X.
- Certificate $t = \varepsilon$, certifier $C(s, t) = A(s)$. ■

Proposition. **NP** $\subseteq$ **EXP**.

Pf. Consider any problem $X \in$ **NP**.

- By definition, there exists a poly-time certifier $C(s, t)$ for X, where certificate t satisfies $|t| \leq p(|s|)$ for some polynomial $p(\cdot)$.
- To solve instance s, run $C(s, t)$ on all strings t with $|t| \leq p(|s|)$.
- Return *yes* iff $C(s, t)$ returns *yes* for any of these potential certificates. ■

Fact. **P** $\neq$ **EXP** $\Rightarrow$ either **P** $\neq$ **NP**, or **NP** $\neq$ **EXP**, or both.

The main question: P vs. NP

Q. How to solve an instance of 3-SAT with n variables?

A. Exhaustive search: try all 2^n truth assignments.

Q. Can we do anything substantially more clever?

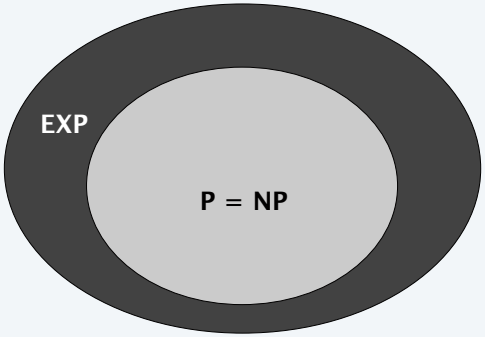
Conjecture. No poly-time algorithm for 3-SAT.

"intractable"

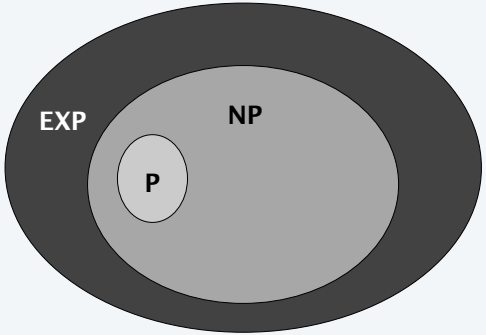
The main question: P vs. NP

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

Is the decision problem as easy as the certification problem?



If P = NP



If P ≠ NP

If yes… Efficient algorithms for 3-SAT, TSP, VERTEX-COVER, FACTOR, …

If no… No efficient algorithms possible for 3-SAT, TSP, VERTEX-COVER, …

Consensus opinion. Probably no.

Possible outcomes

P ≠ NP

" I conjecture that there is no good algorithm for the traveling salesman problem. My reasons are the same as for any mathematical conjecture: (i) It is a legitimate mathematical possibility and (ii) I do not know."

— *Jack Edmonds 1966*

" In my view, there is no way to even make intelligent guesses about the answer to any of these questions. If I had to bet now, I would bet that P is not equal to NP. I estimate the half-life of this problem at 25–50 more years, but I wouldn't bet on it being solved before 2100. "

— *Bob Tarjan (2002)*

Possible outcomes

P ≠ NP

" We seem to be missing even the most basic understanding of the nature of its difficulty…. All approaches tried so far probably (in some cases, provably) have failed. In this sense P =NP is different from many other major mathematical problems on which a gradual progress was being constantly done (sometimes for centuries) whereupon they yielded, either completely or partially. "

— *Alexander Razborov (2002)*

Possible outcomes

P = NP

> *" I think that in this respect I am on the loony fringe of the mathematical community: I think (not too strongly!) that P=NP and this will be proved within twenty years. Some years ago, Charles Read and I worked on it quite bit, and we even had a celebratory dinner in a good restaurant before we found an absolutely fatal mistake. "*
>
> — *Béla Bollobás (2002)*

> *" In my opinion this shouldn't really be a hard problem; it's just that we came late to this theory, and haven't yet developed any techniques for proving computations to be hard. Eventually, it will just be a footnote in the books. "* — *John Conway*

Other possible outcomes

P = **NP**, but only $\Omega(n^{100})$ algorithm for 3-SAT.

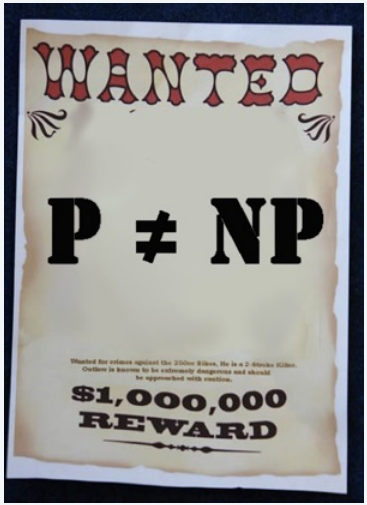
P ≠ **NP**, but with $O(n^{\log^* n})$ algorithm for 3-SAT.

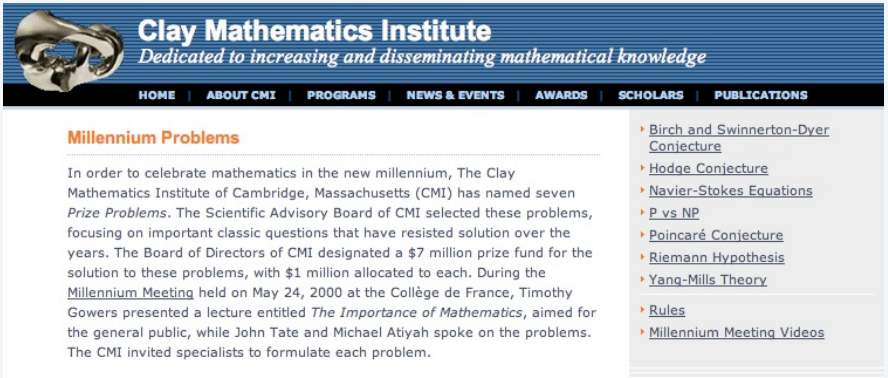
P = **NP** is independent (of ZFC axiomatic set theory).

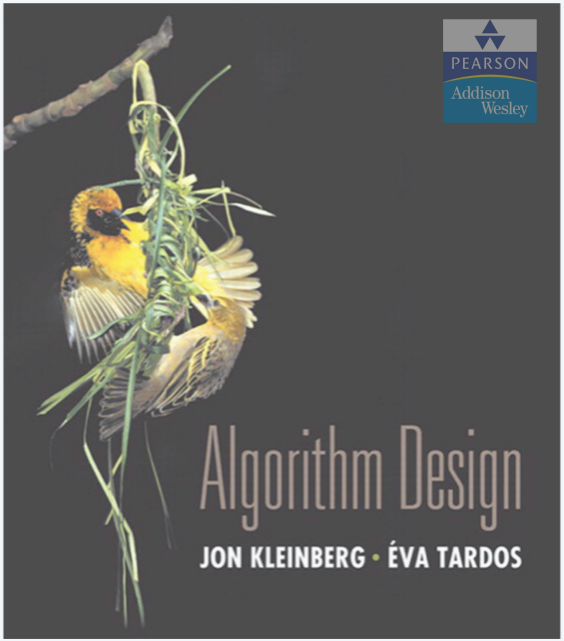
" It will be solved by either 2048 or 4096. I am currently somewhat pessimistic. The outcome will be the truly worst case scenario: namely that someone will prove P = NP because there are only finitely many obstructions to the opposite hypothesis; hence there exists a polynomial time solution to SAT but we will never know its complexity! " — *Donald Knuth*

Millennium prize

Millennium prize. $1 million for resolution of **P ≠ NP** problem.







SECTION 8.4

REDUCTIONS, P AND NP

- *poly-time reductions*
- *packing and covering problems*
- *constraint satisfaction problems*
- *graph coloring*
- *P vs. NP*
- *NP-complete*

Polynomial transformations

Def. Problem X polynomial (Cook) 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.

Def. Problem X polynomial (Karp) transforms to problem Y if given any instance x of X, we can construct an instance y of Y such that x is a *yes* instance of X iff y is a *yes* instance of Y.

we require $|y|$ to be of size polynomial in $|x|$

Note. Polynomial transformation is polynomial reduction with just one call to oracle for Y, exactly at the end of the algorithm for X. Almost all previous reductions were of this form.

Open question. Are these two concepts the same with respect to **NP**?

we abuse notation $\leq_P$ and blur distinction

NP-complete

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

solvable in poly time.

Proposition. Suppose $Y \in$ **NP**-complete. Then $Y \in$ **P** iff **P** = **NP**.

Pf. $\Leftarrow$ If **P** = **NP**, then $Y \in$ **P** because $Y \in$ **NP**.

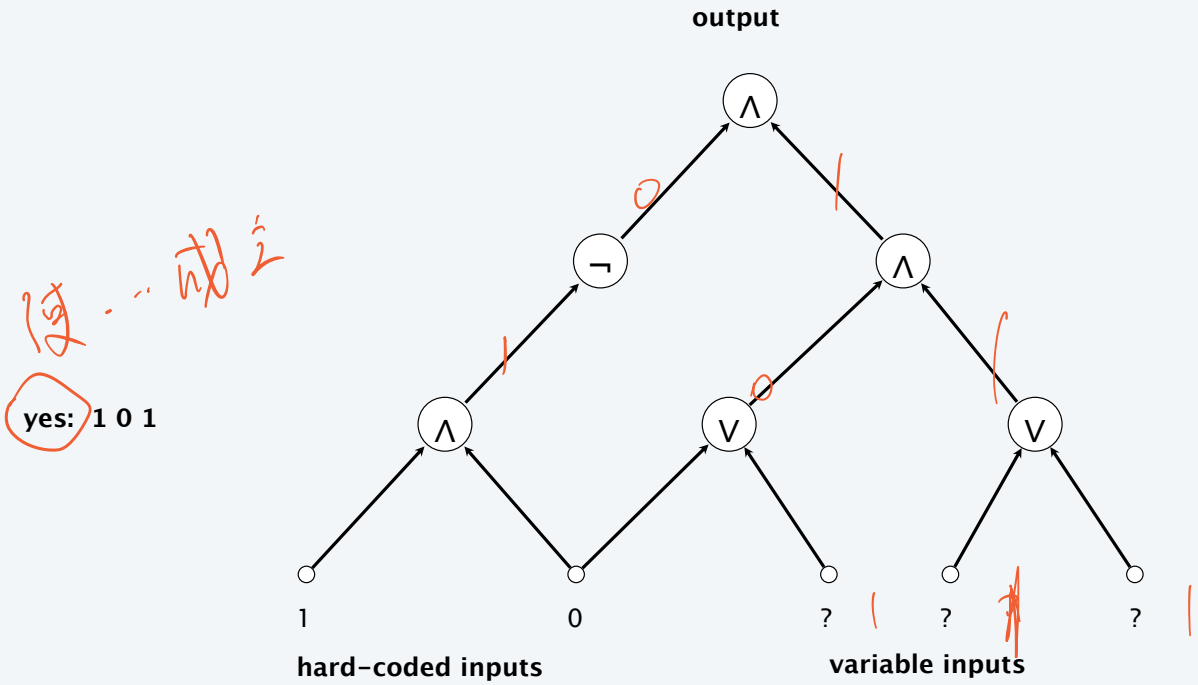
Pf. $\Rightarrow$ Suppose $Y \in$ **P**.

- Consider any problem $X \in$ **NP**. Since $X \leq_P Y$, we have $X \in$ **P**.
- This implies **NP** $\subseteq$ **P**.
- We already know **P** $\subseteq$ **NP**. Thus **P** = **NP**. ■

Fundamental question. Are there any "natural" **NP**-complete problems?

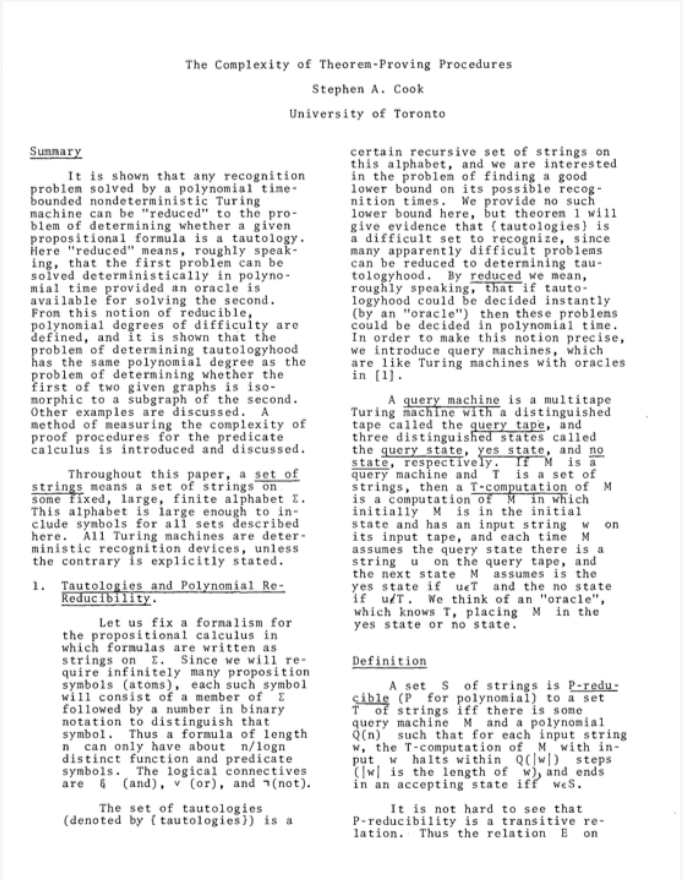
Circuit satisfiability

CIRCUIT-SAT. Given a combinational circuit built from AND, OR, and NOT gates, is there a way to set the circuit inputs so that the output is 1?



The “first” NP-complete problem

Theorem. CIRCUIT-SAT ∈ **NP**-complete. [Cook 1971, Levin 1973]



The Complexity of Theorem-Proving Procedures

Stephen A. Cook

University of Toronto

Summary

It is shown that any recognition problem solved by a polynomial time-bounded nondeterministic Turing machine can be "reduced" to the problem of determining whether a given propositional formula is a tautology. Here "reduced" means, roughly speaking, that the first problem can be solved deterministically in polynomial time provided an oracle is available for solving the second. From this notion of reducible, polynomial degrees of difficulty are defined, and it is shown that the problem of determining tautologyhood has the same polynomial degree as the problem of determining whether the first of two given graphs is isomorphic to a subgraph of the second. Other examples are discussed. A method of measuring the complexity of proof procedures for the predicate calculus is introduced and discussed.

Throughout this paper, a set of strings means a set of strings on some fixed, large, finite alphabet Σ. This alphabet is large enough to include symbols for all sets described here. All Turing machines are deterministic recognition devices, unless the contrary is explicitly stated.

1. Tautologies and Polynomial Reducibility.

Let us fix a formalism for the propositional calculus in which formulas are written as strings on Σ. Since we will require infinitely many proposition symbols (atoms), each such symbol will consist of a member of Σ followed by a number in binary notation to distinguish that symbol. Thus a formula of length n can only have about n/log n distinct function and predicate symbols. The logical connectives are & (and), ∨ (or), and ¬(not).

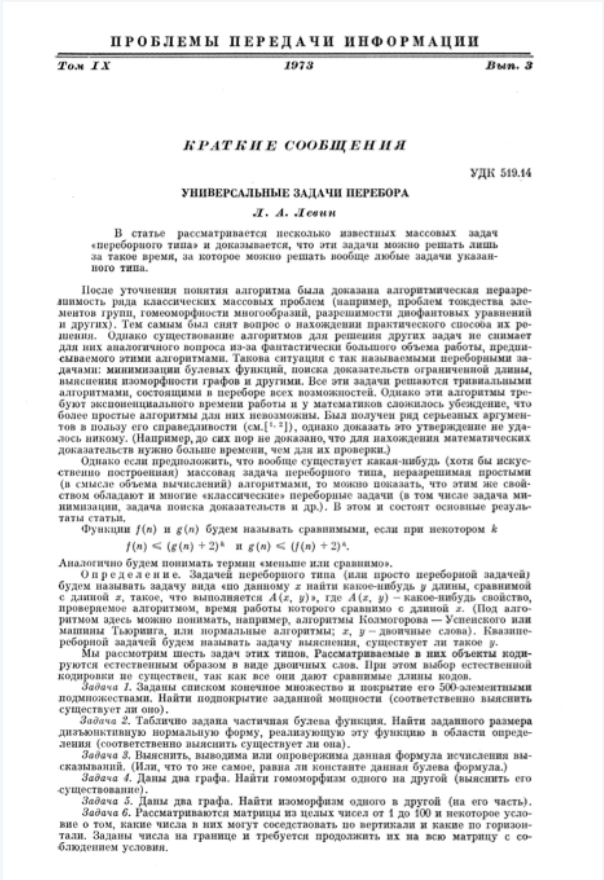
The set of tautologies (denoted by {tautologies}) is a certain recursive set of strings on this alphabet, and we are interested in the problem of finding a good lower bound on its possible recognition times. We provide no such lower bound here, but theorem 1 will give evidence that {tautologies} is a difficult set to recognize, since many apparently difficult problems can be reduced to determining tautologyhood. By reduced we mean, roughly speaking, that if tautologyhood could be decided instantly (by an "oracle") then these problems could be decided in polynomial time. In order to make this notion precise, we introduce query machines, which are like Turing machines with oracles in [1].

A query machine is a multitape Turing machine with a distinguished tape called the query tape, and three distinguished states called the query state, yes state, and no state, respectively. If M is a query machine and T is a set of strings, then a T-computation of M is a computation of M in which initially M is in the initial state and has an input string w on its input tape, and each time M assumes the query state there is a string u on the query tape, and the next state M assumes is the yes state if u∈T and the no state if u∉T. We think of an "oracle", which knows T, placing M in the yes state or no state.

Definition

A set S of strings is P-reducible (P for polynomial) to a set T of strings iff there is some query machine M and a polynomial Q(n) such that for each input string w, the T-computation of M with input w halts within Q(|w|) steps (|w| is the length of w), and ends in an accepting state iff w∈S.

It is not hard to see that P-reducibility is a transitive relation. Thus the relation E on



ПРОБЛЕМЫ ПЕРЕДАЧИ ИНФОРМАЦИИ

Том IX *1973* *Вып. 3*

КРАТКИЕ СООБЩЕНИЯ

УДК 519.14

УНИВЕРСАЛЬНЫЕ ЗАДАЧИ ПЕРЕБОРА

Л. А. Левин

В статье рассматривается несколько известных массовых задач «переборного типа» и доказывается, что эти задачи можно решать лишь за такое время, за которое можно решать вообще любые задачи указанного типа.

После уточнения понятия алгоритма была доказана алгоритмическая неразрешимость ряда классических массовых проблем (например, проблем тождества элементов групп, гомеоморфности многообразий, разрешимости диофантовых уравнений и других). Тем самым был снят вопрос о нахождении практического способа их решения. Однако существование алгоритмов для решения других задач не снимает для них аналогичного вопроса из-за фантастически большого объема работы, предписываемого этими алгоритмами. Такова ситуация с так называемыми переборными задачами: минимизации булевых функций, поиска доказательств ограниченной длины, выяснения изоморфности графов и другими. Все эти задачи решаются тривиальными алгоритмами, состоящими в переборе всех возможностей. Однако эти алгоритмы требуют экспоненциального времени работы и у математиков сложилось убеждение, что более простые алгоритмы для них невозможны. Был получен ряд серьезных аргументов в пользу его справедливости (см.[1, 2]), однако доказать это утверждение не удалось никому. (Например, до сих пор не доказано, что для нахождения математических доказательств нужно больше времени, чем для их проверки.)

Однако если предположить, что вообще существует какая-нибудь (хотя бы искусственно построенная) массовая задача переборного типа, неразрешимая простыми (в смысле объема вычислений) алгоритмами, то можно показать, что этим же свойством обладают и многие «классические» переборные задачи (в том числе задача минимизации, задача поиска доказательств и др.). В этом и состоит основные результаты статьи.

Функции $f(n)$ и $g(n)$ будем называть сравнимыми, если при некотором k

$$f(n) \leqslant (g(n)+2)^k \quad \text{и} \quad g(n) \leqslant (f(n)+2)^k.$$

Аналогично будем понимать термин «меньше или сравнимо».

Определение. Задачей переборного типа (или просто переборной задачей) будем называть задачу вида «по данному x найти какое-нибудь y длины, сравнимой с длиной x, такое, что выполняется $A(x, y)$», где $A(x, y)$ – какое-нибудь свойство, проверяемое алгоритмом, время работы которого сравнимо с длиной x. (Под алгоритмом здесь можно понимать, например, алгоритмы Колмогорова — Успенского или машины Тьюринга, или нормальные алгоритмы; x, y – двоичные слова). Квазиэкстремальной задачей будем называть задачу выяснения, существует ли такое y.

Мы рассмотрим шесть задач этих типов. Рассматриваемые в них объекты кодируются естественным образом в виде двоичных слов. При этом выбор естественной кодировки не существен, так как все они дают сравнимые длины кодов.

Задача 1. Заданы списком конечное множество и покрытие его 500-элементными подмножествами. Найти подпокрытие заданной мощности (соответственно выяснить существует ли оно).

Задача 2. Таблично задана частичная булева функция. Найти заданного размера дизъюнктивную нормальную форму, реализующую эту функцию в области определения (соответственно выяснить существует ли она).

Задача 3. Выяснить, выводима или опровержима данная формула исчисления высказываний. (Или, что то же самое, равна ли константе данная булева формула.)

Задача 4. Даны два графа. Найти гомоморфизм одного на другой (выяснить его существование).

Задача 5. Даны два графа. Найти изоморфизм одного в другой (на его часть).

Задача 6. Рассматриваются матрицы из целых чисел от 1 до 100 и некоторое условие о том, какие числа в них могут соседствовать по вертикали и какие по горизонтали. Заданы числа на границе и требуется продолжить их на всю матрицу с соблюдением условия.

The "first" NP-complete problem

Theorem. CIRCUIT-SAT ∈ **NP**-complete.

Pf sketch.

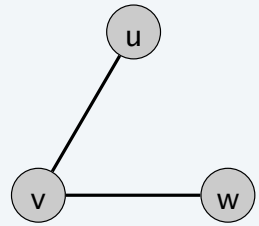
- Clearly, CIRCUIT-SAT ∈ **NP**.
- Any algorithm that takes a fixed number of bits n as input and produces a *yes* or *no* answer can be represented by such a circuit.
- Moreover, if algorithm takes poly-time, then circuit is of poly-size.

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

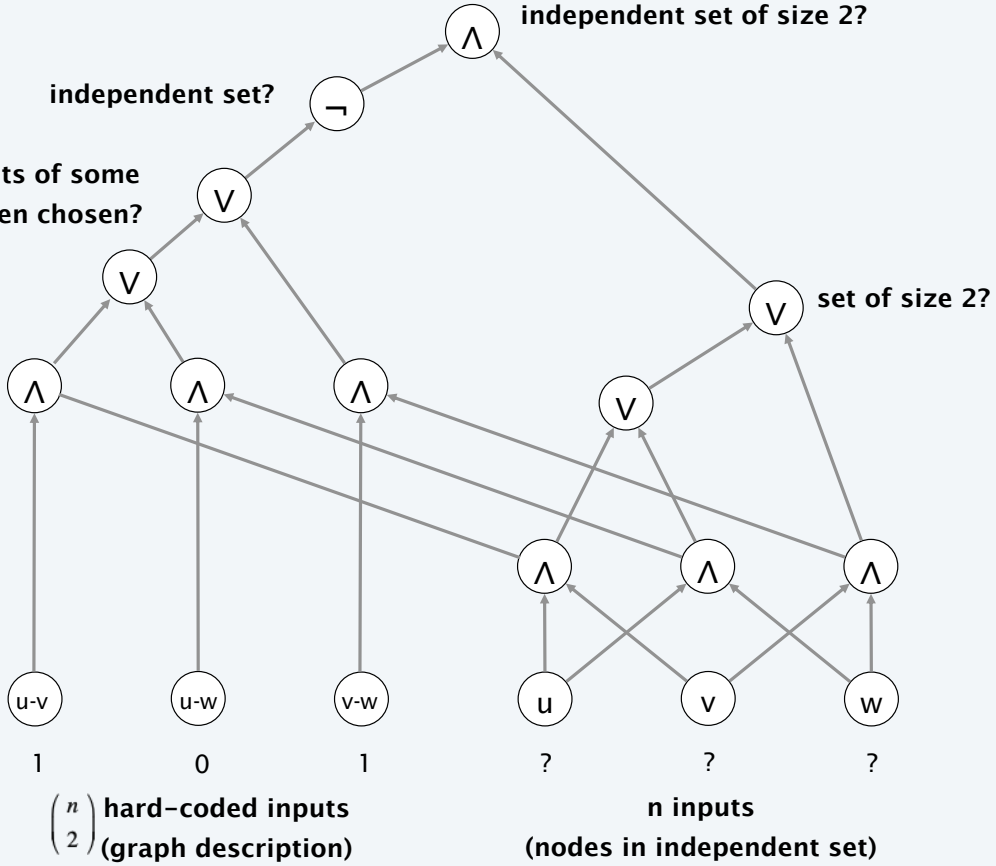
- Consider any problem $X \in$ **NP**. It has a poly-time certifier $C(s, t)$, where certificate t satisfies $|t| \leq p(|s|)$ for some polynomial $p(\cdot)$.
- View $C(s, t)$ as an algorithm with $|s| + p(|s|)$ input bits and convert it into a poly-size circuit K.
  - first $|s|$ bits are hard-coded with s
  - remaining $p(|s|)$ bits represent (unknown) bits of t
- Circuit K is satisfiable iff $C(s, t) = yes$.

Example

Ex. Construction below creates a circuit K whose inputs can be set so that it outputs 1 iff graph G has an independent set of size 2.



G = (V, E), n = 3



Establishing NP-completeness

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

Recipe. To prove that $Y \in$ **NP**-complete:

- Step 1. Show that $Y \in$ **NP**. certificate.
- Step 2. Choose an **NP**-complete problem X.
- Step 3. Prove that $X \leq_P Y$.

reduce X to Y

Proposition. If $X \in$ **NP**-complete, $Y \in$ **NP**, and $X \leq_P Y$, then $Y \in$ **NP**-complete.

Pf. Consider any problem $W \in$ **NP**. Then, both $W \leq_P X$ and $X \leq_P Y$.

- By transitivity, $W \leq_P Y$.
- Hence $Y \in$ **NP**-complete. ■

$W \leq_P X$: by definition of **NP**-complete

$X \leq_P Y$: by assumption

any

$W \to X \to Y$

Reductions: quiz 7

Suppose that $X \in$ NP-COMPLETE, $Y \in$ NP, and $X \leq_P Y$. Which can you infer?

A. Y is **NP**-complete.

B. If $Y \notin$ **P**, then **P** ≠ **NP**.

C. If **P** ≠ **NP**, then neither X nor Y is in **P**.

D. All of the above.

3-satisfiability is NP-complete

Theorem. 3-SAT $\in$ **NP**-complete.

Pf.

- Suffices to show that CIRCUIT-SAT $\leq_P$ 3-SAT since 3-SAT $\in$ **NP**.
- Given a combinational circuit K, we construct an instance Φ of 3-SAT that is satisfiable iff the inputs of K can be set so that it outputs 1.

3-satisfiability is NP-complete

Construction. Let K be any circuit.

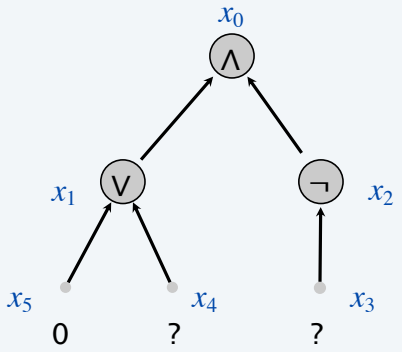
Step 1. Create a 3-SAT variable x_i for each circuit element i.

Step 2. Make circuit compute correct values at each node:

- $x_2 = \neg x_3$ ⇒ add 2 clauses: $x_2 \vee x_3$, $\overline{x_2} \vee \overline{x_3}$
- $x_1 = x_4 \vee x_5$ ⇒ add 3 clauses: $x_1 \vee \overline{x_4}$, $x_1 \vee \overline{x_5}$, $\overline{x_1} \vee x_4 \vee x_5$
- $x_0 = x_1 \wedge x_2$ ⇒ add 3 clauses: $\overline{x_0} \vee x_1$, $\overline{x_0} \vee x_2$, $x_0 \vee \overline{x_1} \vee \overline{x_2}$

Step 3. Hard-coded input values and output value.

- $x_5 = 0$ ⇒ add 1 clause: $\overline{x_5}$
- $x_0 = 1$ ⇒ add 1 clause: x_0



3-satisfiability is NP-complete

Construction. [continued]

Step 4. Turn clauses of length 1 or 2 into clauses of length 3.

- Create four new variables z_1, z_2, z_3, and z_4.
- Add 8 clauses to force $z_1 = z_2 = 0$:

$$(\overline{z_1} \vee z_3 \vee z_4),\ (\overline{z_1} \vee z_3 \vee \overline{z_4}),\ (\overline{z_1} \vee \overline{z_3} \vee z_4),\ (\overline{z_1} \vee \overline{z_3} \vee \overline{z_4})$$
$$(\overline{z_2} \vee z_3 \vee z_4),\ (\overline{z_2} \vee z_3 \vee \overline{z_4}),\ (\overline{z_2} \vee \overline{z_3} \vee z_4),\ (\overline{z_2} \vee \overline{z_3} \vee \overline{z_4})$$

- Replace any clause with a single term (t_i) with $(t_i \vee z_1 \vee z_2)$.
- Replace any clause with two terms $(t_i \vee t_j)$ with $(t_i \vee t_j \vee z_1)$.

3-satisfiability is NP-complete

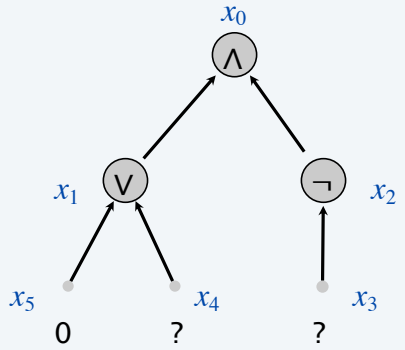
Lemma. Φ is satisfiable iff the inputs of K can be set so that it outputs 1.

Pf. $\Leftarrow$ Suppose there are inputs of K that make it output 1.

- Can propagate input values to create values at all nodes of K.
- This set of values satisfies Φ.

Pf. $\Rightarrow$ Suppose Φ is satisfiable.

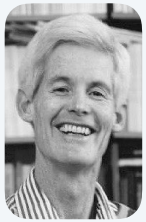
- We claim that the set of values corresponding to the circuit inputs constitutes a way to make circuit K output 1.
- The 3-SAT clauses were designed to ensure that the values assigned to all node in K exactly match what the circuit would compute for these nodes. ■



Implications of Karp

SAT poly-time reduces to all of these problems (and many, many more)

Implications of Cook–Levin



SAT

INDEPENDENT-SET poly-time reduces to CIRCUIT-SAT

3-SAT

INDEPENDENT-SET

DIR-HAM-CYCLE

3-COLOR

SUBSET-SUM

VERTEX-COVER

HAM-CYCLE

KNAPSACK

SET-COVER

All of these problems (and many, many more) poly-time reduce to SAT.

Implications of Karp + Cook–Levin

All of these problems are **NP**-complete; they are manifestations of the same really hard problem.