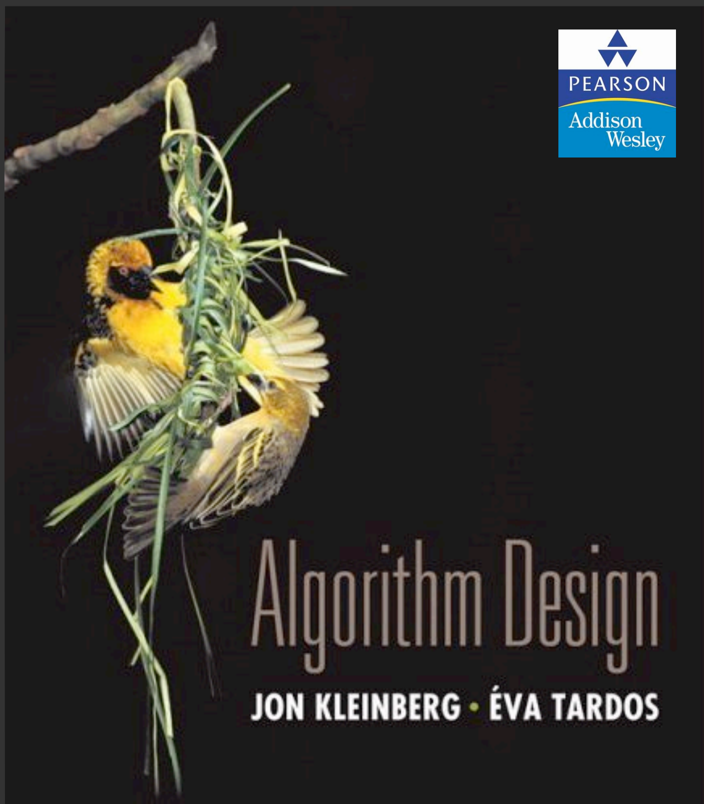


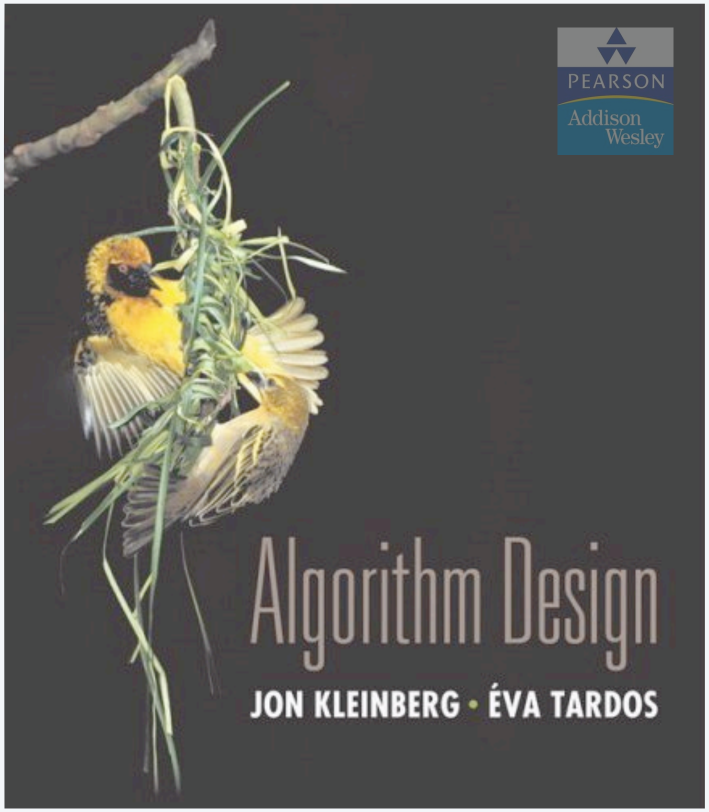

# 8. INTRACTABILITY I

- *poly-time reductions*
- *packing and covering problems*
- *constraint satisfaction problems*
- *sequencing problems*
- *partitioning problems*
- *graph coloring*
- *numerical problems*

Lecture slides by Kevin Wayne

http://www.cs.princeton.edu/~wayne/kleinberg-tardos

Last updated on Sep 8, 2013 6:40 AM




SECTION 8.1

# 8. INTRACTABILITY I

- *poly-time reductions*
- *packing and covering problems*
- *constraint satisfaction problems*
- *sequencing problems*
- *partitioning problems*
- *graph coloring*
- *numerical problems*

# Algorithm design patterns and antipatterns

**Algorithm design patterns.**

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

**Algorithm design antipatterns.**

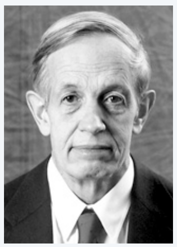
- NP-completeness. $O(n^k)$ algorithm unlikely.
- PSPACE-completeness. $O(n^k)$ certification algorithm unlikely.
- Undecidability. No algorithm possible.

# Classify problems according to computational requirements

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

A working definition. Those with polynomial-time algorithms.

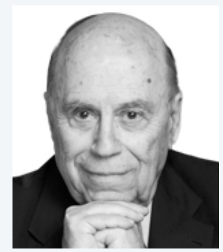

von Neumann (1953)


Nash (1955)


Gödel (1956)


Cobham (1964)


Edmonds (1965)


Rabin (1966)

Theory. Definition is broad and robust.

constants $a$ and $b$ tend to be small, e.g., $3\ N^2$

Practice. Poly-time algorithms scale to huge problems.

# Classify problems according to computational requirements

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

A working definition. Those with polynomial-time algorithms.

| yes | probably no |
| --- | --- |
| shortest path | longest path |
| min cut | max cut |
| 2-satisfiability | 3-satisfiability |
| planar 4-colorability | planar 3-colorability |
| bipartite vertex cover | vertex cover |
| matching | 3d-matching |
| primality testing | factoring |
| linear programming | integer linear programming |

# Classify problems

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

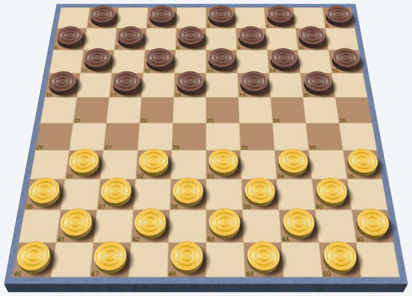
input size = c + lg k

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?

using forced capture rule


Alan designed the perfect computer



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

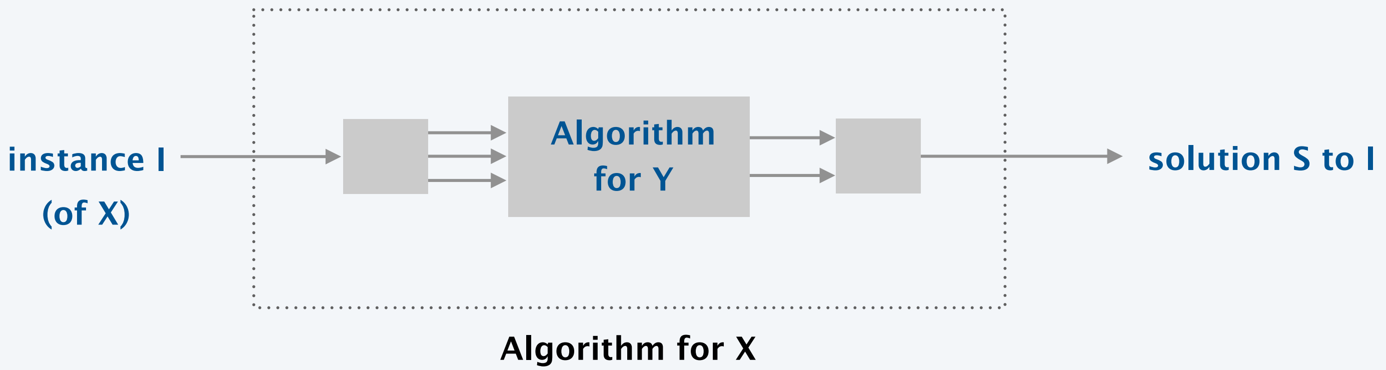
# Polynomial-time reductions

Desiderata'. Suppose we could solve $X$ 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



# Polynomial-time reductions

Desiderata'. Suppose we could solve $X$ 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. $X \le_P Y$.

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

Caveat. Don't mistake $X \le_P Y$ with $Y \le_P X$.

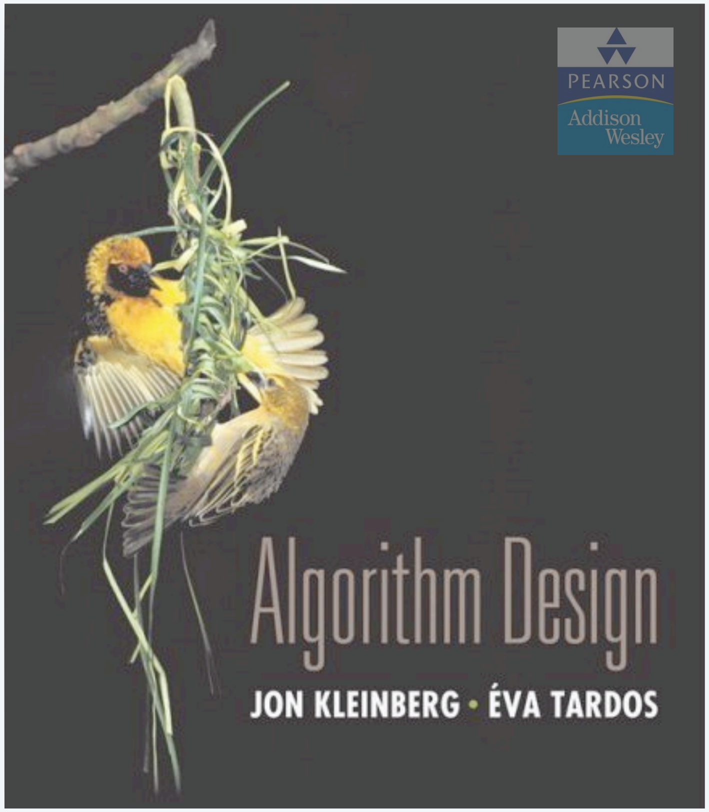
# Polynomial-time reductions

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

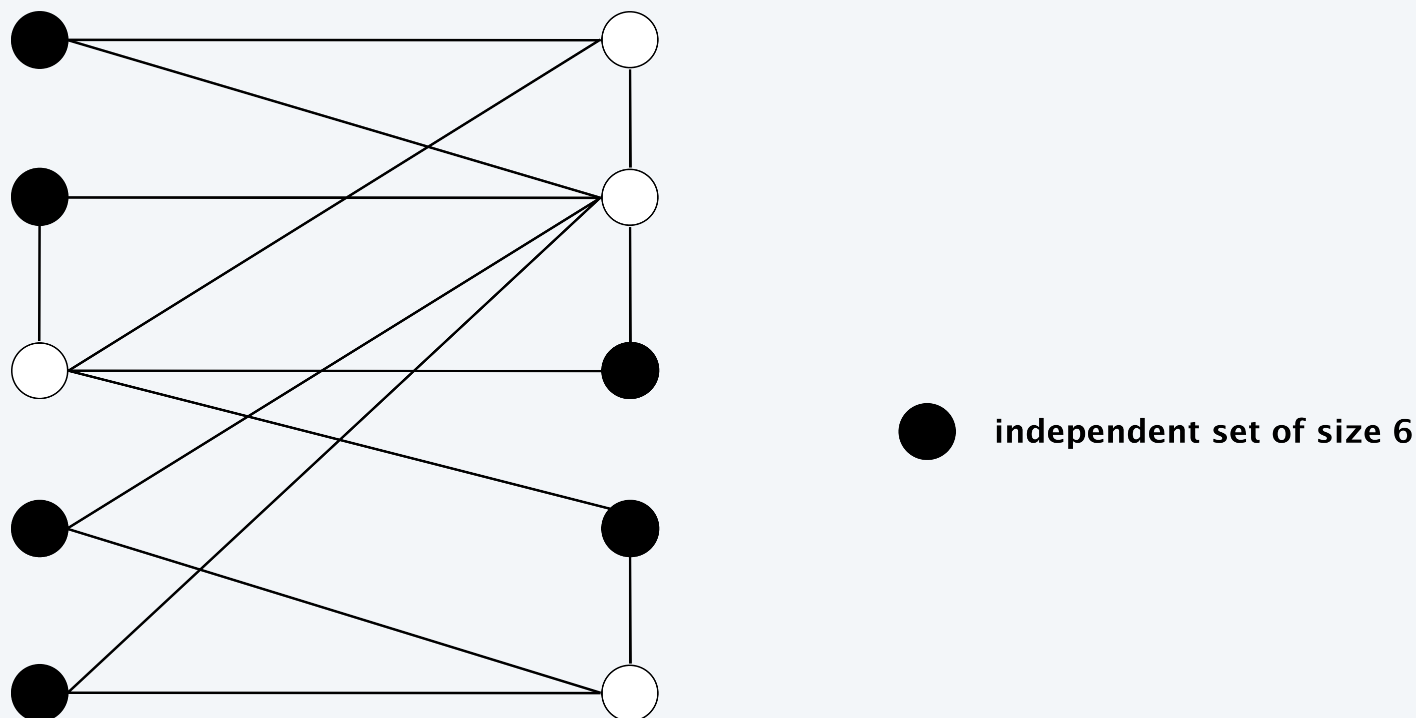
# 8. INTRACTABILITY I

- poly-time reductions
- *packing and covering problems*
- constraint satisfaction problems
- sequencing problems
- partitioning problems
- graph coloring
- numerical problems

# 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 $S$?

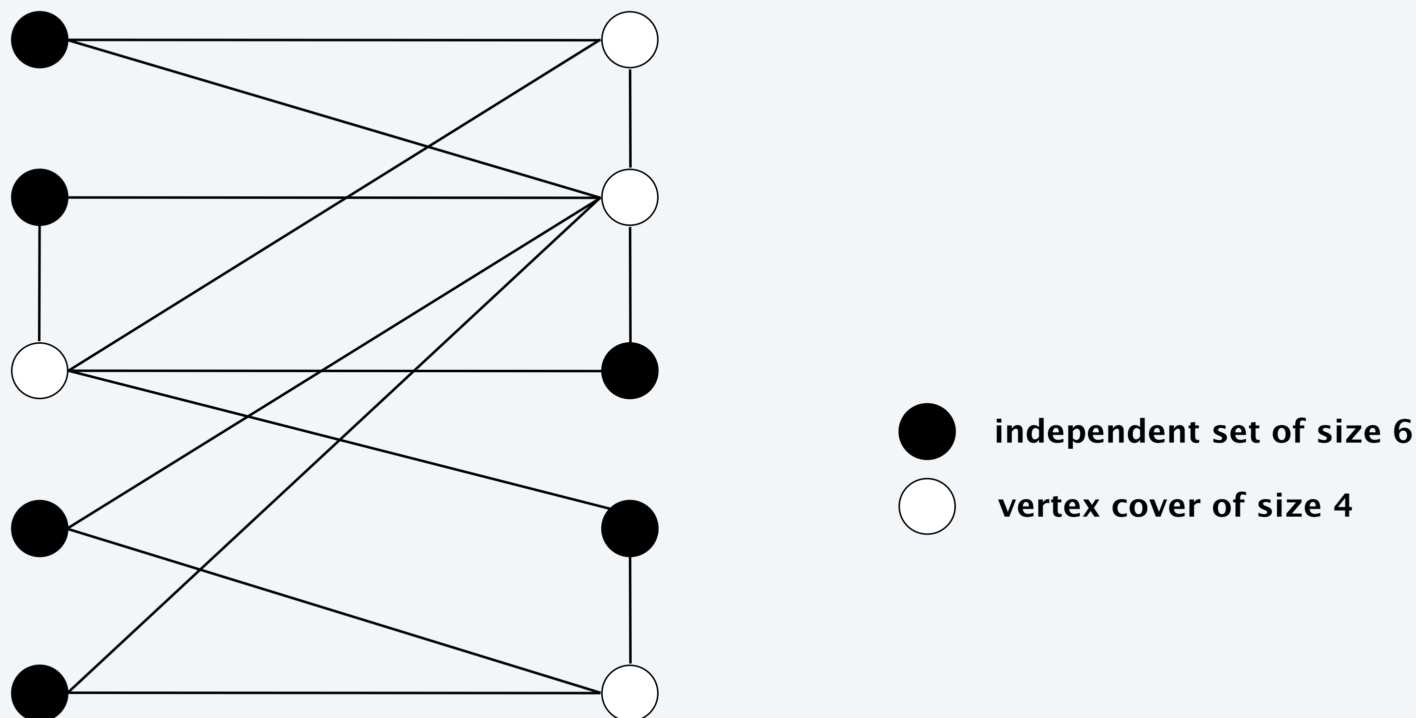
Ex. Is there an independent set of size $\geq 6$?
Ex. Is there an independent set of size $\geq 7$?



# 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 $S$?

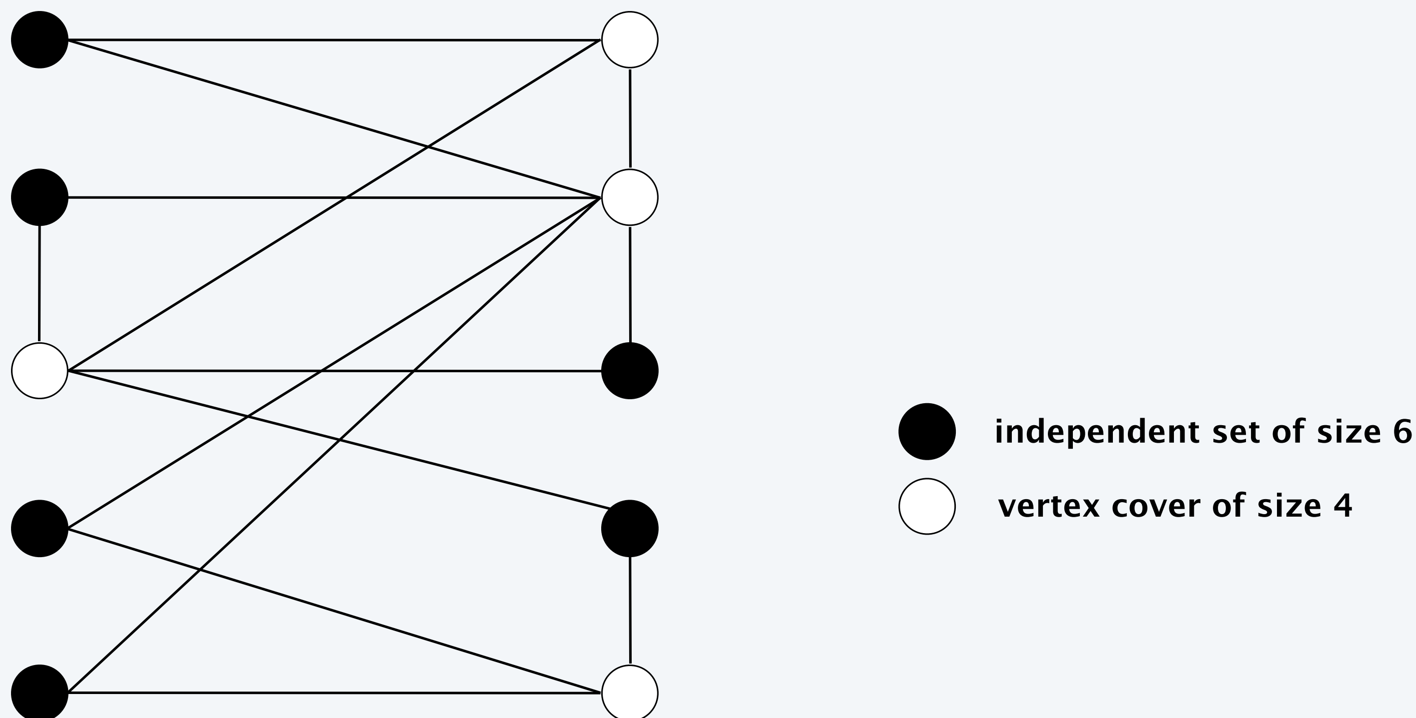
Ex. Is there a vertex cover of size $\leq 4$?
Ex. Is there a vertex cover of size $\leq 3$?



# Vertex cover and independent set reduce to one another

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

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

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 $(u, v)$.
- $S$ 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. VERTEX-COVER $\equiv_P$ INDEPENDENT-SET.

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 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. ▪

# 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 $i^{th}$ piece of software provides the set $S_i \subseteq U$ of capabilities.
- Goal: achieve all $n$ capabilities using fewest pieces of software.

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

$S_1 = \{ 3, 7 \}$ $\quad S_4 = \{ 2, 4 \}$

$S_2 = \{ 3, 4, 5, 6 \}$ $\quad S_5 = \{ 5 \}$

$S_3 = \{ 1 \}$ $\quad S_6 = \{ 1, 2, 6, 7 \}$

$k = 2$

a set cover instance

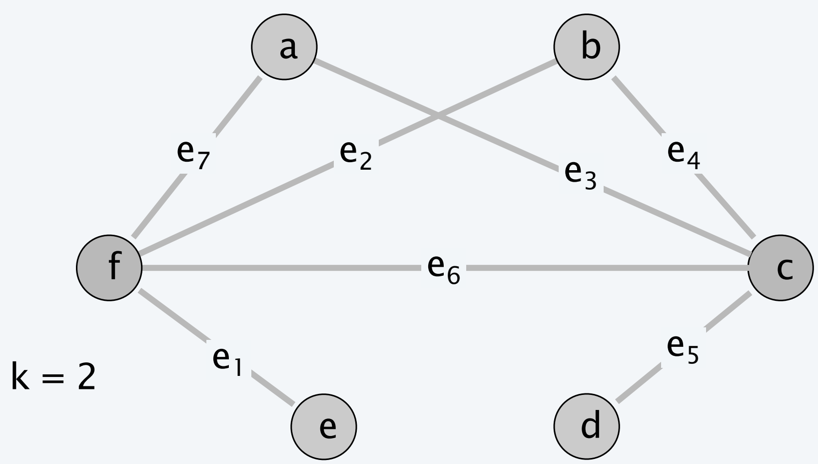
# Vertex cover reduces to set cover

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

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

**Construction.**

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




**vertex cover instance**
**(k = 2)**

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

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

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

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

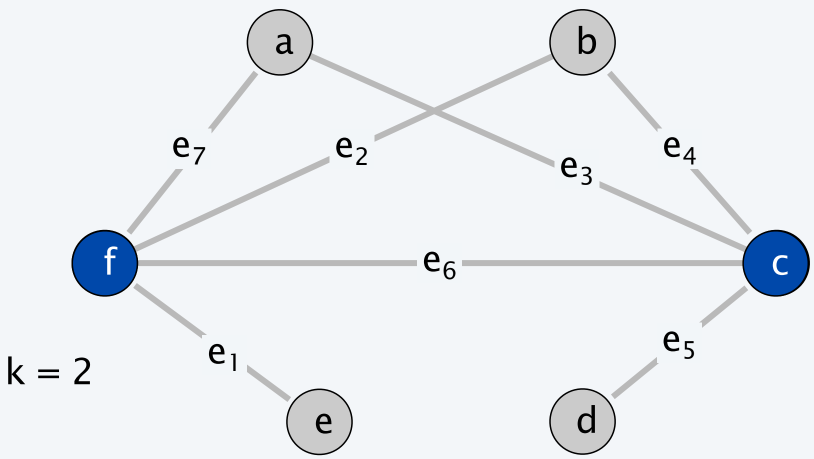
**set cover instance**
**(k = 2)**

# Vertex cover reduces to set cover

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




**vertex cover instance**
**(k = 2)**

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

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

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

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

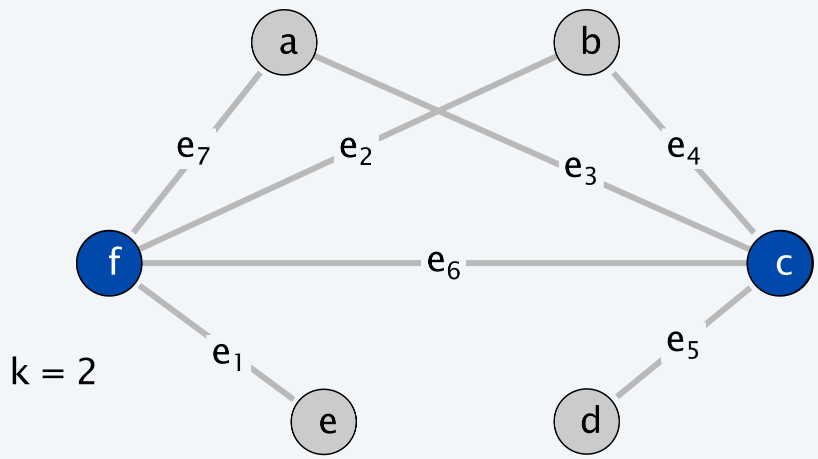
**set cover instance**
**(k = 2)**

# Vertex cover reduces to set cover

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

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

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




**vertex cover instance**
**(k = 2)**

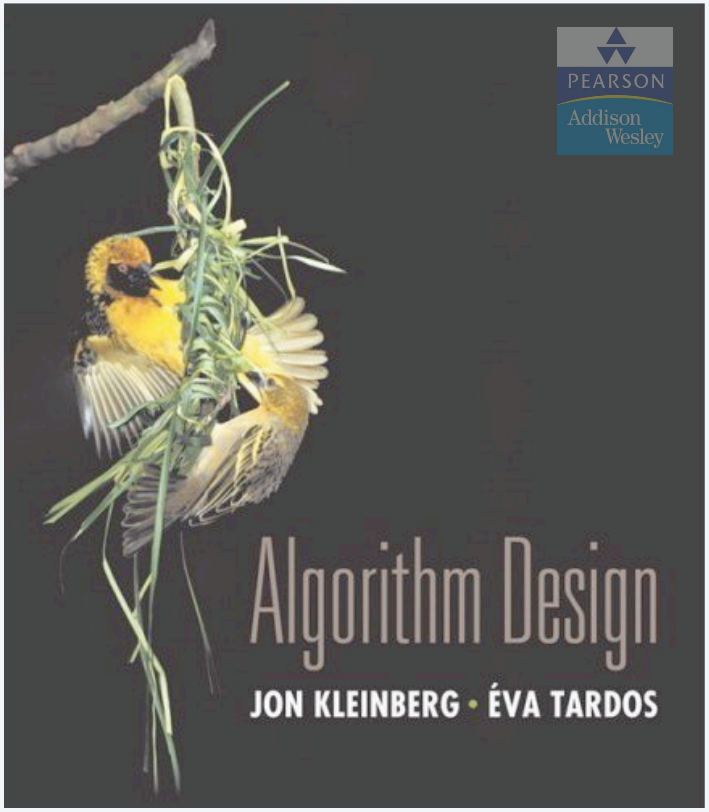
$U = \{ 1, 2, 3, 4, 5, 6, 7 \}$

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

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

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

**set cover instance**
**(k = 2)**




SECTION 8.2

# 8. INTRACTABILITY I

- *poly-time reductions*
- *packing and covering problems*
- *constraint satisfaction problems*
- *sequencing problems*
- *partitioning problems*
- *graph coloring*
- *numerical problems*

# 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 $\Phi$ that is the conjunction of clauses. $\Phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$

SAT. Given CNF formula $\Phi$, 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).

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

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

Key application. Electronic design automation (EDA).

# 3-satisfiability reduces to independent set

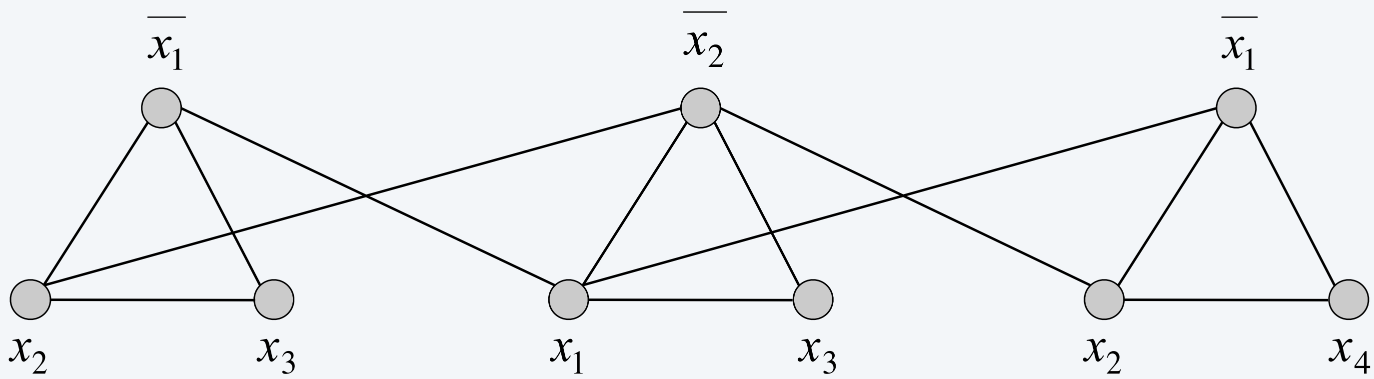
Theorem. 3-SAT $\leq_P$ INDEPENDENT-SET.

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

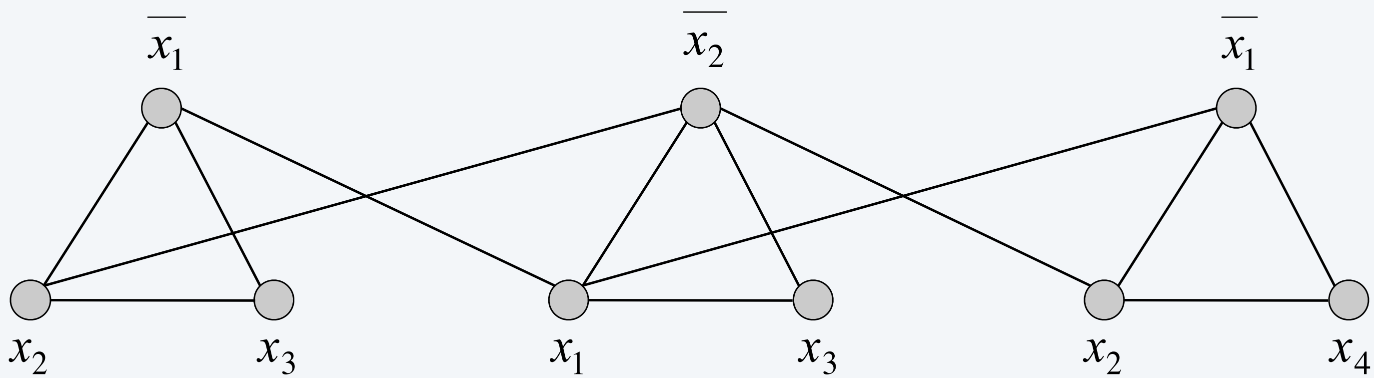
Lemma. $G$ contains independent set of size $k = |\Phi|$ iff $\Phi$ is satisfiable.

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

- $S$ must contain exactly one node in each triangle.
- Set these literals to *true* (and remaining variables consistently).
- 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**

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

## 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.

# Search problems

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

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

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

- Determine if there exists a vertex cover of size $\leq k$.
- 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

Bottom line. VERTEX-COVER $\equiv_P$ FIND-VERTEX-COVER.

# Optimization problems

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

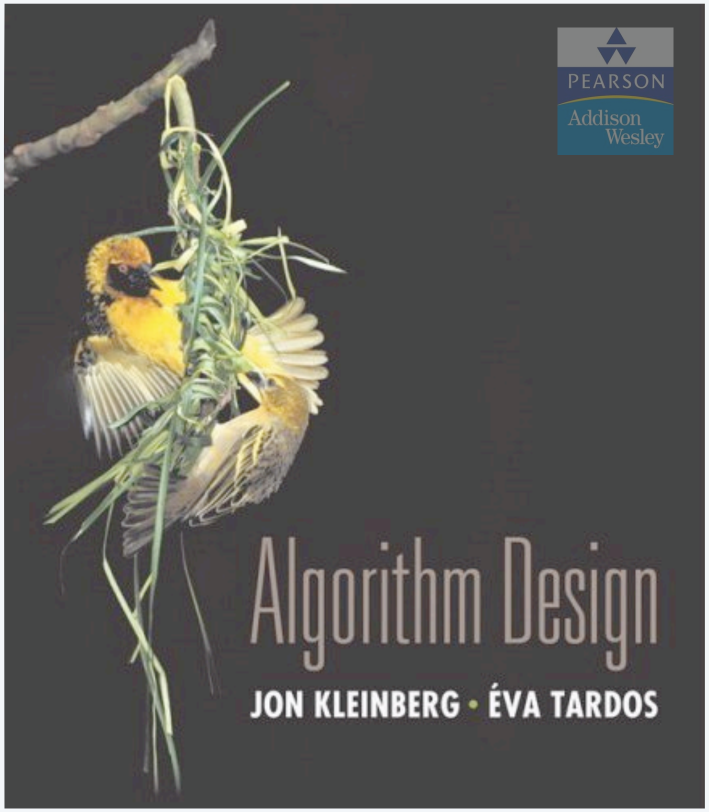
Search problem. Find a vertex cover of size ≤ $k$.

Optimization problem. Find a vertex cover of minimum size.

Ex. To find vertex cover of minimum size:

- (Binary) search for size $k^*$ of min vertex cover.
- Solve corresponding search problem.

Bottom line. VERTEX-COVER $\equiv_P$ FIND-VERTEX-COVER $\equiv_P$ OPTIMAL-VERTEX-COVER.



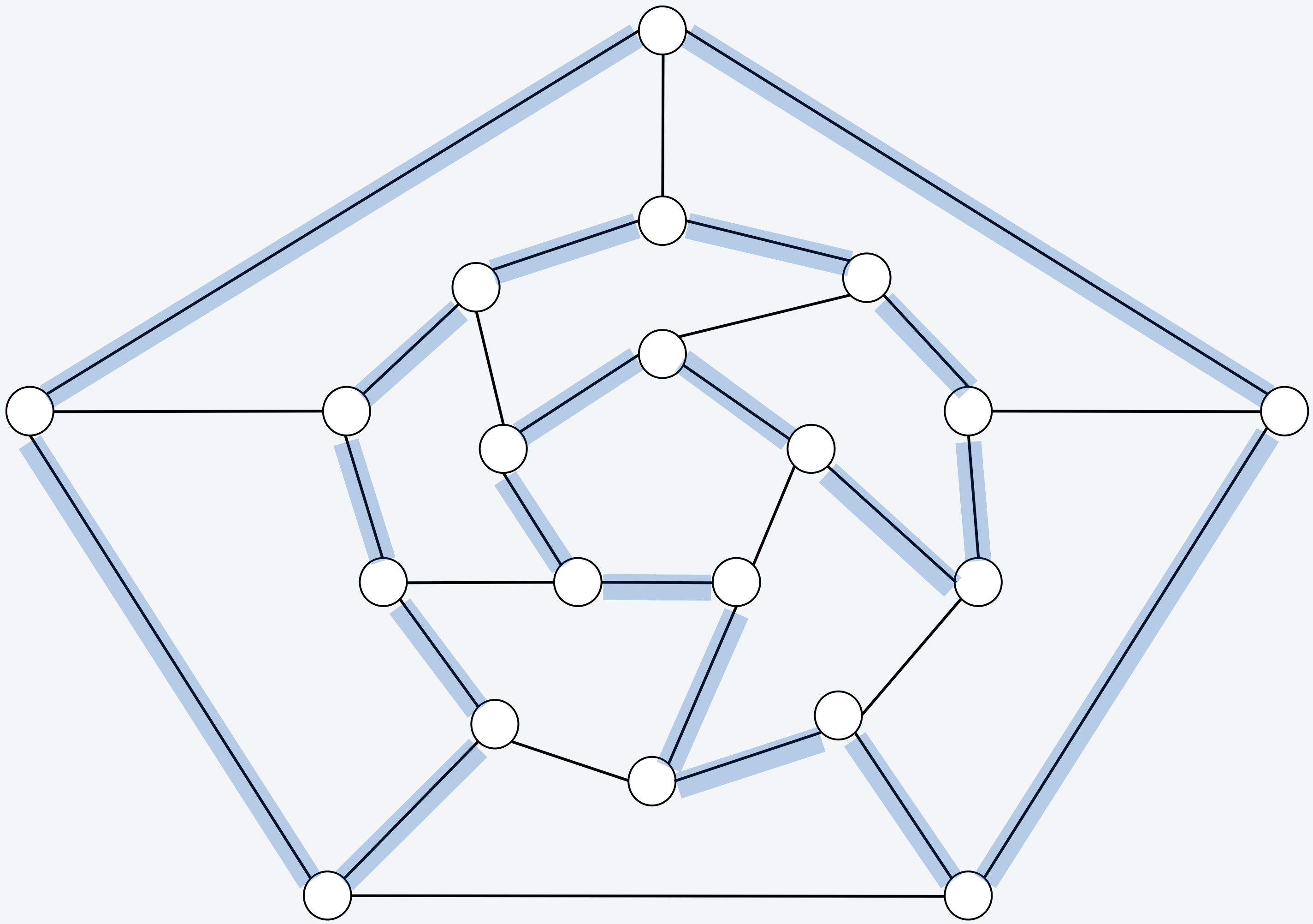

**SECTION 8.5**

# 8. INTRACTABILITY I

- *poly-time reductions*
- *packing and covering problems*
- *constraint satisfaction problems*
- *sequencing problems*
- *partitioning problems*
- *graph coloring*
- *numerical problems*

# Hamilton cycle

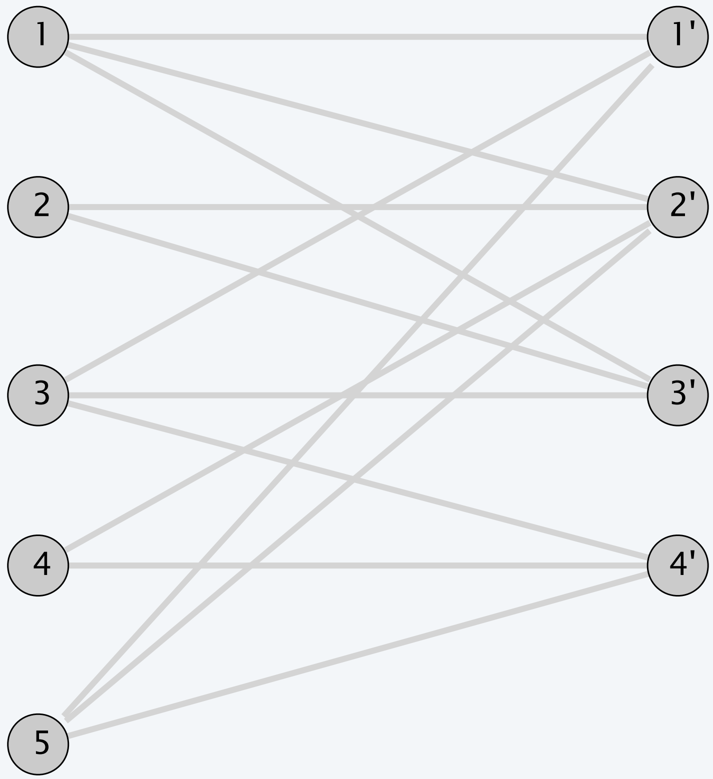
HAM-CYCLE. Given an undirected graph $G = (V, E)$, does there exist a simple cycle $\Gamma$ that contains every node in $V$?



**yes**

# Hamilton cycle

HAM-CYCLE. Given an undirected graph $G = (V, E)$, does there exist a simple cycle $\Gamma$ that contains every node in $V$ ?



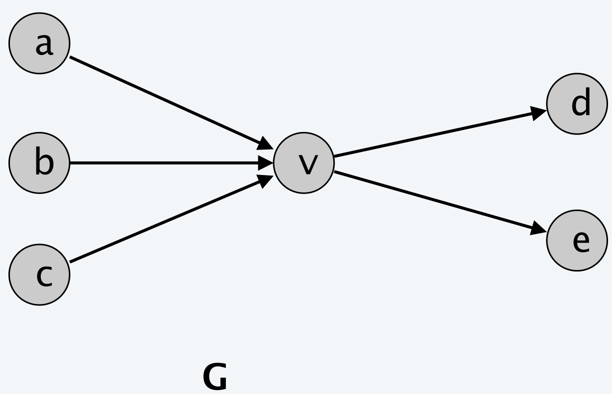

**no**

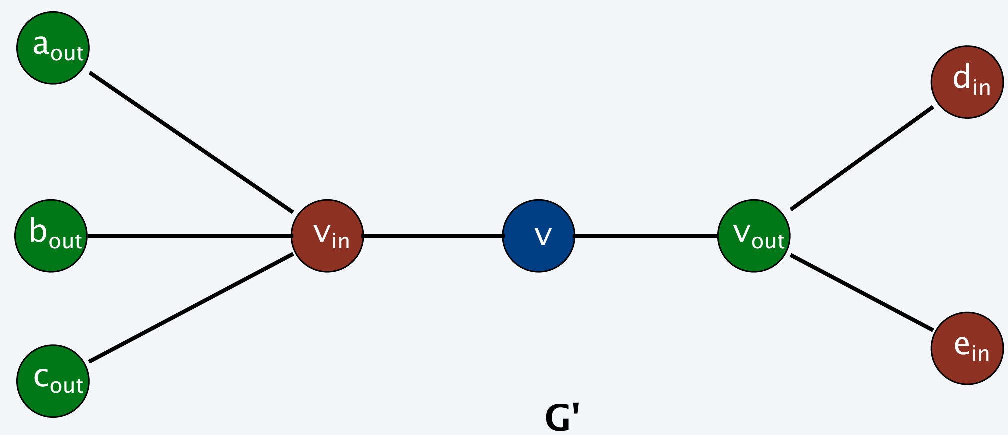
# Directed hamilton cycle reduces to hamilton cycle

DIR-HAM-CYCLE: Given a digraph $G = (V, E)$, does there exist a simple directed cycle $\Gamma$ that contains every node in $V$ ?

Theorem. DIR-HAM-CYCLE $\leq_P$ HAM-CYCLE.

Pf. Given a digraph $G = (V, E)$, construct a graph $G'$ with $3n$ nodes.



# Directed hamilton cycle reduces to hamilton cycle

Lemma. $G$ has a directed Hamilton cycle iff $G'$ has a Hamilton cycle.

Pf. $\Rightarrow$

- Suppose $G$ has a directed Hamilton cycle $\Gamma$.
- Then $G'$ has an undirected Hamilton cycle (same order).

Pf. $\Leftarrow$

- Suppose $G'$ has an undirected Hamilton cycle $\Gamma'$.
- $\Gamma'$ must visit nodes in $G'$ using one of following two orders:

  $\ldots, B, G, R, B, G, R, B, G, R, B, \ldots$

  $\ldots, B, R, G, B, R, G, B, R, G, B, \ldots$
- Blue nodes in $\Gamma'$ make up directed Hamilton cycle $\Gamma$ in $G$, or reverse of one. ▪

# 3-satisfiability reduces to directed hamilton cycle

Theorem. 3-SAT $\le_P$ DIR-HAM-CYCLE.

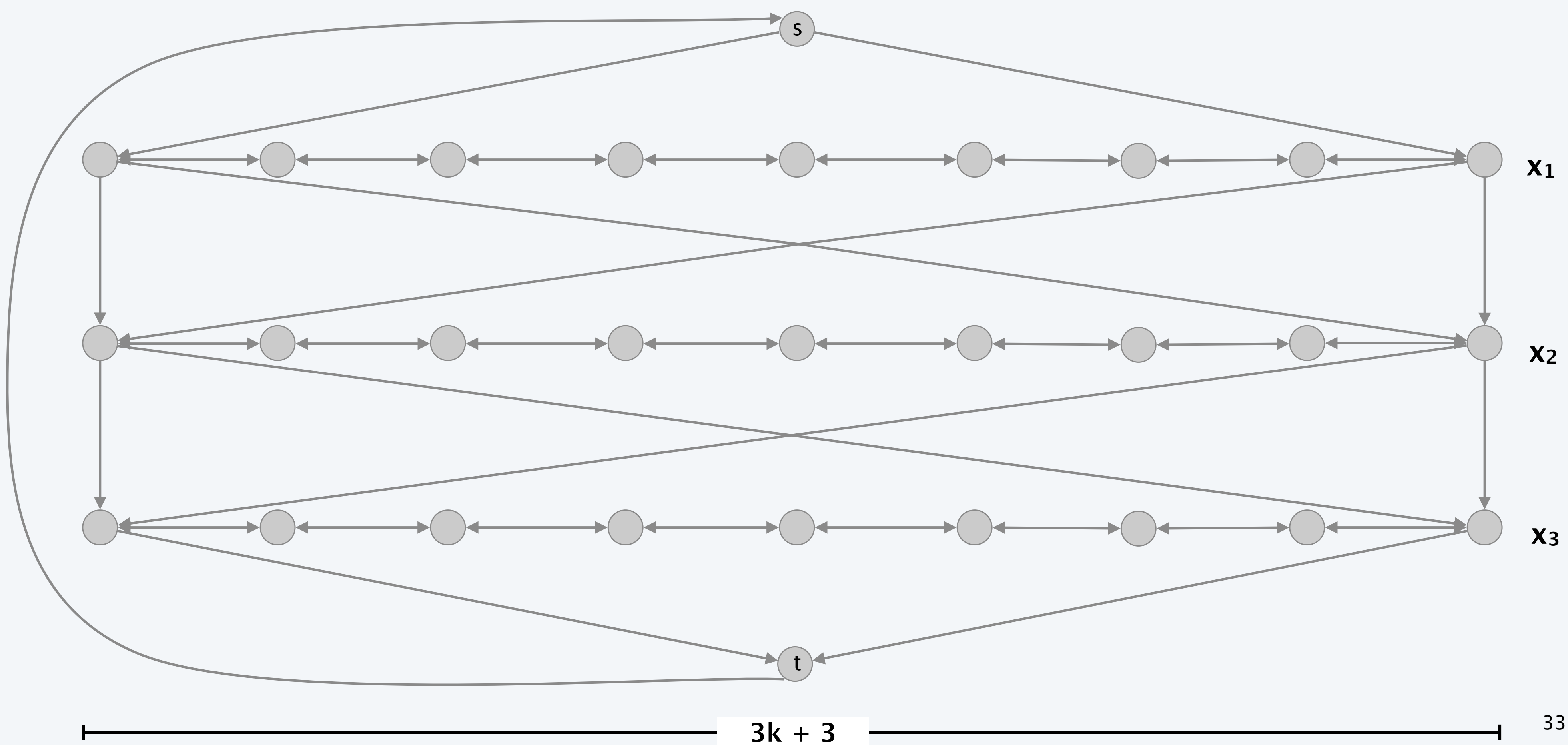
Pf. Given an instance $\Phi$ of 3-SAT, we construct an instance of DIR-HAM-CYCLE that has a Hamilton cycle iff $\Phi$ is satisfiable.

Construction. First, create graph that has $2^n$ Hamilton cycles which correspond in a natural way to $2^n$ possible truth assignments.

# 3-satisfiability reduces to directed hamilton cycle

Construction. Given 3-SAT instance $\Phi$ with $n$ variables $x_i$ and $k$ clauses.

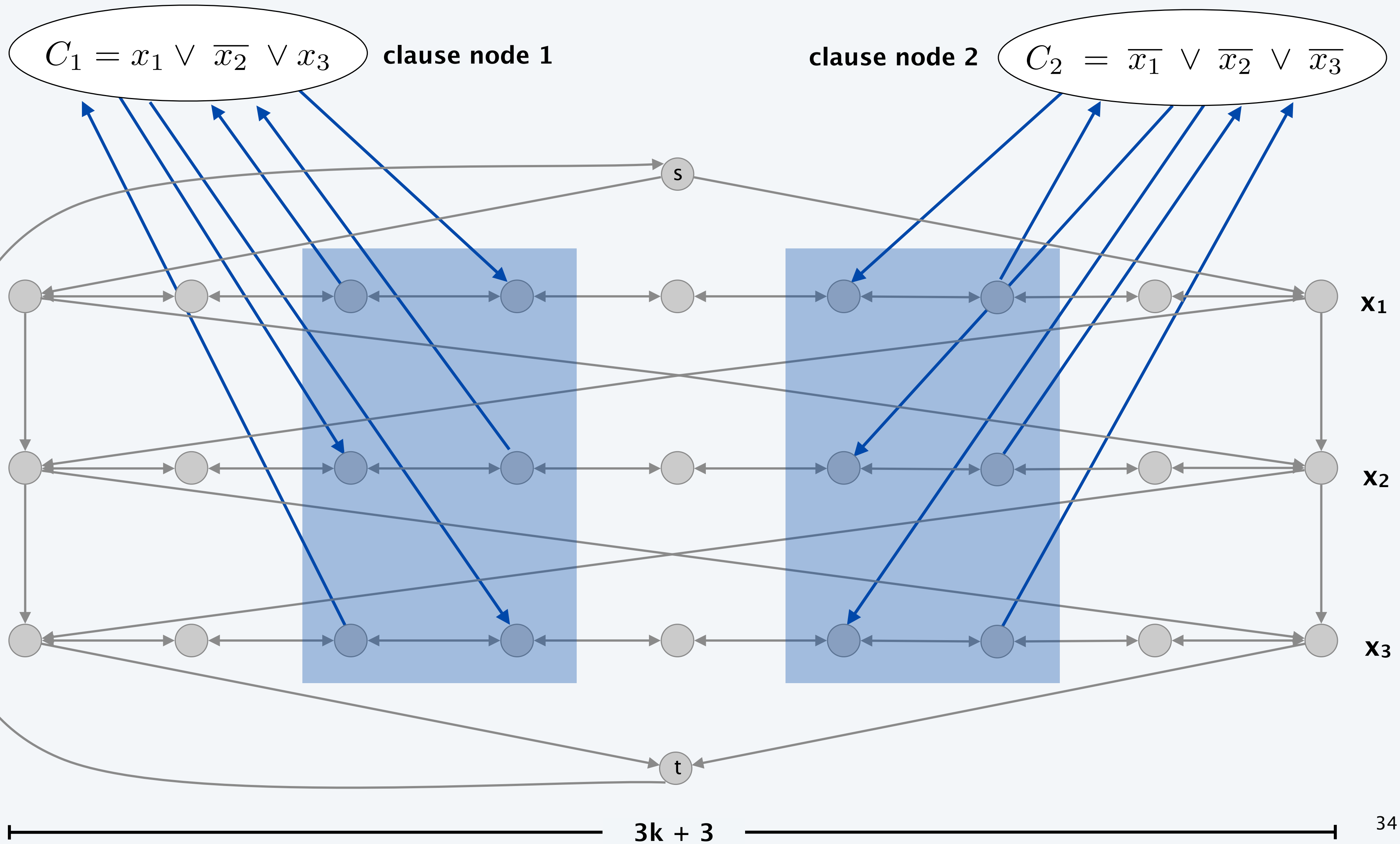
- Construct $G$ to have $2^n$ Hamilton cycles.
- Intuition: traverse path $i$ from left to right $\Leftrightarrow$ set variable $x_i = \mathit{true}$.



# 3-satisfiability reduces to directed hamilton cycle

Construction. Given 3-SAT instance $\Phi$ with $n$ variables $x_i$ and $k$ clauses.

- For each clause, add a node and 6 edges.



# 3-satisfiability reduces to directed hamilton cycle

Lemma. $\Phi$ is satisfiable iff $G$ has a Hamilton cycle.

Pf. $\Rightarrow$

- Suppose 3-SAT instance has satisfying assignment $x^*$.
- Then, define Hamilton cycle in $G$ as follows:
  - if $x^*_i = true$, traverse row $i$ from left to right
  - if $x^*_i = false$, traverse row $i$ from right to left
  - for each clause $C_j$, there will be at least one row $i$ in which we are going in "correct" direction to splice clause node $C_j$ into cycle (and we splice in $C_j$ exactly once)

# 3-satisfiability reduces to directed hamilton cycle

Lemma. $\Phi$ is satisfiable iff $G$ has a Hamilton cycle.

Pf. $\Leftarrow$

- Suppose $G$ has a Hamilton cycle $\Gamma$.
- If $\Gamma$ enters clause node $C_j$ , it must depart on mate edge.
  - nodes immediately before and after $C_j$ are connected by an edge $e \in E$
  - removing $C_j$ from cycle, and replacing it with edge $e$ yields Hamilton cycle on $G - \{ C_j \}$
- Continuing in this way, we are left with a Hamilton cycle $\Gamma'$ in $G - \{ C_1, C_2, \ldots, C_k \}$.
- Set $x^*_i = true$ iff $\Gamma'$ traverses row $i$ left to right.
- Since $\Gamma$ visits each clause node $C_j$ , at least one of the paths is traversed in "correct" direction, and each clause is satisfied. ▪

# 3-satisfiability reduces to longest path

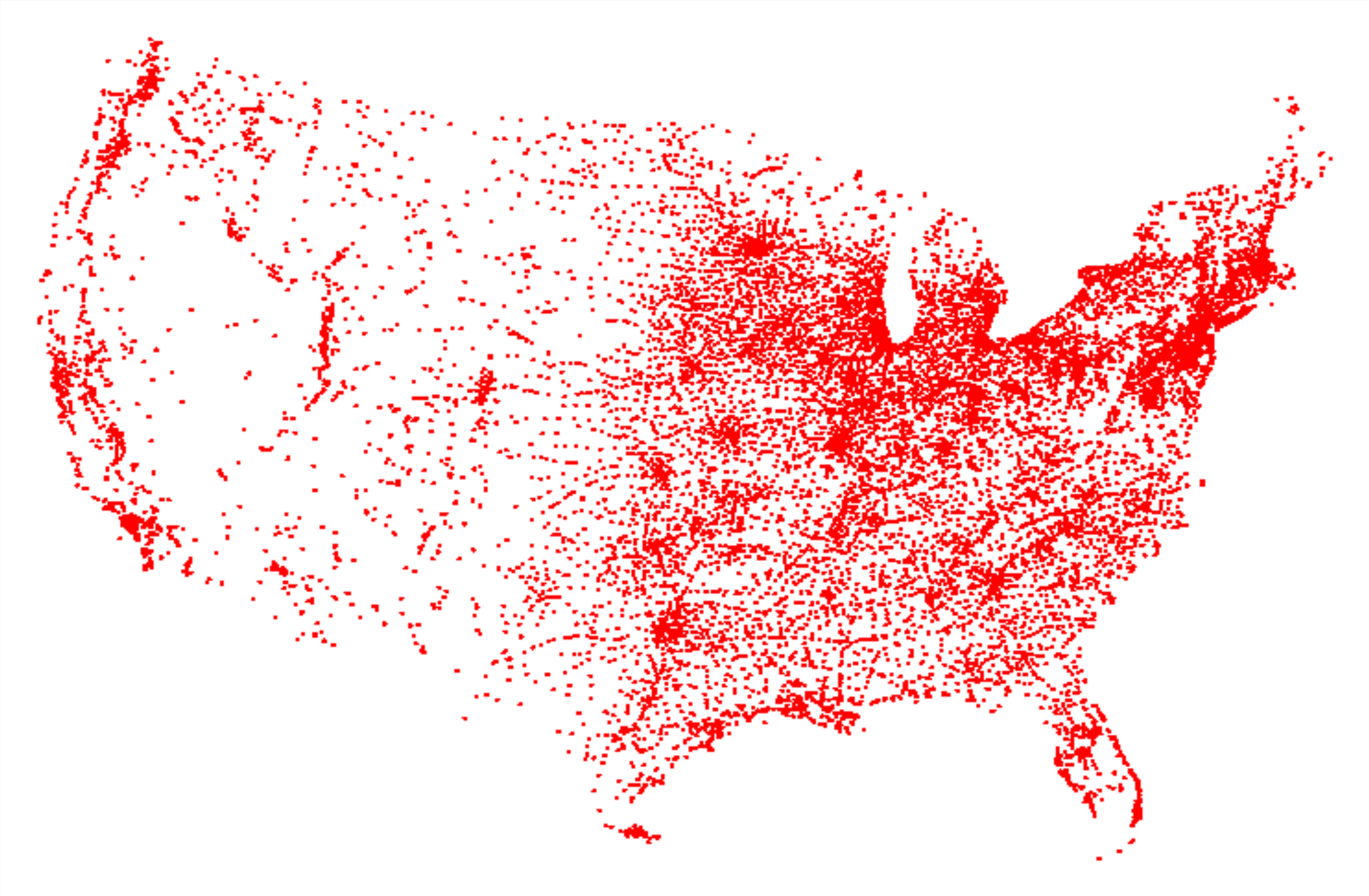
LONGEST-PATH. Given a directed graph $G = (V, E)$, does there exists a simple path consisting of at least $k$ edges?

Theorem. 3-SAT $\leq_P$ LONGEST-PATH.

Pf 1. Redo proof for DIR-HAM-CYCLE, ignoring back-edge from $t$ to $s$.
Pf 2. Show HAM-CYCLE $\leq_P$ LONGEST-PATH.

# Traveling salesperson problem

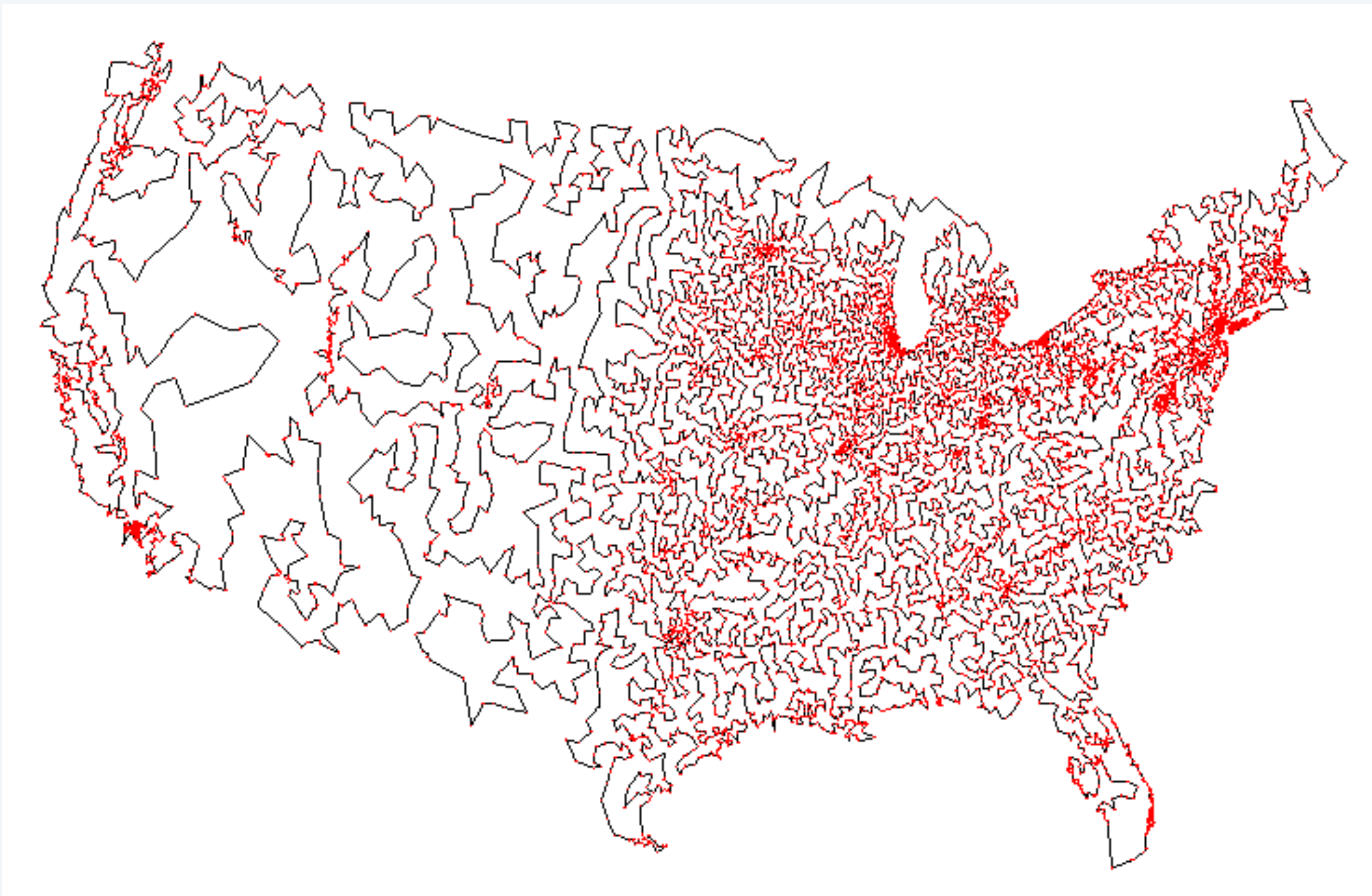
TSP. Given a set of $n$ cities and a pairwise distance function $d(u, v)$, is there a tour of length $\leq D$ ?



**13,509 cities in the United States**
**http://www.tsp.gatech.edu**

# Traveling salesperson problem

TSP. Given a set of $n$ cities and a pairwise distance function $d(u, v)$, is there a tour of length $\leq D$ ?



**optimal TSP tour**
**http://www.tsp.gatech.edu**

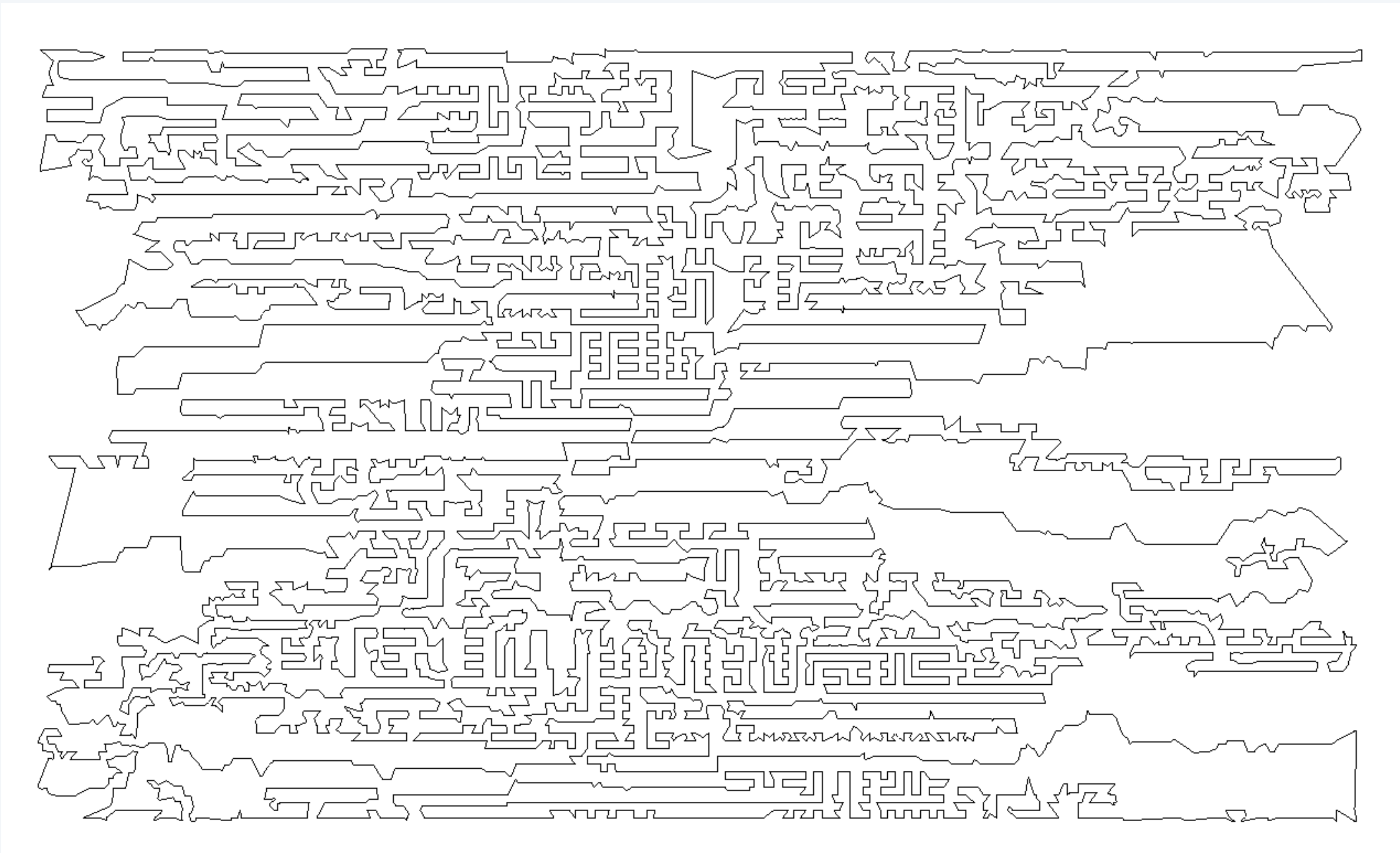
# Traveling salesperson problem

TSP. Given a set of $n$ cities and a pairwise distance function $d(u, v)$, is there a tour of length $\leq D$ ?

**11,849 holes to drill in a programmed logic array**
**http://www.tsp.gatech.edu**

# Traveling salesperson problem

TSP. Given a set of $n$ cities and a pairwise distance function $d(u, v)$, is there a tour of length $\leq D$ ?



**optimal TSP tour**
**http://www.tsp.gatech.edu**

# Hamilton cycle reduces to traveling salesperson problem

TSP. Given a set of $n$ cities and a pairwise distance function $d(u, v)$, is there a tour of length $\leq D$ ?

HAM-CYCLE. Given an undirected graph $G = (V, E)$, does there exist a simple cycle $\Gamma$ that contains every node in $V$ ?

Theorem. HAM-CYCLE $\leq_P$ TSP.

Pf.

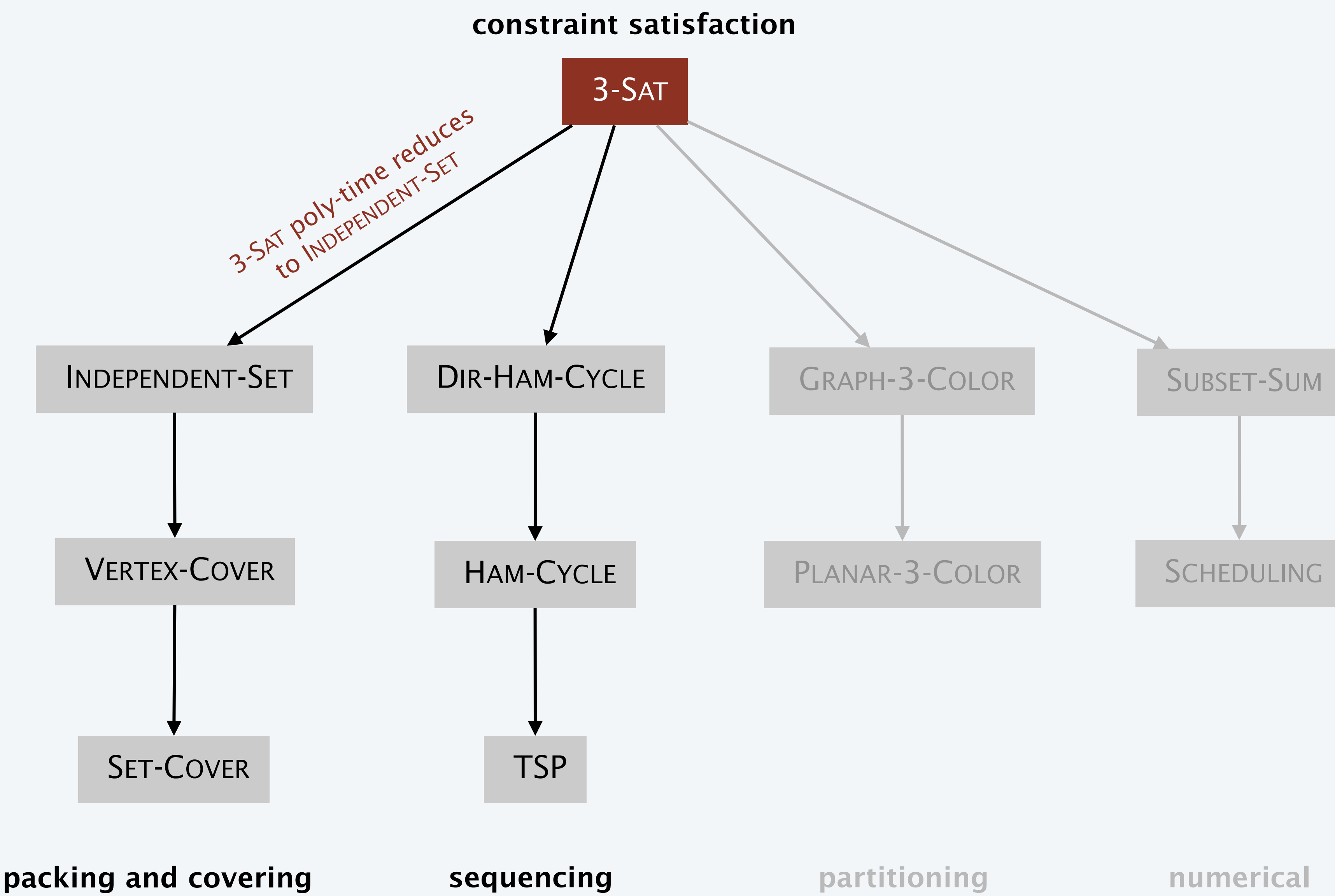
- Given instance $G = (V, E)$ of HAM-CYCLE, create $n$ cities with distance function

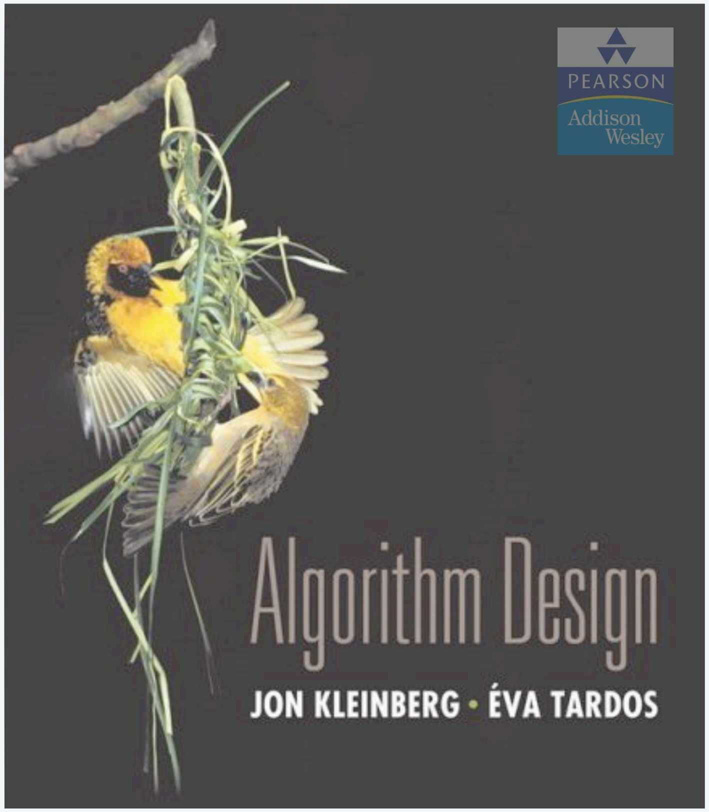
$$d(u, v) = \begin{cases} 1 & \text{if } (u, v) \in E \\ 2 & \text{if } (u, v) \notin E \end{cases}$$

- TSP instance has tour of length $\leq n$ iff $G$ has a Hamilton cycle. ▪

Remark. TSP instance satisfies triangle inequality: $d(u, w) \leq d(u, v) + d(v, w)$.

# Polynomial-time reductions






SECTION 8.6

# 8. INTRACTABILITY I

- *poly-time reductions*
- *packing and covering problems*
- *constraint satisfaction problems*
- *sequencing problems*
- *partitioning problems*
- *graph coloring*
- *numerical problems*

# 3-dimensional matching

3D-MATCHING. Given $n$ instructors, $n$ courses, and $n$ times, and a list of the possible courses and times each instructor is willing to teach, is it possible to make an assignment so that all courses are taught at different times?

| instructor | course | time |
|---|---|---|
| Wayne | COS 226 | TTh 11–12:20 |
| Wayne | COS 423 | MW 11–12:20 |
| Wayne | COS 423 | TTh 11–12:20 |
| Tardos | COS 423 | TTh 3–4:20 |
| Tardos | COS 523 | TTh 3–4:20 |
| Kleinberg | COS 226 | TTh 3–4:20 |
| Kleinberg | COS 226 | MW 11–12:20 |
| Kleinberg | COS 423 | MW 11–12:20 |

# 3-dimensional matching

3D-MATCHING. Given 3 disjoint sets $X$, $Y$, and $Z$, each of size $n$ and a set $T \subseteq X \times Y \times Z$ of triples, does there exist a set of $n$ triples in $T$ such that each element of $X \cup Y \cup Z$ is in exactly one of these triples?

$X = \{ x_1, x_2, x_3 \}$, $\quad Y = \{ y_1, y_2, y_3 \}$, $\quad Z = \{ z_1, z_2, z_3 \}$

$T_1 = \{ x_1, y_1, z_2 \}$, $\quad T_2 = \{ x_1, y_2, z_1 \}$, $\quad T_3 = \{ x_1, y_2, z_2 \}$

$T_4 = \{ x_2, y_2, z_3 \}$, $\quad T_5 = \{ x_2, y_3, z_3 \}$,

$T_7 = \{ x_3, y_1, z_3 \}$, $\quad T_8 = \{ x_3, y_1, z_1 \}$, $\quad T_9 = \{ x_3, y_2, z_1 \}$

**an instance of 3d-matching (with n = 3)**

Remark. Generalization of bipartite matching.

# 3-dimensional matching

3D-MATCHING. Given $3$ disjoint sets $X$, $Y$, and $Z$, each of size $n$ and a set $T \subseteq X \times Y \times Z$ of triples, does there exist a set of $n$ triples in $T$ such that each element of $X \cup Y \cup Z$ is in exactly one of these triples?

Theorem. 3-SAT $\le_P$ 3D-MATCHING.

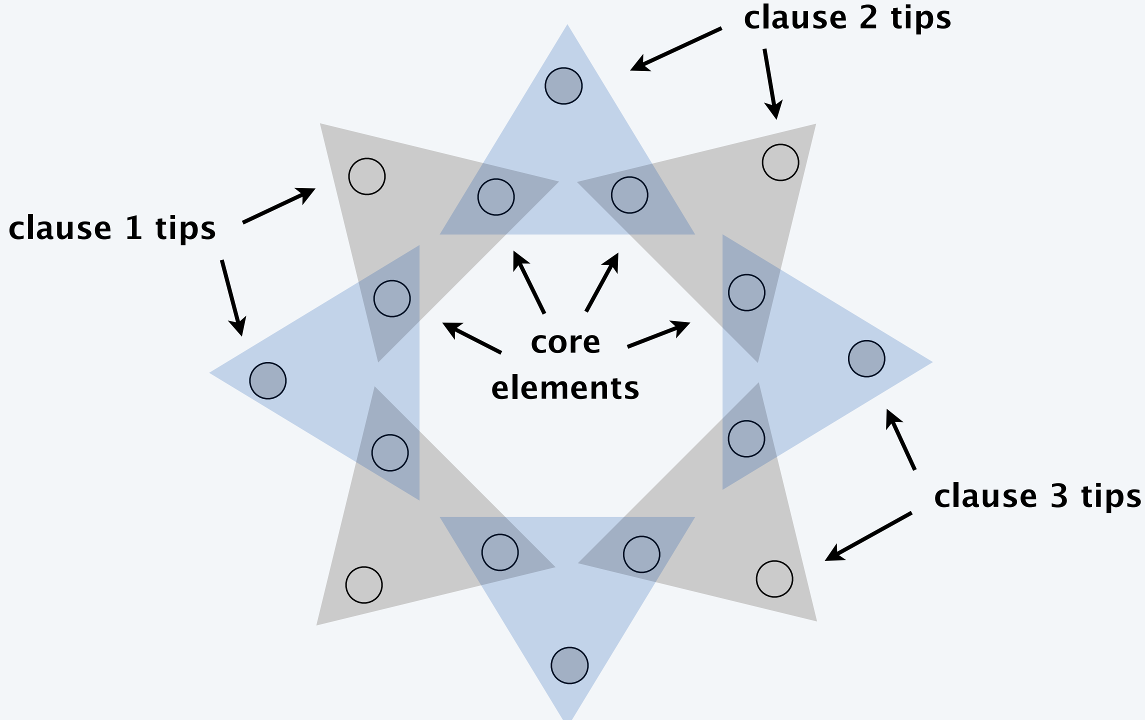
Pf. Given an instance $\Phi$ of 3-SAT, we construct an instance of 3D-MATCHING that has a perfect matching iff $\Phi$ is satisfiable.

# 3-satisfiability reduces to 3-dimensional matching

Construction. (part 1)

number of clauses

- Create gadget for each variable $x_i$ with $2k$ core elements and $2k$ tip ones.



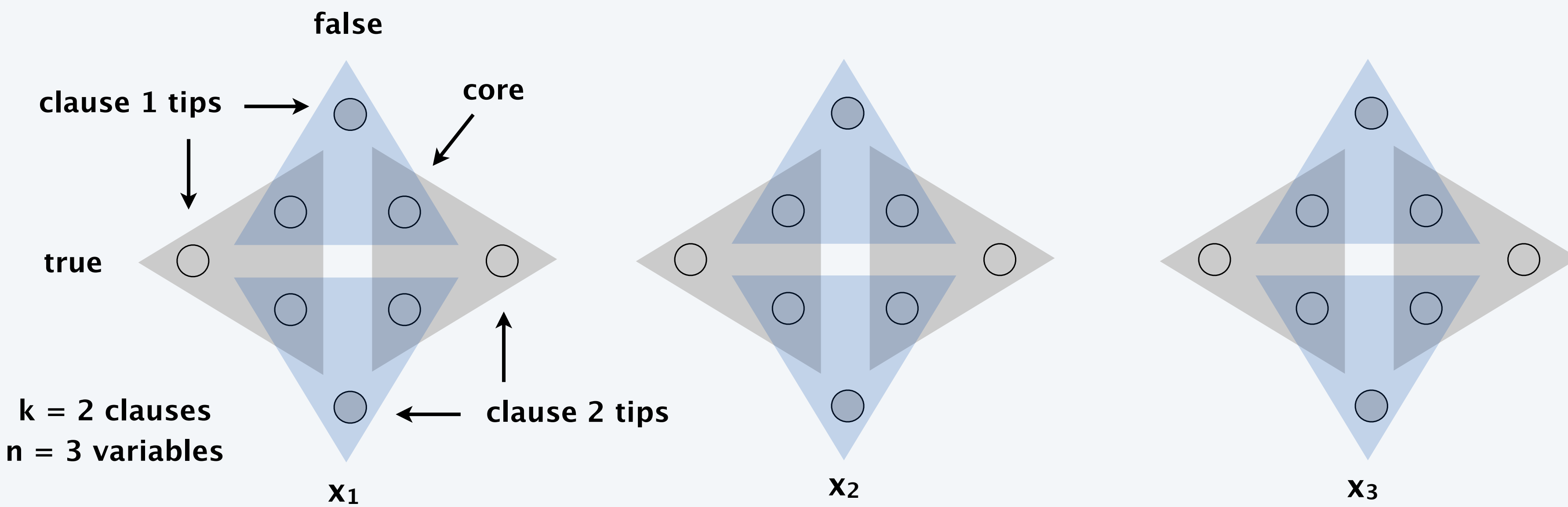

a gadget for variable $x_i$ ($k$ = 4)

# 3-satisfiability reduces to 3-dimensional matching

Construction. (part 1)

number of clauses

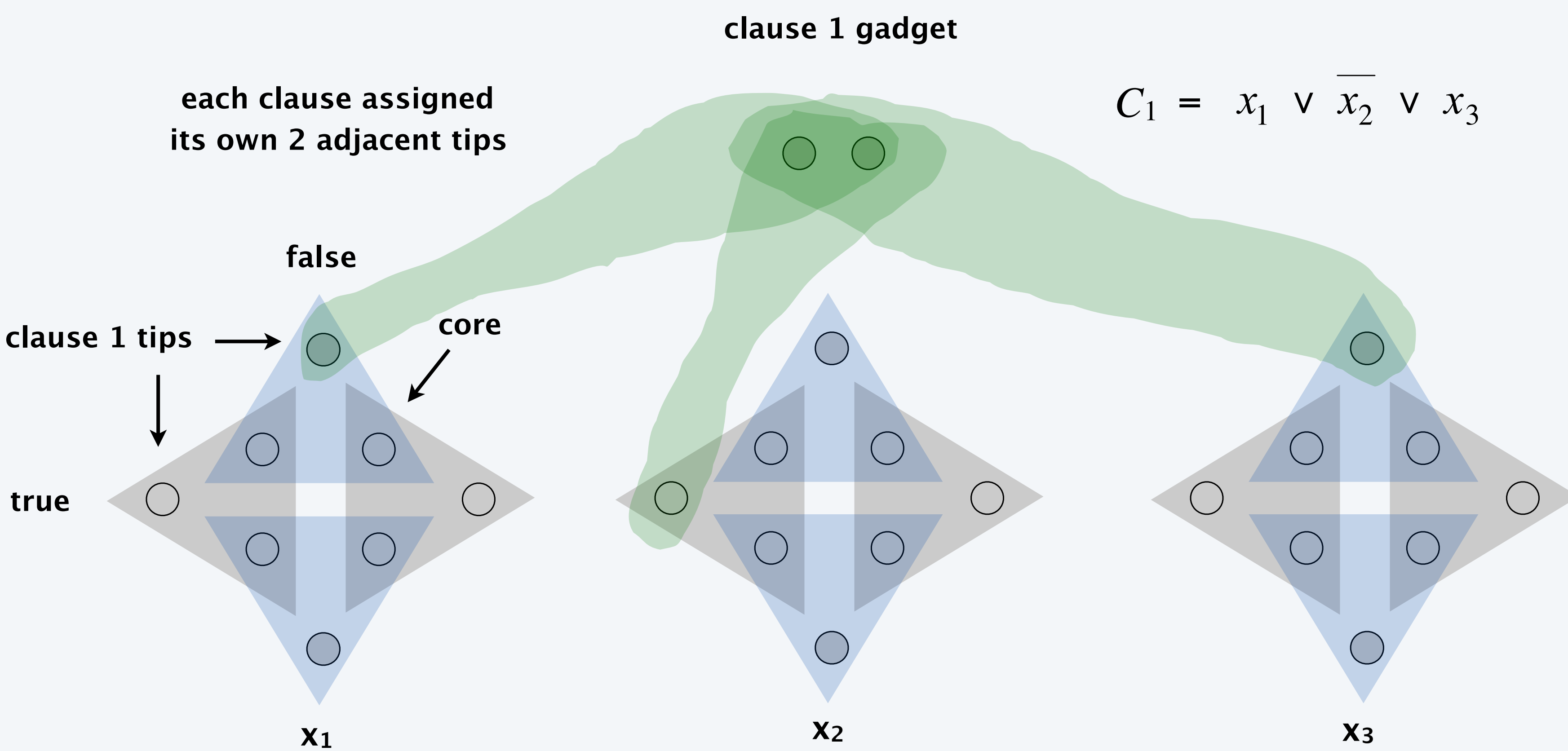
- Create gadget for each variable $x_i$ with $2k$ core elements and $2k$ tip ones.
- No other triples will use core elements.
- In gadget for $x_i$, any perfect matching must use either all gray triples (corresponding to $x_i = true$) or all blue ones (corresponding to $x_i = false$).



# 3-satisfiability reduces to 3-dimensional matching

**Construction.** (part 2)

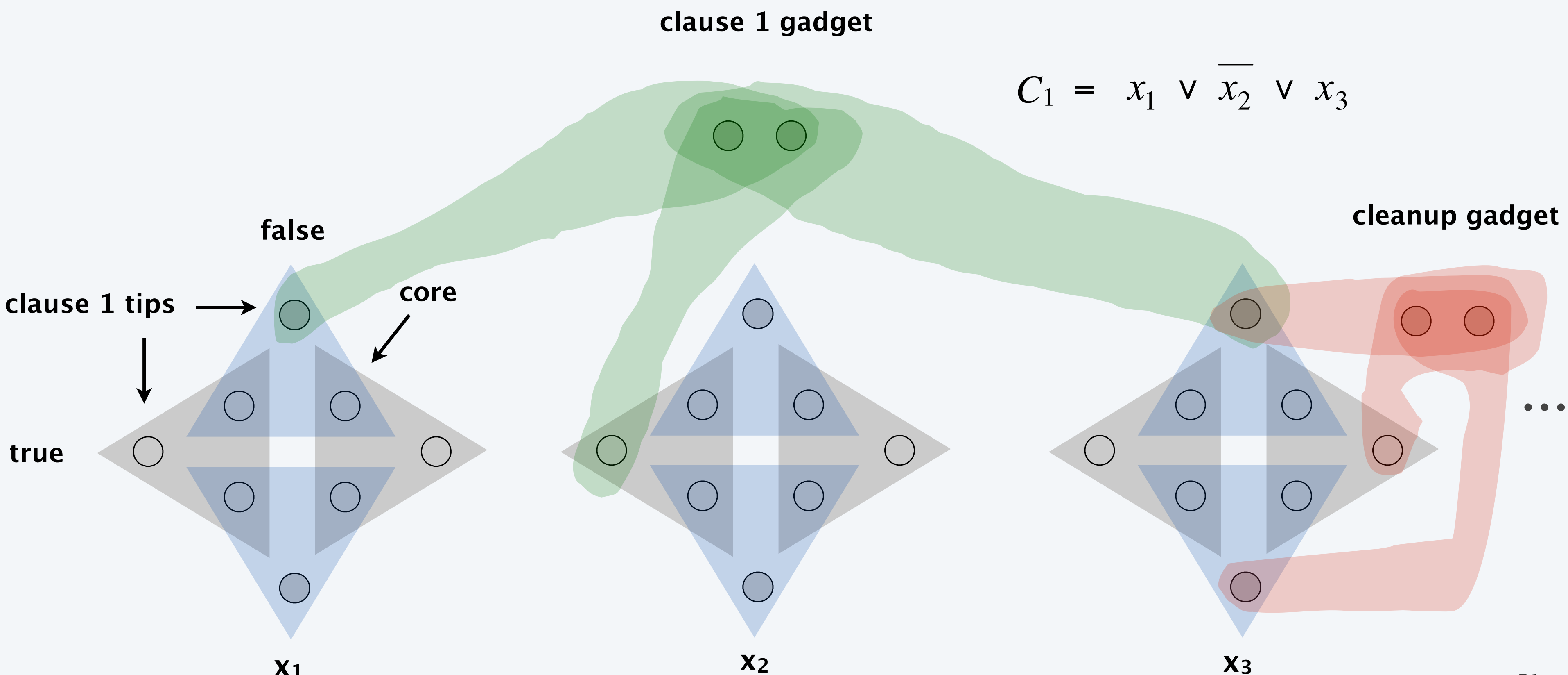
- Create gadget for each clause $C_j$ with two elements and three triples.
- Exactly one of these triples will be used in any 3d-matching.
- Ensures any perfect matching uses either (i) grey core of $x_1$ or (ii) blue core of $x_2$ or (iii) grey core of $x_3$.



# 3-satisfiability reduces to 3-dimensional matching

Construction. (part 3)

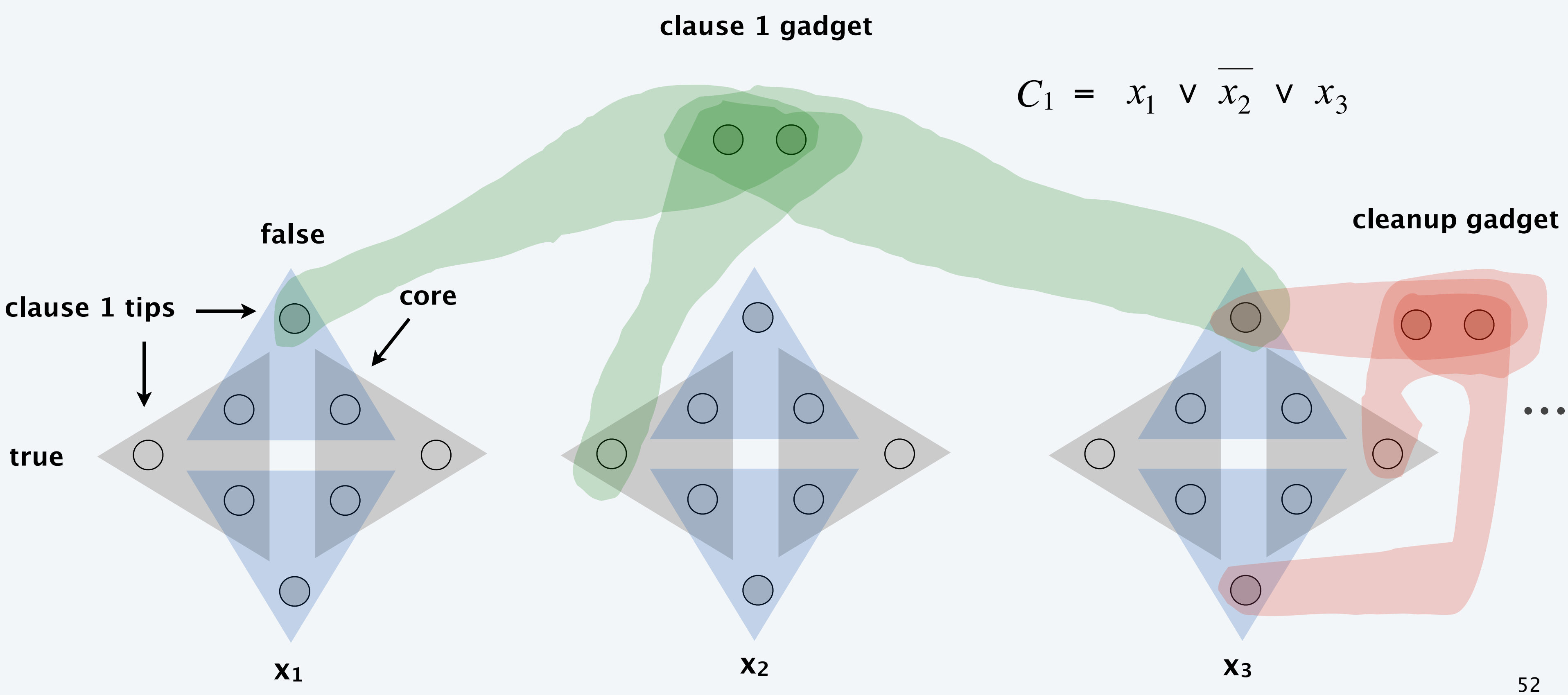
- There are $2nk$ tips: $nk$ covered by blue/gray triples; $k$ by clause triples.
- To cover remaining $(n-1)k$ tips, create $(n-1)k$ cleanup gadgets: same as clause gadget but with $2nk$ triples, connected to every tip.



# 3-satisfiability reduces to 3-dimensional matching

Lemma. Instance $(X, Y, Z)$ has a perfect matching iff $\Phi$ is satisfiable.

Q. What are $X$, $Y$, and $Z$?

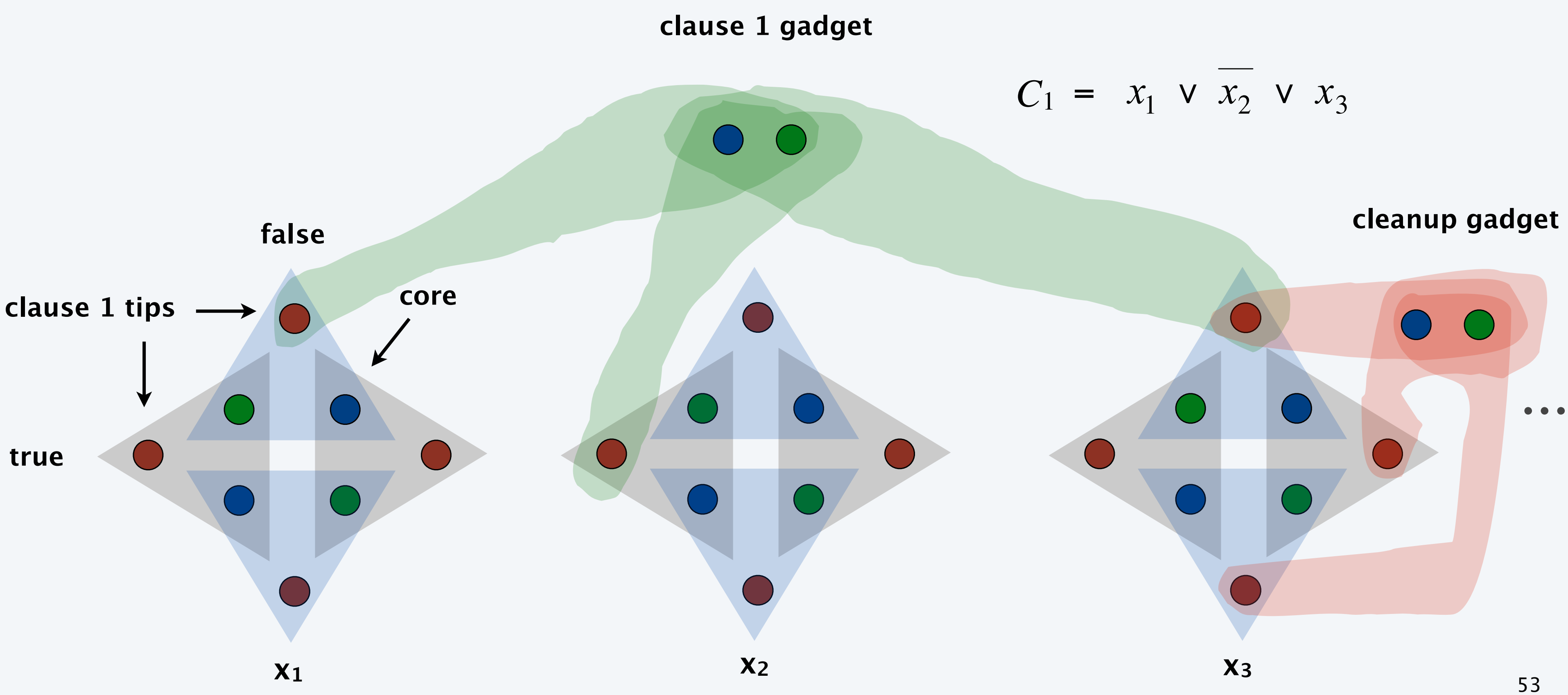


# 3-satisfiability reduces to 3-dimensional matching

Lemma. Instance $(X, Y, Z)$ has a perfect matching iff $\Phi$ is satisfiable.

Q. What are $X$, $Y$, and $Z$?

A. $X = red$, $Y = green$, and $Z = blue$.

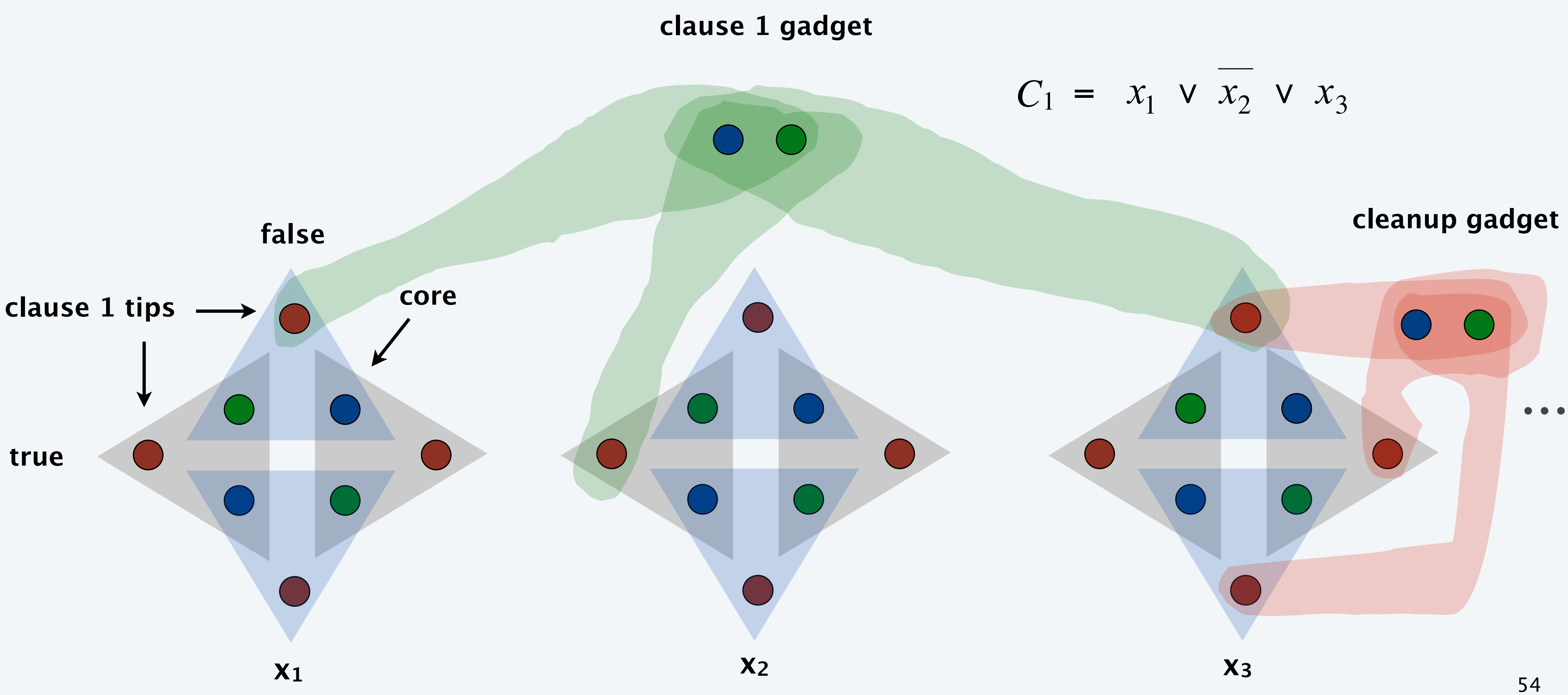


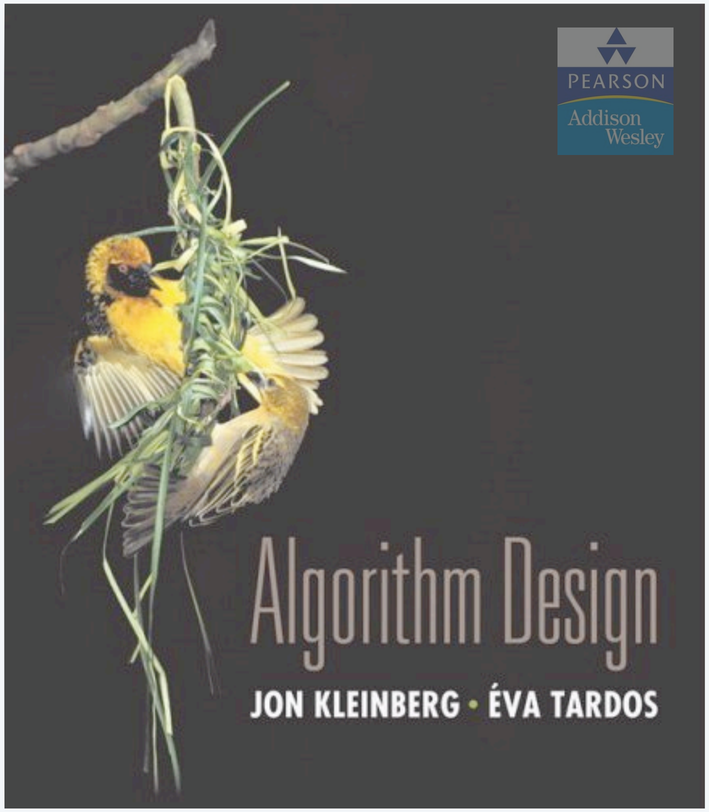
# 3-satisfiability reduces to 3-dimensional matching

Lemma. Instance $(X, Y, Z)$ has a perfect matching iff $\Phi$ is satisfiable.

Pf. $\Rightarrow$ If 3d-matching, then assign $x_i$ according to gadget $x_i$.

Pf. $\Leftarrow$ If $\Phi$ is satisfiable, use any true literal in $C_j$ to select gadget $C_j$ triple. ▪





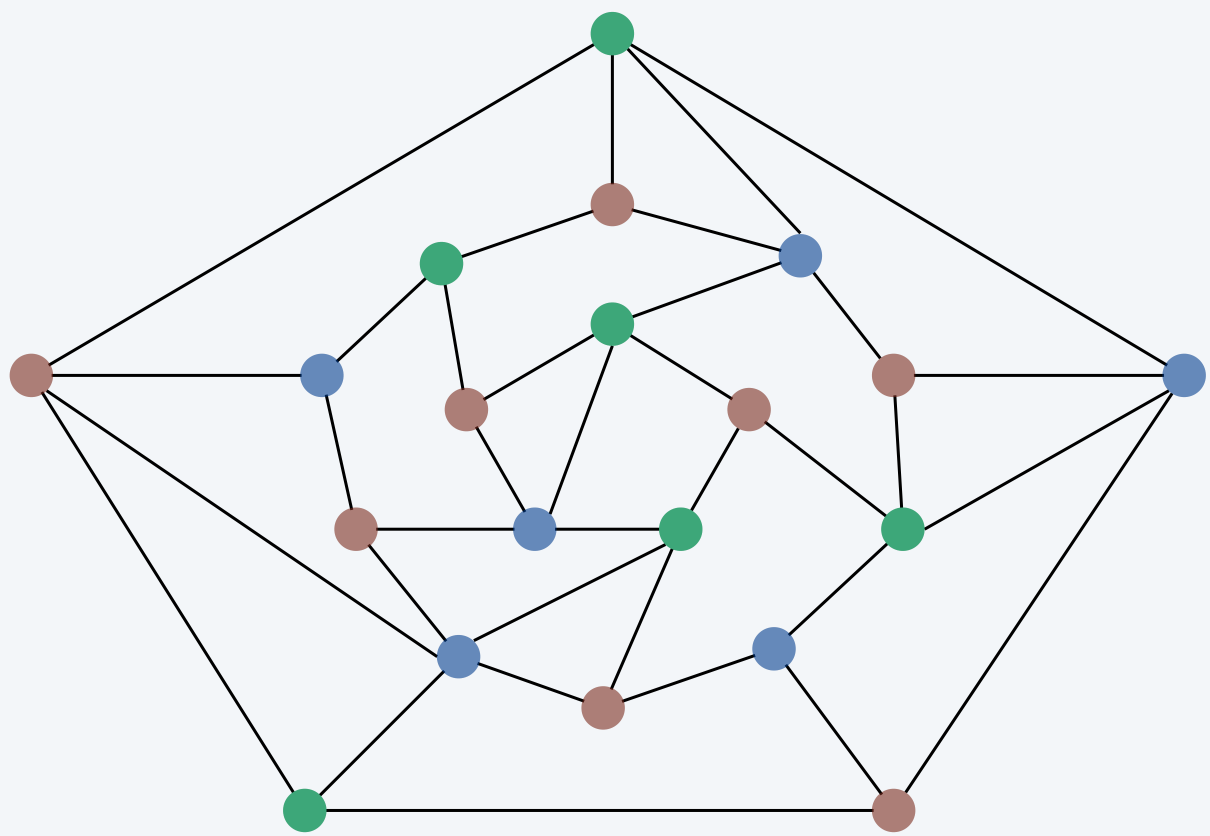

SECTION 8.7

# 8. INTRACTABILITY I

- poly-time reductions
- packing and covering problems
- constraint satisfaction problems
- sequencing problems
- partitioning problems
- graph coloring
- numerical problems

# 3-colorability

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



**yes instance**

## Application: register allocation

Register allocation. Assign program variables to machine register 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 names; 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 $\le_P$ K-REGISTER-ALLOCATION for any constant $k \ge 3$.

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 $\le_P$ 3-COLOR.

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

# 3-satisfiability reduces to 3-colorability

**Construction.**

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

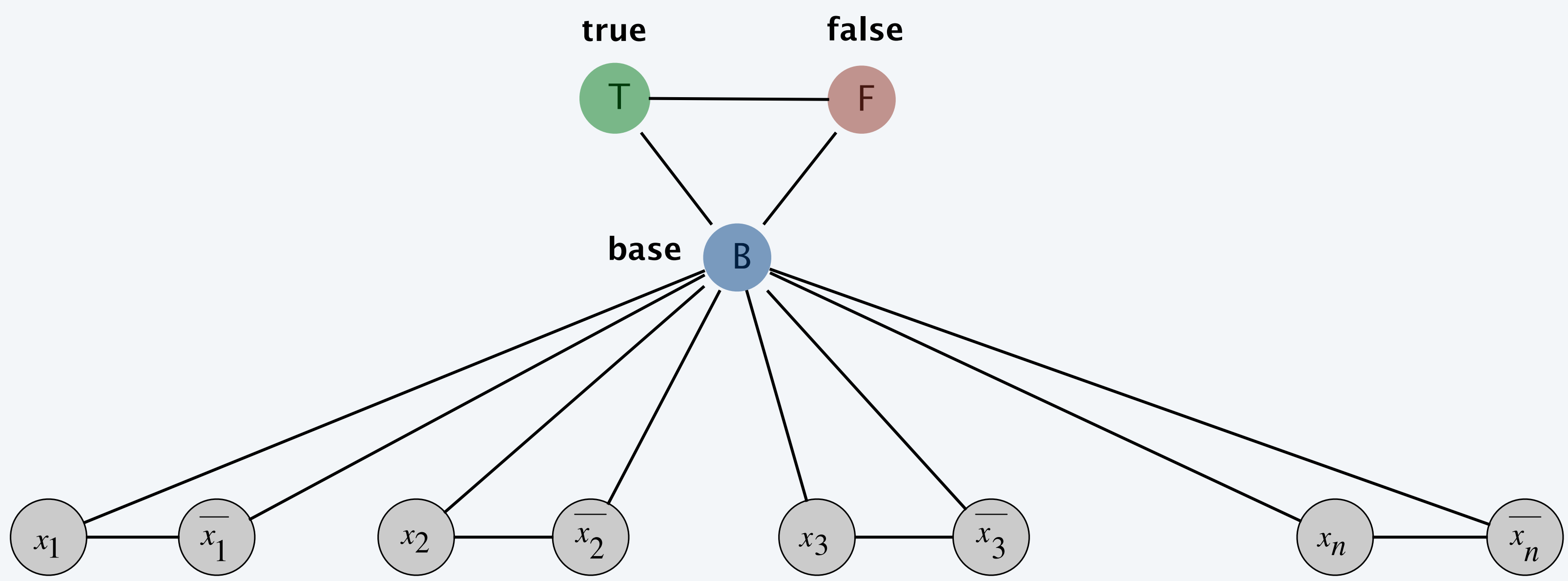
(ii) Connect each literal to its negation.

(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

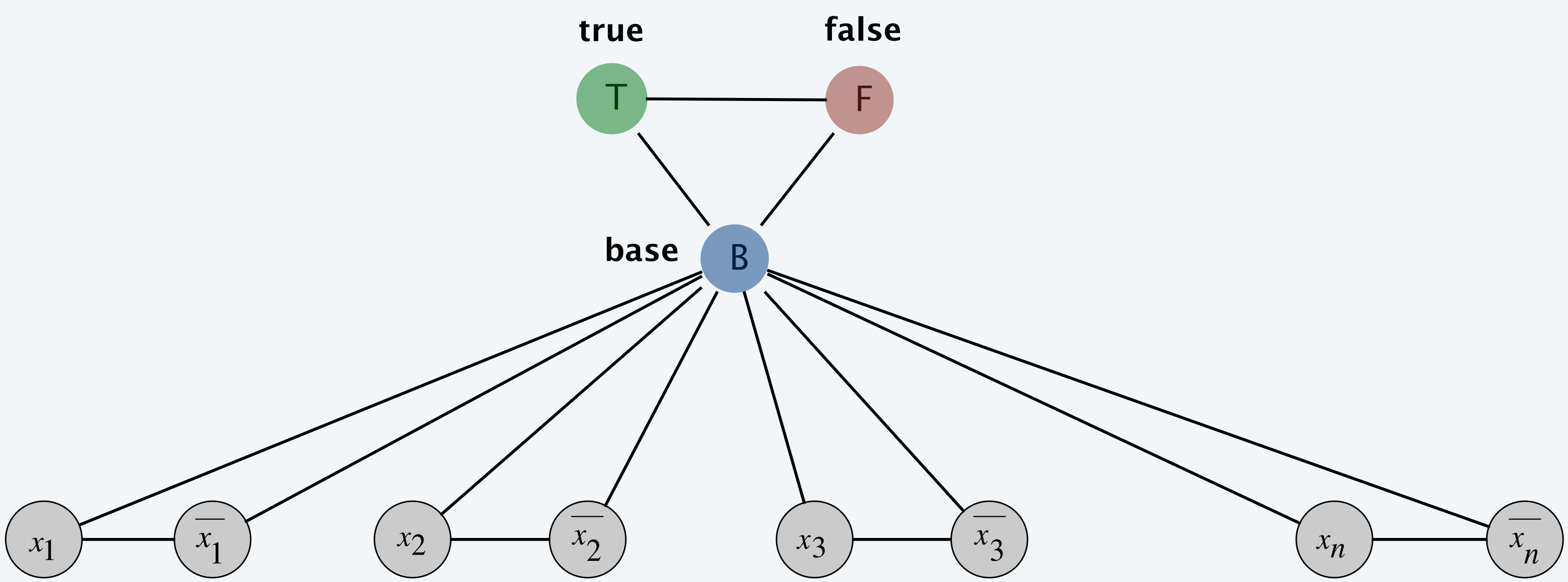


# 3-satisfiability reduces to 3-colorability

Lemma. Graph $G$ is 3-colorable iff $\Phi$ is satisfiable.

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

- Consider assignment that sets all $T$ literals to true.
- (iv) ensures each literal is $T$ or $F$.
- (ii) ensures a literal and its negation are opposites.

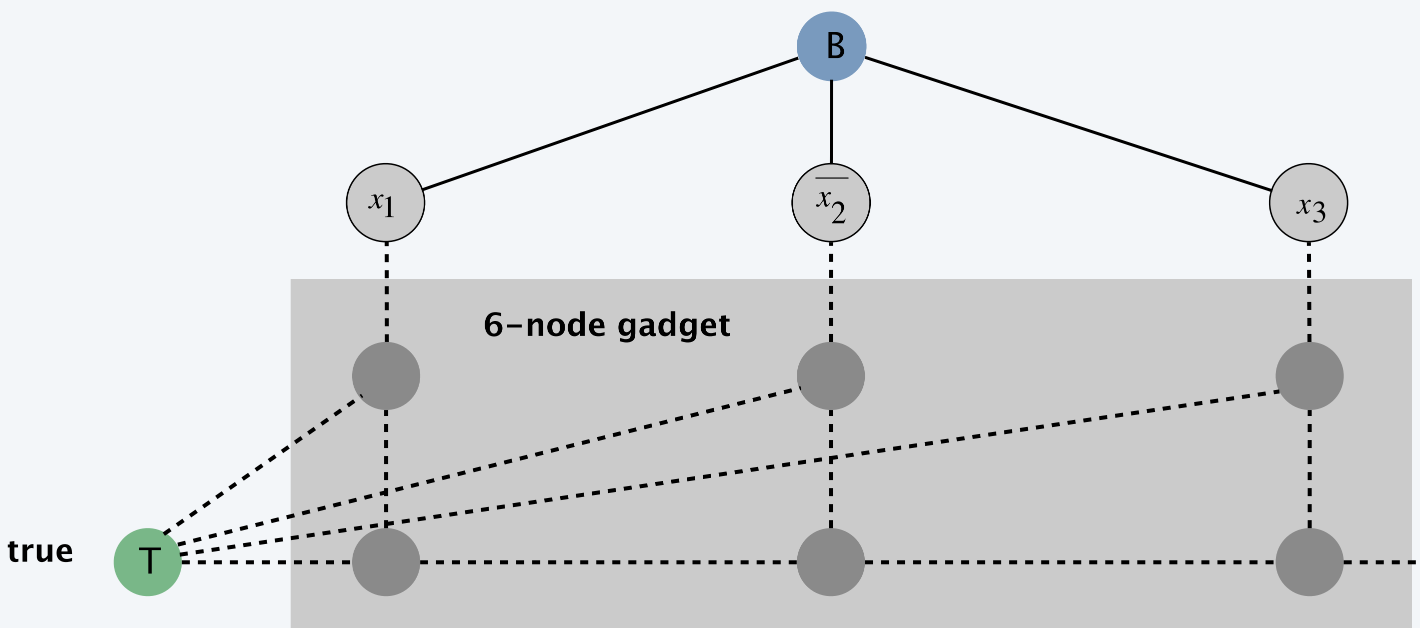


# 3-satisfiability reduces to 3-colorability

Lemma. Graph $G$ is 3-colorable iff $\Phi$ is satisfiable.

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

- Consider assignment that sets all $T$ literals to true.
- (iv) ensures each literal is $T$ or $F$.
- (ii) ensures a literal and its negation are opposites.
- (v) ensures at least one literal in each clause is $T$.



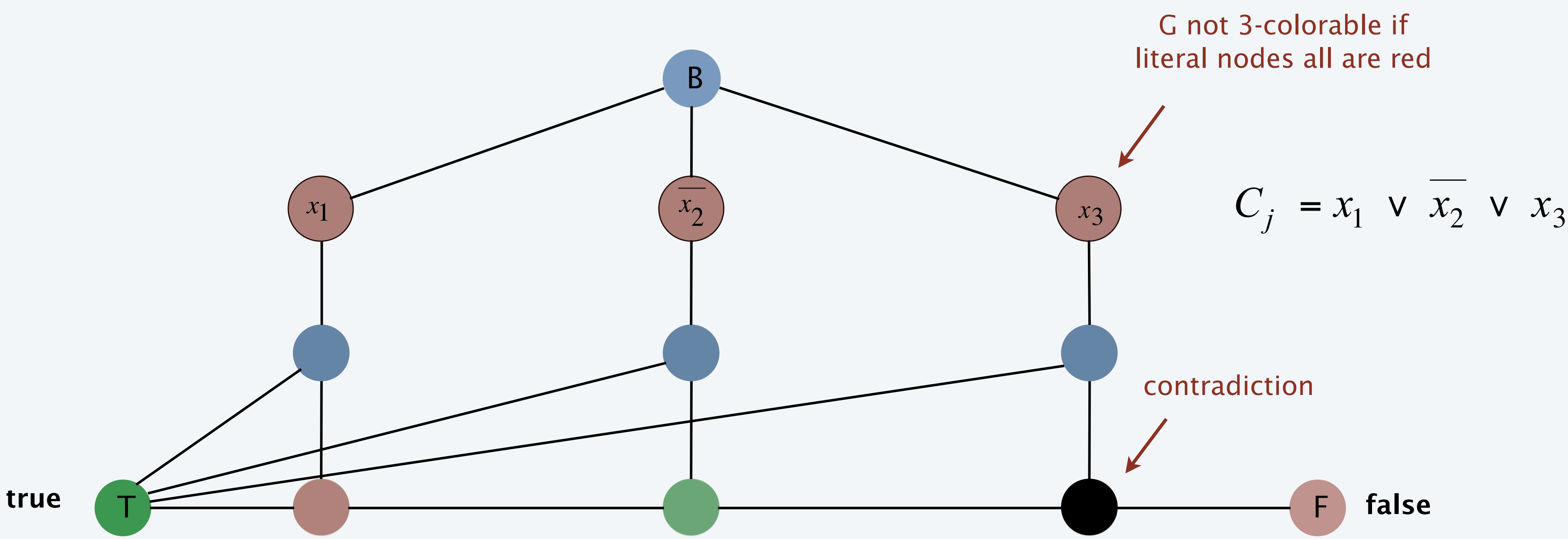

$C_j = x_1 \lor \overline{x_2} \lor x_3$

# 3-satisfiability reduces to 3-colorability

Lemma. Graph $G$ is 3-colorable iff $\Phi$ is satisfiable.

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

- Consider assignment that sets all $T$ literals to true.
- (iv) ensures each literal is $T$ or $F$.
- (ii) ensures a literal and its negation are opposites.
- (v) ensures at least one literal in each clause is $T$.

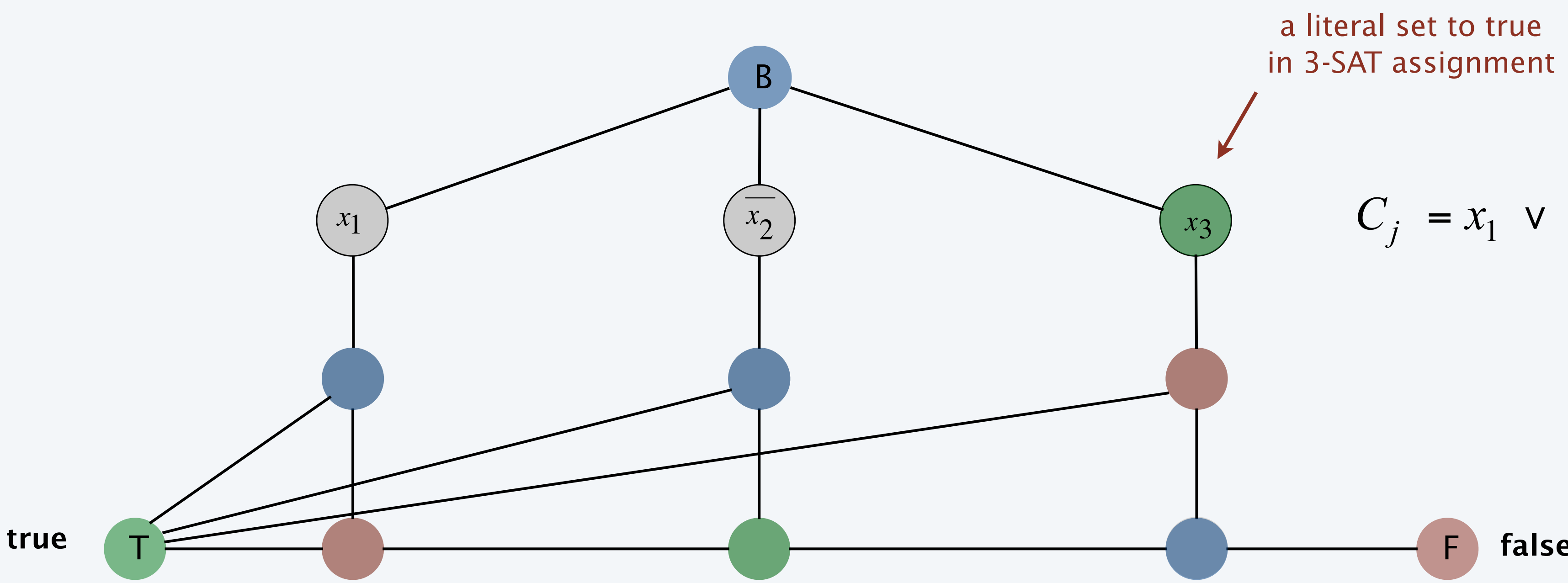


# 3-satisfiability reduces to 3-colorability

Lemma. Graph $G$ is 3-colorable iff $\Phi$ is satisfiable.

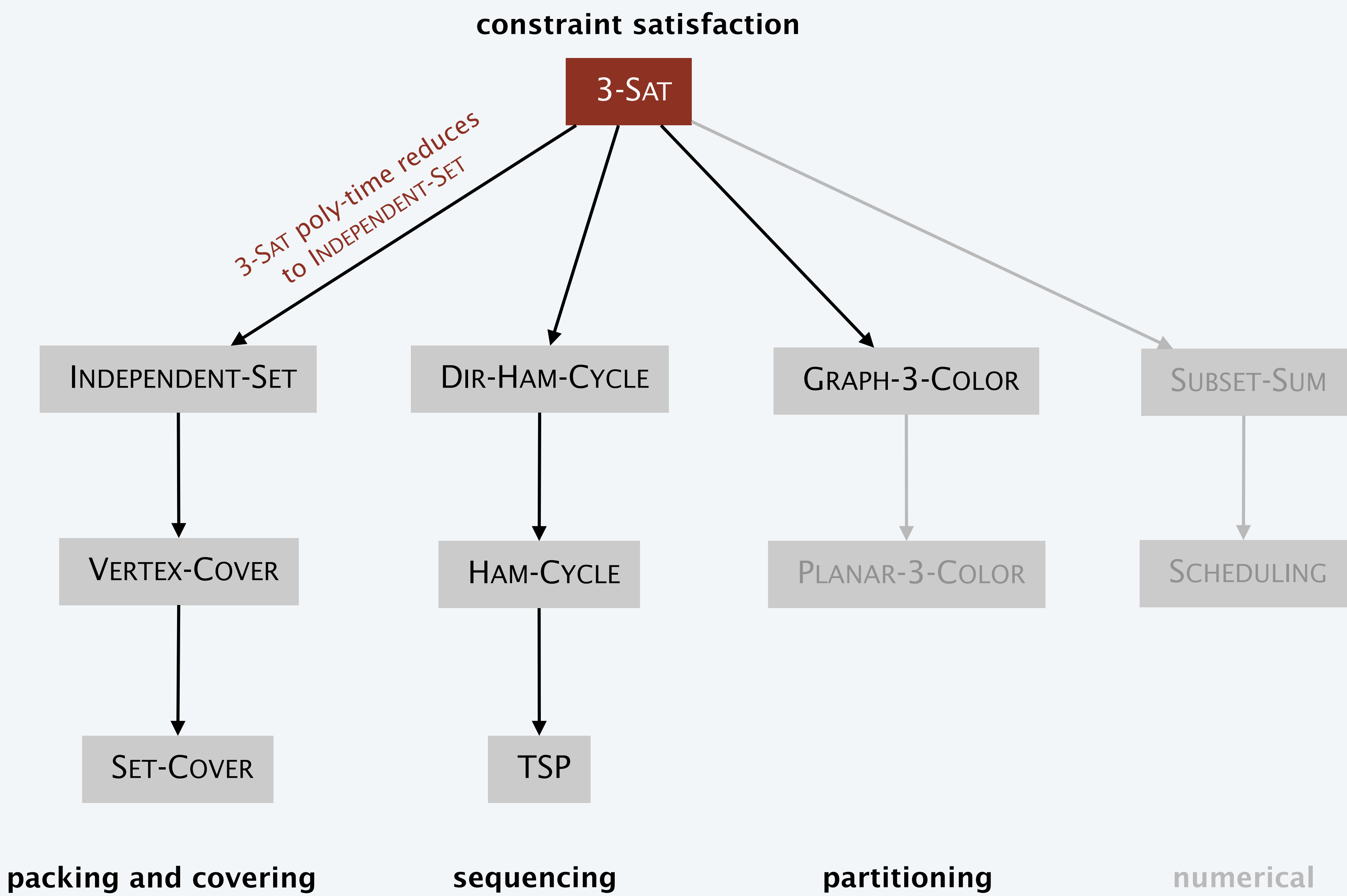
Pf. $\Leftarrow$ Suppose 3-SAT instance $\Phi$ is satisfiable.

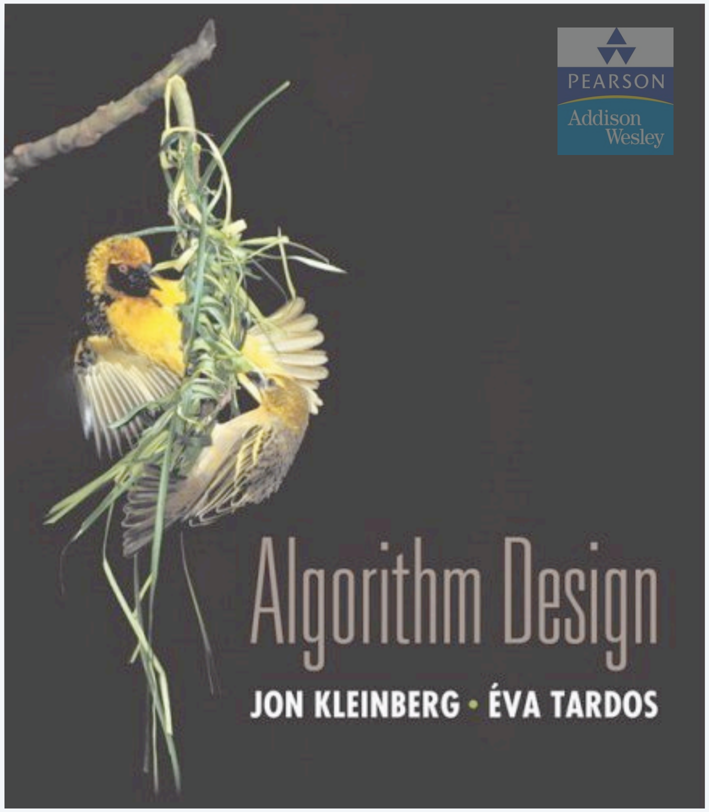
- Color all true literals $T$.
- Color node below green node $F$, and node below that $B$.
- Color remaining middle row nodes $B$.
- Color remaining bottom nodes $T$ or $F$ as forced. ▪




$$C_j = x_1 \lor \overline{x_2} \lor x_3$$

# Polynomial-time reductions






**SECTION 8.8**

# 8. INTRACTABILITY I

- *poly-time reductions*
- *packing and covering problems*
- *constraint satisfaction problems*
- *sequencing problems*
- *partitioning problems*
- *graph coloring*
- *numerical problems*

# Subset sum

SUBSET-SUM. Given natural numbers $w_1, \ldots, w_n$ and an integer $W$, is there a subset that adds up to exactly $W$?

Ex. $\{ 1, 4, 16, 64, 256, 1040, 1041, 1093, 1284, 1344 \}$, $W = 3754$.
Yes. $1 + 16 + 64 + 256 + 1040 + 1093 + 1284 = 3754$.

Remark. With arithmetic problems, input integers are encoded in binary. Poly-time reduction must be polynomial in binary encoding.

# Subset sum

Theorem. 3-SAT $\le_P$ SUBSET-SUM.

Pf. Given an instance $\Phi$ of 3-SAT, we construct an instance of SUBSET-SUM that has solution iff $\Phi$ is satisfiable.

# 3-satisfiability reduces to subset sum

Construction. Given 3-SAT instance $\Phi$ with $n$ variables and $k$ clauses, form $2n + 2k$ decimal integers, each of $n + k$ digits:

- Include one digit for each variable $x_i$ and for each clause $C_j$.
- Include two numbers for each variable $x_i$.
- Include two numbers for each clause $C_j$.
- Sum of each $x_i$ digit is $1$;
  sum of each $C_j$ digit is $4$.

Key property. No carries possible $\Rightarrow$ each digit yields one equation.

$$C_1 = \neg x_1 \lor x_2 \lor x_3$$
$$C_2 = x_1 \lor \neg x_2 \lor x_3$$
$$C_3 = \neg x_1 \lor \neg x_2 \lor \neg x_3$$

3-SAT instance

| | $x_1$ | $x_2$ | $x_3$ | $C_1$ | $C_2$ | $C_3$ | |
|---|---|---|---|---|---|---|---|
| $x_1$ | 1 | 0 | 0 | 0 | 1 | 0 | 100,010 |
| $\neg x_1$ | 1 | 0 | 0 | 1 | 0 | 1 | 100,101 |
| $x_2$ | 0 | 1 | 0 | 1 | 0 | 0 | 10,100 |
| $\neg x_2$ | 0 | 1 | 0 | 0 | 1 | 1 | 10,011 |
| $x_3$ | 0 | 0 | 1 | 1 | 1 | 0 | 1,110 |
| $\neg x_3$ | 0 | 0 | 1 | 0 | 0 | 1 | 1,001 |
| | 0 | 0 | 0 | 1 | 0 | 0 | 100 |
| | 0 | 0 | 0 | 2 | 0 | 0 | 200 |
| | 0 | 0 | 0 | 0 | 1 | 0 | 10 |
| | 0 | 0 | 0 | 0 | 2 | 0 | 20 |
| | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| | 0 | 0 | 0 | 0 | 0 | 2 | 2 |
| $W$ | 1 | 1 | 1 | 4 | 4 | 4 | 111,444 |

SUBSET-SUM instance

# 3-satisfiability reduces to subset sum

**Lemma.** $\Phi$ is satisfiable iff there exists a subset that sums to $W$.

**Pf.** $\Rightarrow$ Suppose $\Phi$ is satisfiable.

- Choose integers corresponding to each *true* literal.
- Since $\Phi$ is satisfiable, each $C_j$ digit sums to at least $1$ from $x_i$ rows.
- Choose dummy integers to make clause digits sum to $4$.

| | $x_1$ | $x_2$ | $x_3$ | $C_1$ | $C_2$ | $C_3$ | |
|---|---|---|---|---|---|---|---|
| $x_1$ | 1 | 0 | 0 | 0 | 1 | 0 | 100,010 |
| $\neg x_1$ | 1 | 0 | 0 | 1 | 0 | 1 | 100,101 |
| $x_2$ | 0 | 1 | 0 | 1 | 0 | 0 | 10,100 |
| $\neg x_2$ | 0 | 1 | 0 | 0 | 1 | 1 | 10,011 |
| $x_3$ | 0 | 0 | 1 | 1 | 1 | 0 | 1,110 |
| $\neg x_3$ | 0 | 0 | 1 | 0 | 0 | 1 | 1,001 |
| dummies to get clause columns to sum to 4 | 0 | 0 | 0 | 1 | 0 | 0 | 100 |
| | 0 | 0 | 0 | 2 | 0 | 0 | 200 |
| | 0 | 0 | 0 | 0 | 1 | 0 | 10 |
| | 0 | 0 | 0 | 0 | 2 | 0 | 20 |
| | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| | 0 | 0 | 0 | 0 | 0 | 2 | 2 |
| $W$ | 1 | 1 | 1 | 4 | 4 | 4 | 111,444 |

**SUBSET-SUM instance**

$$C_1 = \neg x_1 \lor x_2 \lor x_3$$
$$C_2 = x_1 \lor \neg x_2 \lor x_3$$
$$C_3 = \neg x_1 \lor \neg x_2 \lor \neg x_3$$

**3-SAT instance**

# 3-satisfiability reduces to subset sum

**Lemma.** $\Phi$ is satisfiable iff there exists a subset that sums to $W$.

**Pf.** $\Leftarrow$ Suppose there is a subset that sums to $W$.

- Digit $x_i$ forces subset to select either row $x_i$ or $\neg x_i$ (but not both).
- Digit $C_j$ forces subset to select at least one literal in clause.
- Assign $x_i = true$ iff row $x_i$ selected. ▪

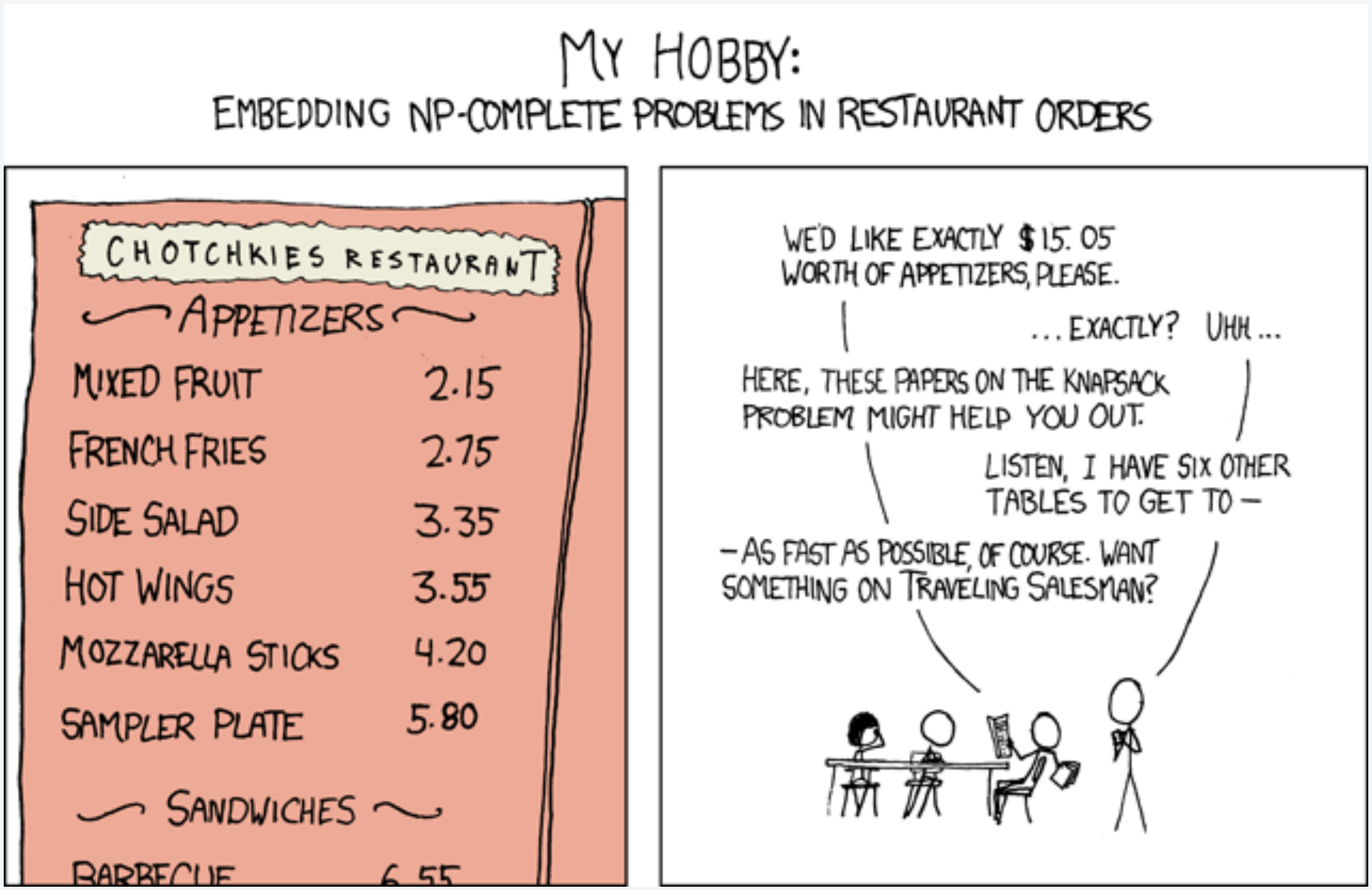
$$C_1 = \neg x_1 \vee x_2 \vee x_3$$
$$C_2 = x_1 \vee \neg x_2 \vee x_3$$
$$C_3 = \neg x_1 \vee \neg x_2 \vee \neg x_3$$

3-SAT instance

| | $x_1$ | $x_2$ | $x_3$ | $C_1$ | $C_2$ | $C_3$ | |
|---|---|---|---|---|---|---|---|
| $x_1$ | 1 | 0 | 0 | 0 | 1 | 0 | 100,010 |
| $\neg x_1$ | 1 | 0 | 0 | 1 | 0 | 1 | 100,101 |
| $x_2$ | 0 | 1 | 0 | 1 | 0 | 0 | 10,100 |
| $\neg x_2$ | 0 | 1 | 0 | 0 | 1 | 1 | 10,011 |
| $x_3$ | 0 | 0 | 1 | 1 | 1 | 0 | 1,110 |
| $\neg x_3$ | 0 | 0 | 1 | 0 | 0 | 1 | 1,001 |
| dummies to get clause columns to sum to 4 | 0 | 0 | 0 | 1 | 0 | 0 | 100 |
| | 0 | 0 | 0 | 2 | 0 | 0 | 200 |
| | 0 | 0 | 0 | 0 | 1 | 0 | 10 |
| | 0 | 0 | 0 | 0 | 2 | 0 | 20 |
| | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| | 0 | 0 | 0 | 0 | 0 | 2 | 2 |
| $W$ | 1 | 1 | 1 | 4 | 4 | 4 | 111,444 |

SUBSET-SUM instance

# My hobby




**Randall Munro**
**http://xkcd.com/c287.html**

# Partition

**SUBSET-SUM.** Given natural numbers $w_1, \ldots, w_n$ and an integer $W$, is there a subset that adds up to exactly $W$?

**PARTITION.** Given natural numbers $v_1, \ldots, v_m$, can they be partitioned into two subsets that add up to the same value $\frac{1}{2} \Sigma_i v_i$?

**Theorem.** SUBSET-SUM $\leq_P$ PARTITION.

**Pf.** Let $W, w_1, \ldots, w_n$ be an instance of SUBSET-SUM.

- Create instance of PARTITION with $m = n + 2$ elements.
  - $v_1 = w_1, v_2 = w_2, \ldots, v_n = w_n, \quad v_{n+1} = 2 \Sigma_i w_i - W, \quad v_{n+2} = \Sigma_i w_i + W$
- Lemma: there exists a subset that sums to $W$ iff there exists a partition since elements $v_{n+1}$ and $v_{n+2}$ cannot be in the same partition. ▪

## Scheduling with release times

SCHEDULE. Given a set of $n$ jobs with processing time $t_j$, release time $r_j$, and deadline $d_j$, is it possible to schedule all jobs on a single machine such that job $j$ is processed with a contiguous slot of $t_j$ time units in the interval $[r_j, d_j]$?

Ex.

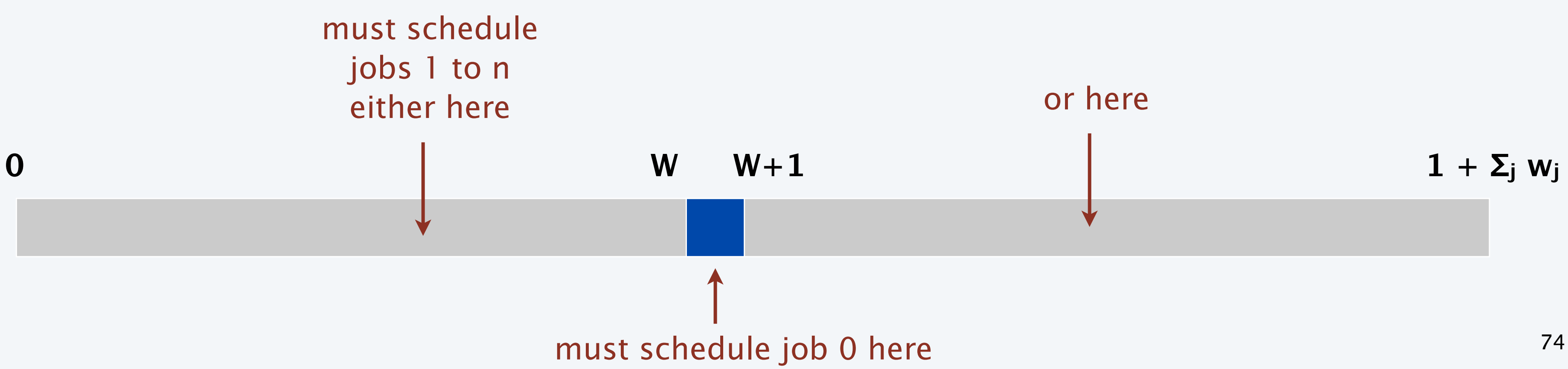
# Scheduling with release times

**Theorem.** SUBSET-SUM $\leq_P$ SCHEDULE.

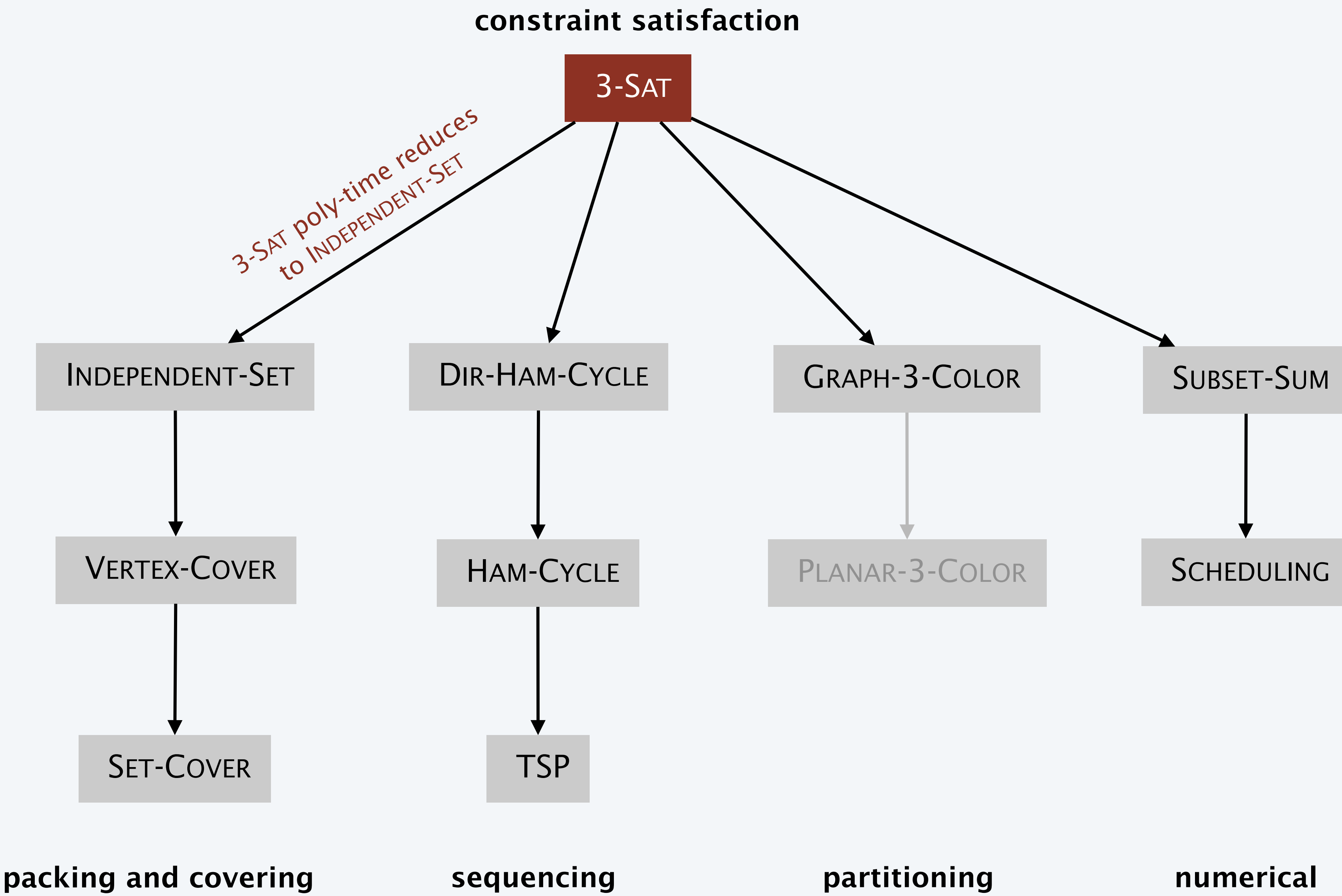
**Pf.** Given SUBSET-SUM instance $w_1, \ldots, w_n$ and target $W$, construct an instance of SCHEDULE that is feasible iff there exists a subset that sums to exactly $W$.

**Construction.**

- Create $n$ jobs with processing time $t_j = w_j$, release time $r_j = 0$, and no deadline ($d_j = 1 + \Sigma_j w_j$).
- Create job $0$ with $t_0 = 1$, release time $r_0 = W$, and deadline $d_0 = W + 1$.
- Lemma: subset that sums to $W$ iff there exists a feasible schedule. ▪



# Polynomial-time reductions



# Karp's 21 NP-complete problems

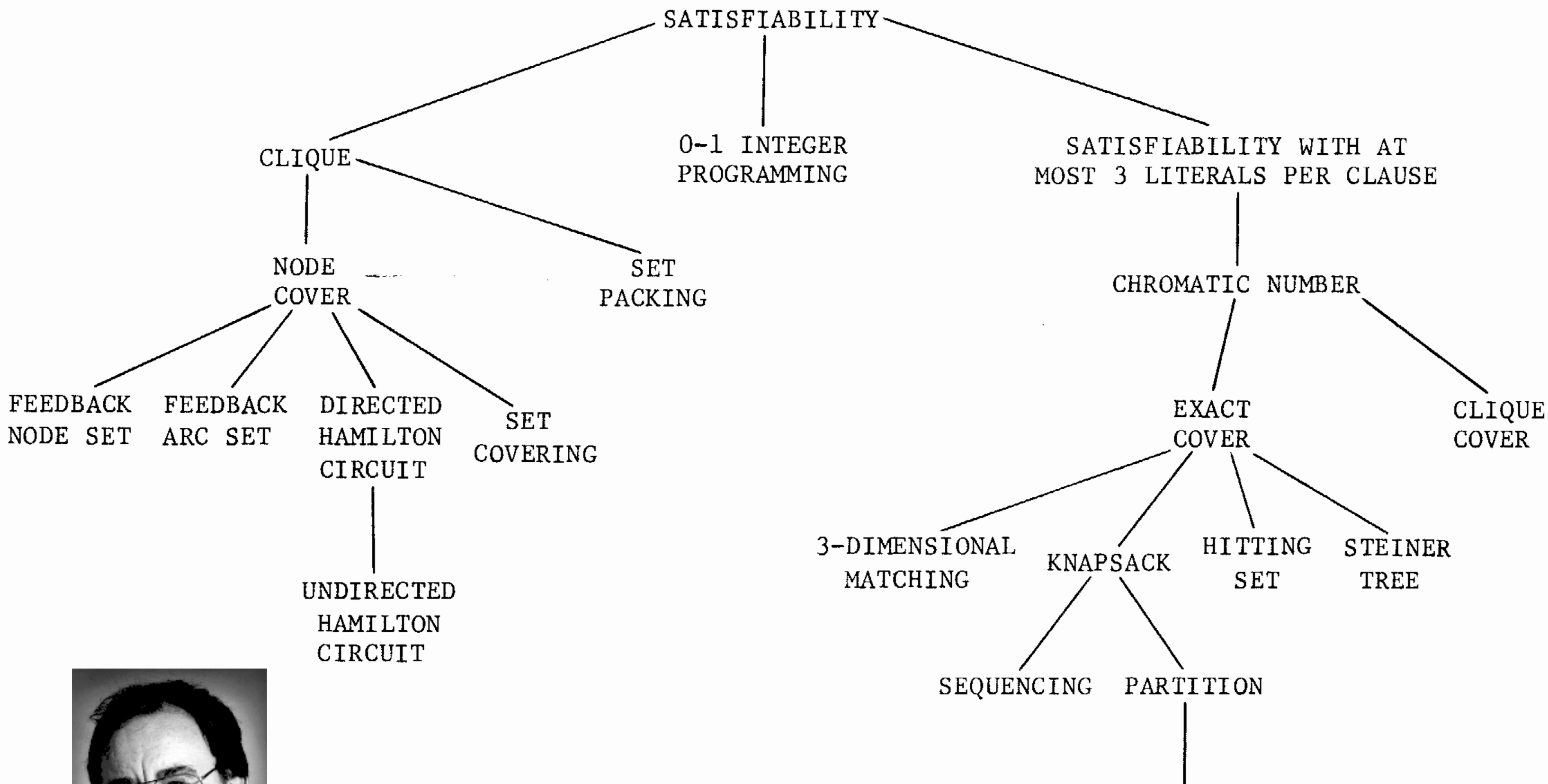
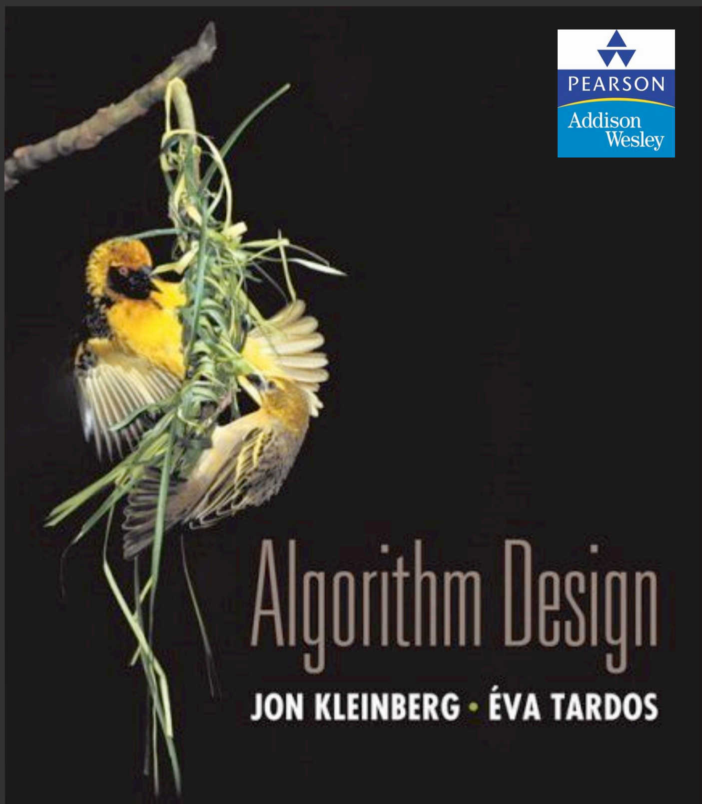



FIGURE 1 - Complete Problems

96

RICHARD M. KARP

**Dick Karp (1972)**
**1985 Turing Award**




# 8. INTRACTABILITY II

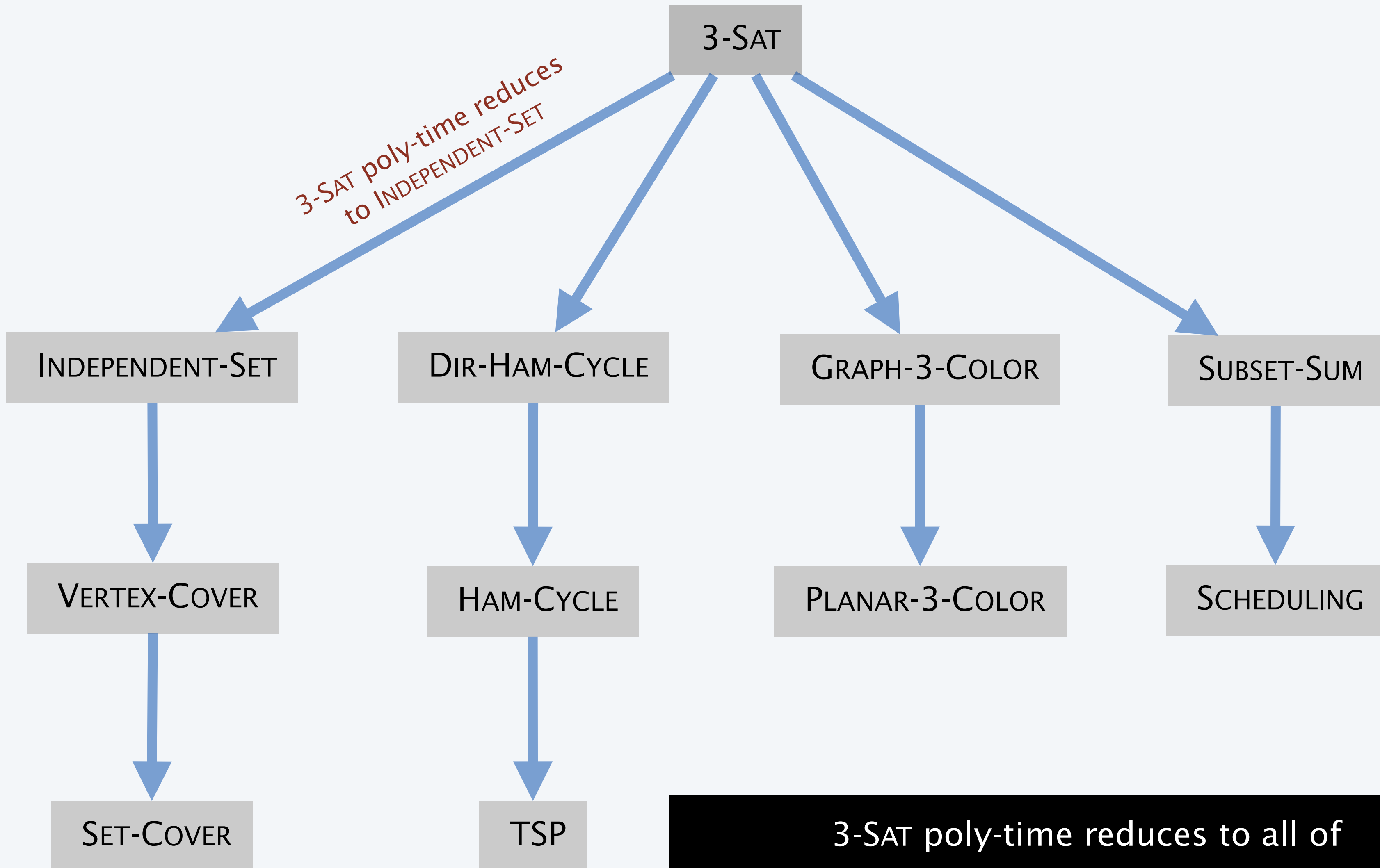
- *P vs. NP*
- *NP-complete*
- *co-NP*
- *NP-hard*

Lecture slides by Kevin Wayne

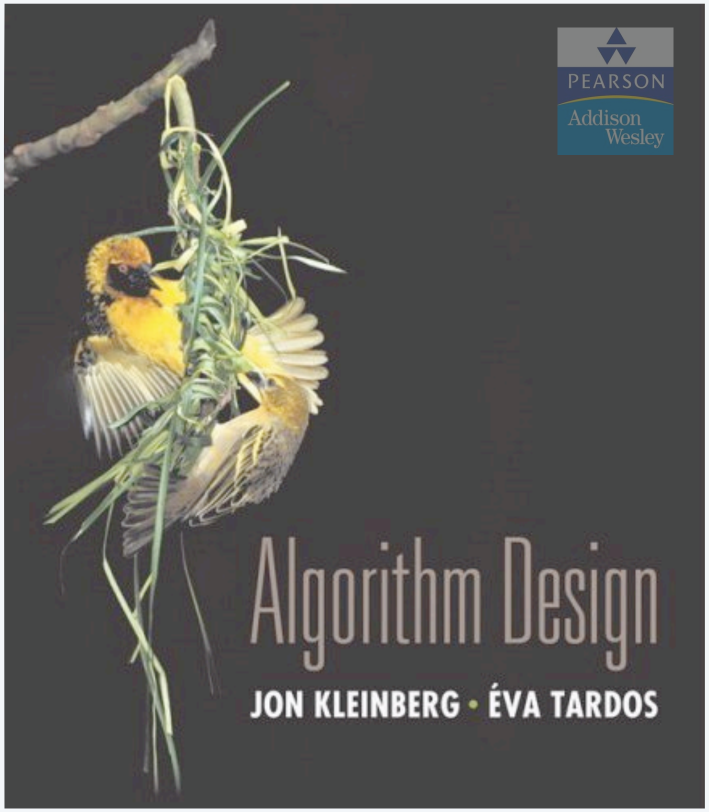
http://www.cs.princeton.edu/~wayne/kleinberg-tardos

Last updated on Sep 8, 2013 6:41 AM

# Recap




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




SECTION 8.3

# 8. INTRACTABILITY II

- *P vs. NP*
- *NP-complete*
- *co-NP*
- *NP-hard*

# Decision problems

Decision problem.

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

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

length of s

Ex.

- Problem PRIMES = $\{ 2, 3, 5, 7, 11, 13, 17, 23, 29, 31, 37, \ldots. \}$.
- Instance $s = 592335744548702854681$.
- AKS algorithm PRIMES in $O(|s|^8)$ steps.

# Definition of P

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

| Problem | Description | Algorithm | yes | no |
|---|---|---|---|---|
| MULTIPLE | Is $x$ a multiple of $y$ ? | grade-school division | 51, 17 | 51, 16 |
| REL-PRIME | Are $x$ and $y$ relatively prime ? | Euclid (300 BCE) | 34, 39 | 34, 51 |
| PRIMES | Is $x$ prime ? | AKS (2002) | 53 | 51 |
| EDIT-DISTANCE | Is the edit distance between $x$ and $y$ less than 5 ? | dynamic programming | niether neither | acgggt ttttta |
| L-SOLVE | Is there a vector $x$ that satisfies $Ax = b$ ? | Gauss-Edmonds elimination | $\begin{bmatrix} 0 & 1 & 1 \\ 2 & 4 & -2 \\ 0 & 3 & 15 \end{bmatrix}, \begin{bmatrix} 4 \\ 2 \\ 36 \end{bmatrix}$ | $\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$ |
| ST-CONN | Is there a path between $s$ and $t$ in a graph $G$ ? | depth-first search (Theseus) |  |  |

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

"certificate" or "witness"

Def. **NP** is the set of 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 $p(\cdot)$

Remark. **NP** stands for nondeterministic polynomial time.

## Certifiers and certificates: composite

COMPOSITES. Given an integer $s$, is $s$ composite?

Certificate. A nontrivial factor $t$ of $s$. Such a certificate exists iff $s$ is composite. Moreover $|t| \leq |s|$.

Certifier. Check that $1 < t < s$ and that $s$ is a multiple of $t$.

| | |
|---|---|
| **instance s** | 437669 |
| **certificate t** | 541 or 809 |

← 437,669 = 541 × 809

Conclusion. COMPOSITES ∈ **NP**.

# Certifiers and certificates: 3-satisfiability

3-SAT. Given a CNF formula $\Phi$, is there a satisfying assignment?

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

Certifier. Check that each clause in $\Phi$ has at least one true literal.

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

**certificate t** $x_1 = true,\ x_2 = true,\ x_3 = false,\ x_4 = false$

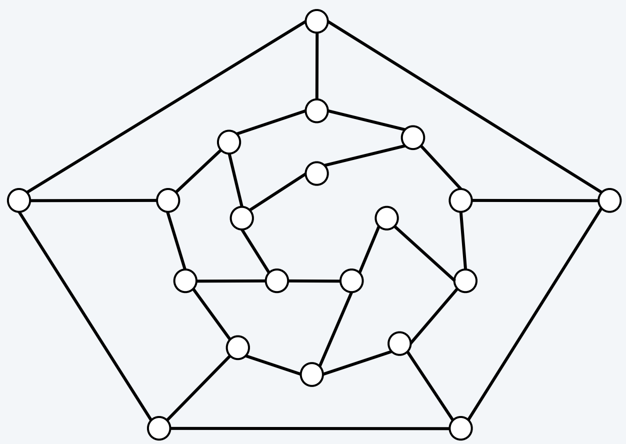
Conclusion. 3-SAT $\in$ **NP**.

# Certifiers and certificates: Hamilton path

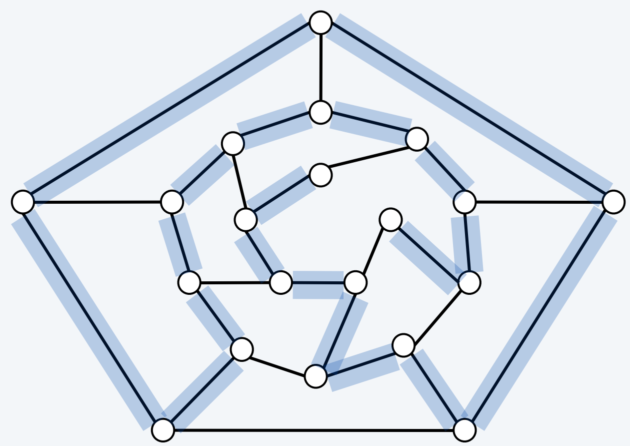
HAM-PATH. Given an undirected graph $G = (V, E)$, does there exist a simple path $P$ that visits every node?

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.



instance s

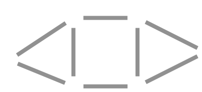


certificate t

Conclusion. HAM-PATH $\in$ **NP**.

# Definition of NP

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

| Problem | Description | Algorithm | yes | no |
|---|---|---|---|---|
| L-SOLVE | Is there a vector $x$ that satisfies $Ax = b$ ? | Gauss-Edmonds elimination | $\begin{bmatrix} 0 & 1 & 1 \\ 2 & 4 & -2 \\ 0 & 3 & 15 \end{bmatrix}, \begin{bmatrix} 4 \\ 2 \\ 36 \end{bmatrix}$ | $\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$ |
| COMPOSITES | Is $x$ composite ? | AKS (2002) | 51 | 53 |
| FACTOR | Does $x$ have a nontrivial factor less than $y$ ? | ? | (56159, 50) | (55687, 50) |
| SAT | Is there a truth assignment that satisfies the formula ? | ? | $\neg x_1 \lor x_2$<br>$x_1 \lor x_2$ | $\neg x_2$<br>$\neg x_1 \lor x_2$<br>$x_1 \lor x_2$ |
| 3-COLOR | Can the nodes of a graph $G$ be colored with 3 colors? | ? |  |  |
| HAM-PATH | Is there a simple path between $s$ and $t$ that visits every node? | ? |  |  |

# Definition of NP

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

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

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

Claim. **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 $C(s, t) = A(s)$. ▪

Claim. **NP** $\subseteq$ **EXP**.

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

- By definition, there exists a poly-time certifier $C(s, t)$ for $X$.
- To solve input $s$, run $C(s, t)$ on all strings $t$ with $|t| \le p(|s|)$.
- Return $yes$ if $C(s, t)$ returns $yes$ for any of these potential certificates. ▪

Remark. Time-hierarchy theorem implies **P** $\subsetneq$ **EXP**.

# The main question: P vs. NP

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

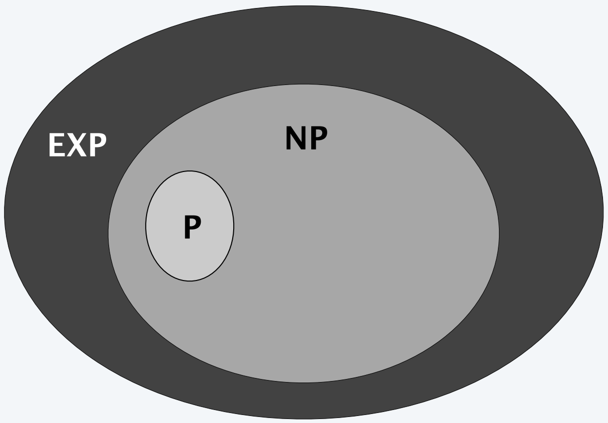
A. Exhaustive search: try all $2^n$ truth assignments.

Q. Can we do anything substantially more clever?

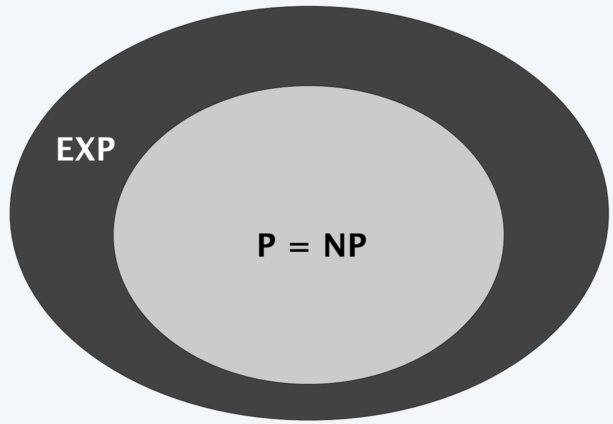
Conjecture. $\underbrace{\text{No poly-time algorithm}}_{\text{"intractable"}}$ for 3-SAT.

# 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, 3-COLOR, FACTOR, ...
If no. No efficient algorithms possible for 3-SAT, TSP, 3-COLOR, ...

Consensus opinion. Probably no.

# Possible outcomes

$P \neq NP$.

> *" 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*

# Possible outcomes

P ≠ NP.

> *" 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*

> *" 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*

# Possible outcomes

P = NP.

> *" P = NP. 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.

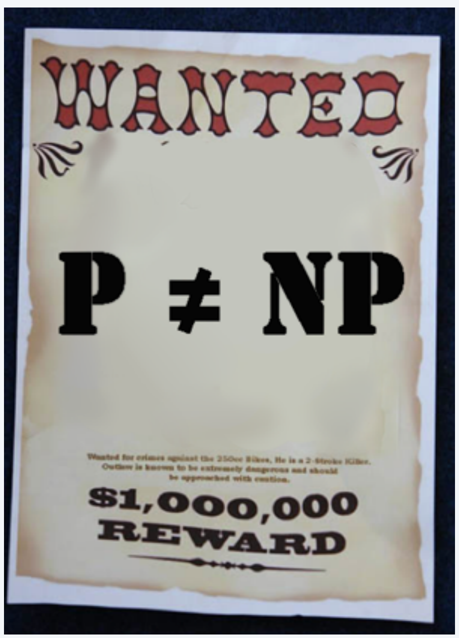
**P** ≠ **NP**, but with $O(n^{\log^* n})$ algorithm for 3-SAT.

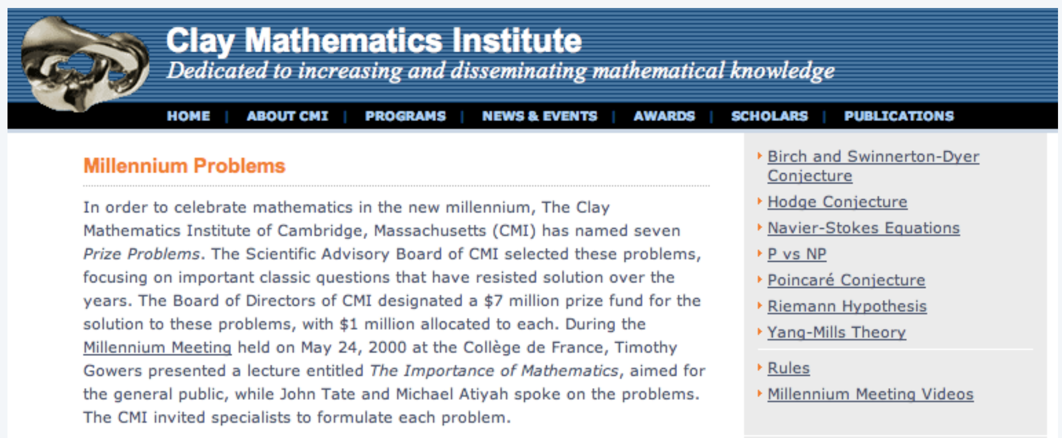
**P** = **NP** is independent (of ZFC axiomatic set theory).

> *" 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 will 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.



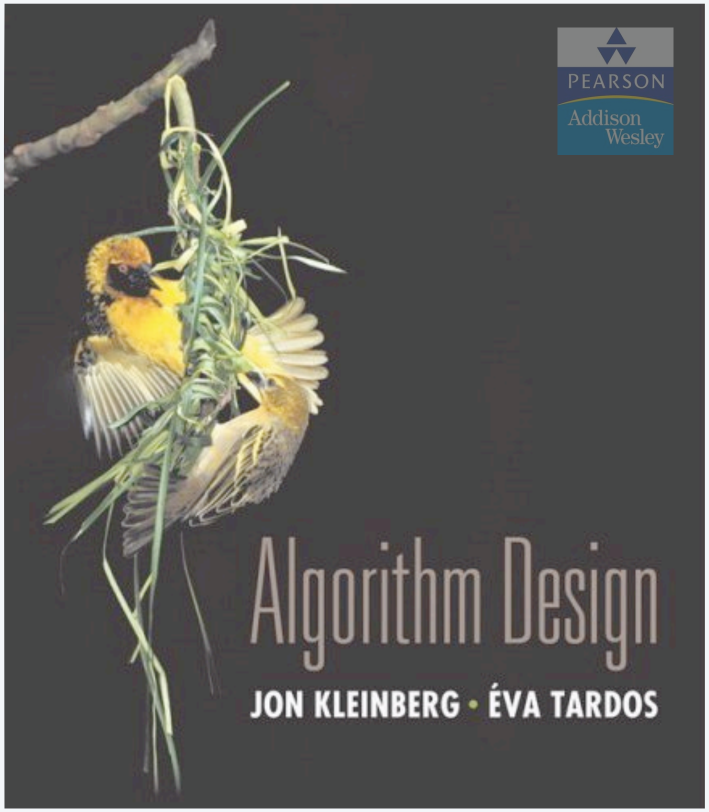



# Looking for a job?

Some writers for the Simpsons and Futurama.

- J. Steward Burns. *M.S. in mathematics* (*Berkeley '93*).
- David X. Cohen. *M.S. in computer science* (*Berkeley '92*).
- Al Jean. *B.S. in mathematics*. (*Harvard '81*).
- Ken Keeler. *Ph.D. in applied mathematics* (*Harvard '90*).
- Jeff Westbrook. *Ph.D. in computer science* (*Princeton '89*).




SECTION 8.4

# 8. INTRACTABILITY II

- *P vs. NP*
- ***NP-complete***
- *co-NP*
- *NP-hard*

# Polynomial transformation

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 input $x$ to $X$, we can construct an input $y$ 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**, $X \leq_p Y$.

Theorem. 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**.

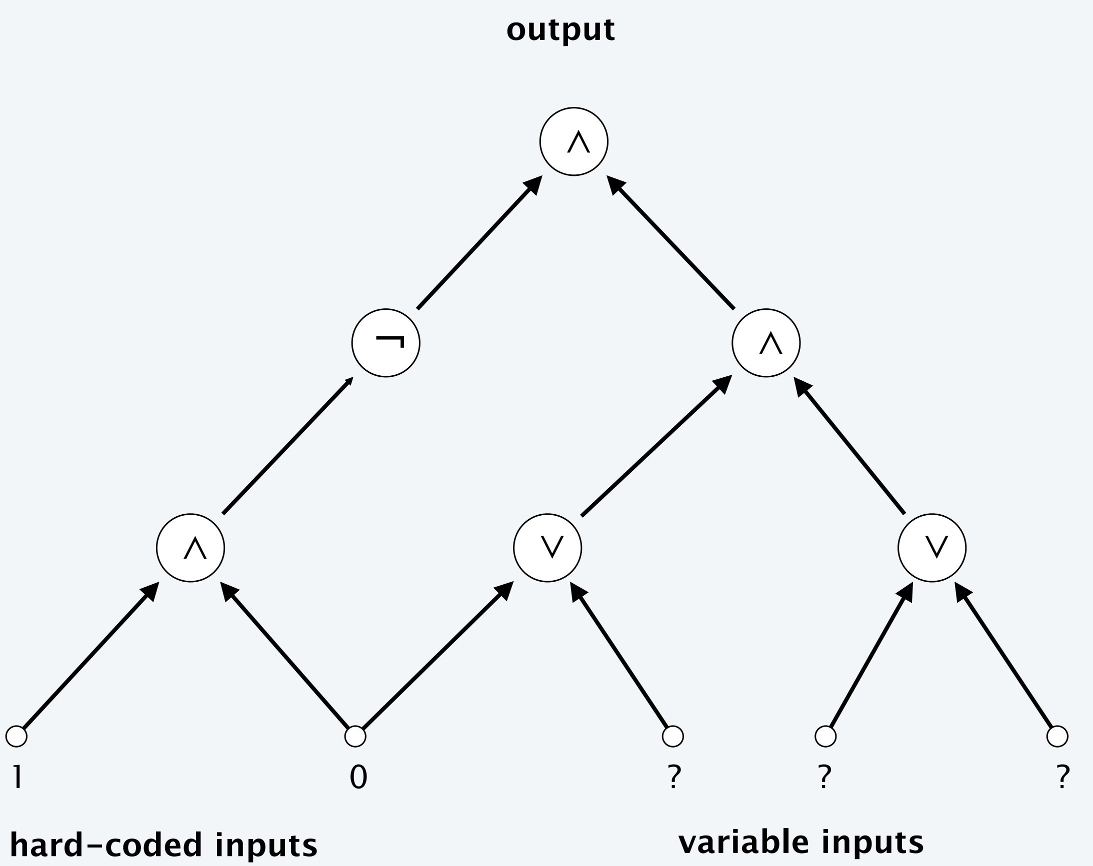
- Consider any problem $X \in$ **NP**. Since $X \leq_p Y$, we have $X \in$ **P**.
- This implies **NP** $\subseteq$ **P**.
- We already know **P** $\subseteq$ **NP**. Thus **P** = **NP**. ▪

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

# Circuit satisfiability

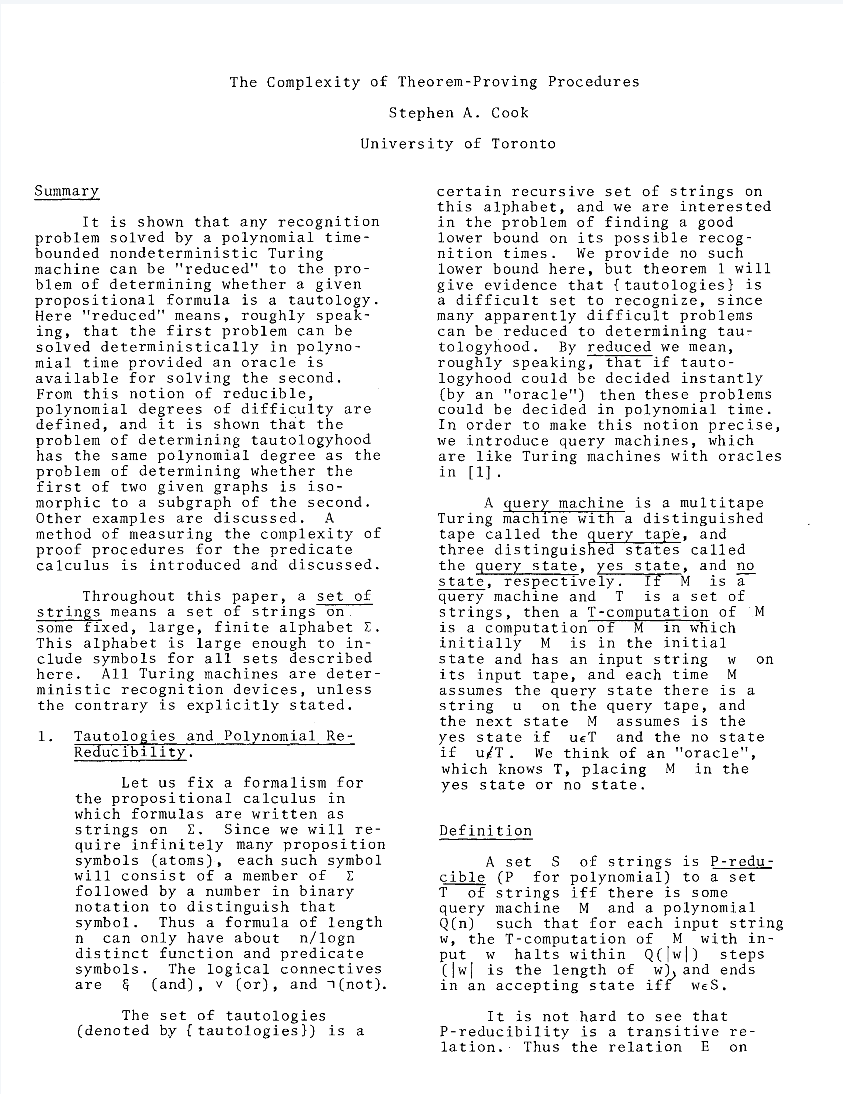
CIRCUIT-SAT. Given a combinational circuit built from AND, OR, and NOT gates, is there a way to set the circuit inputs so that the output is 1?

yes: 1 0 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 Re-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/logn distinct function and predicate symbols. The logical connectives are & (and), v (or), and ¬(not).

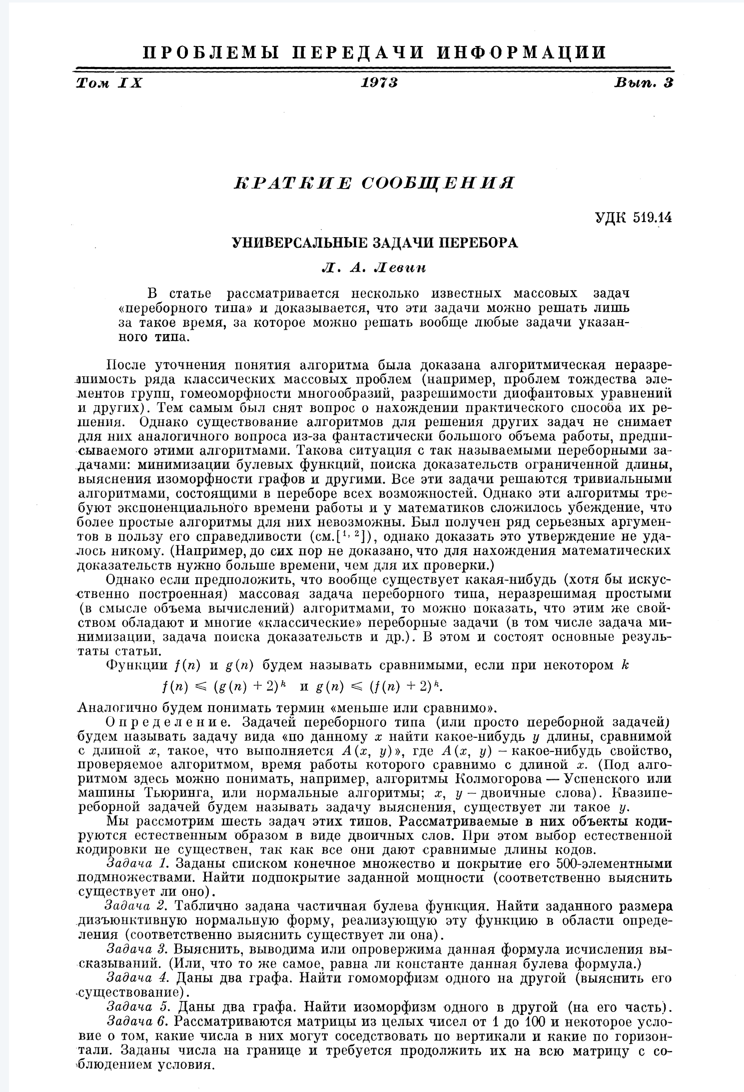
The set of tautologies (denoted by {tautologies}) is a certain recursive set of strings on this alphabet, and we are interested in the problem of finding a good lower bound on its possible recognition times. We provide no such lower bound here, but theorem 1 will give evidence that {tautologies} is a 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 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 состоят 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). Квазипереборной 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 $\in$ **NP**-complete.

Pf sketch.

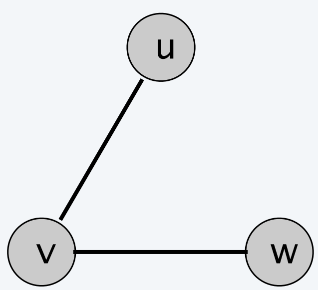
- Clearly, CIRCUIT-SAT $\in$ **NP**.
- Any algorithm that takes a fixed number of bits $n$ as input and produces a *yes* or *no* answer can be represented by such a circuit.
- Moreover, if algorithm takes poly-time, then circuit is of poly-size.

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

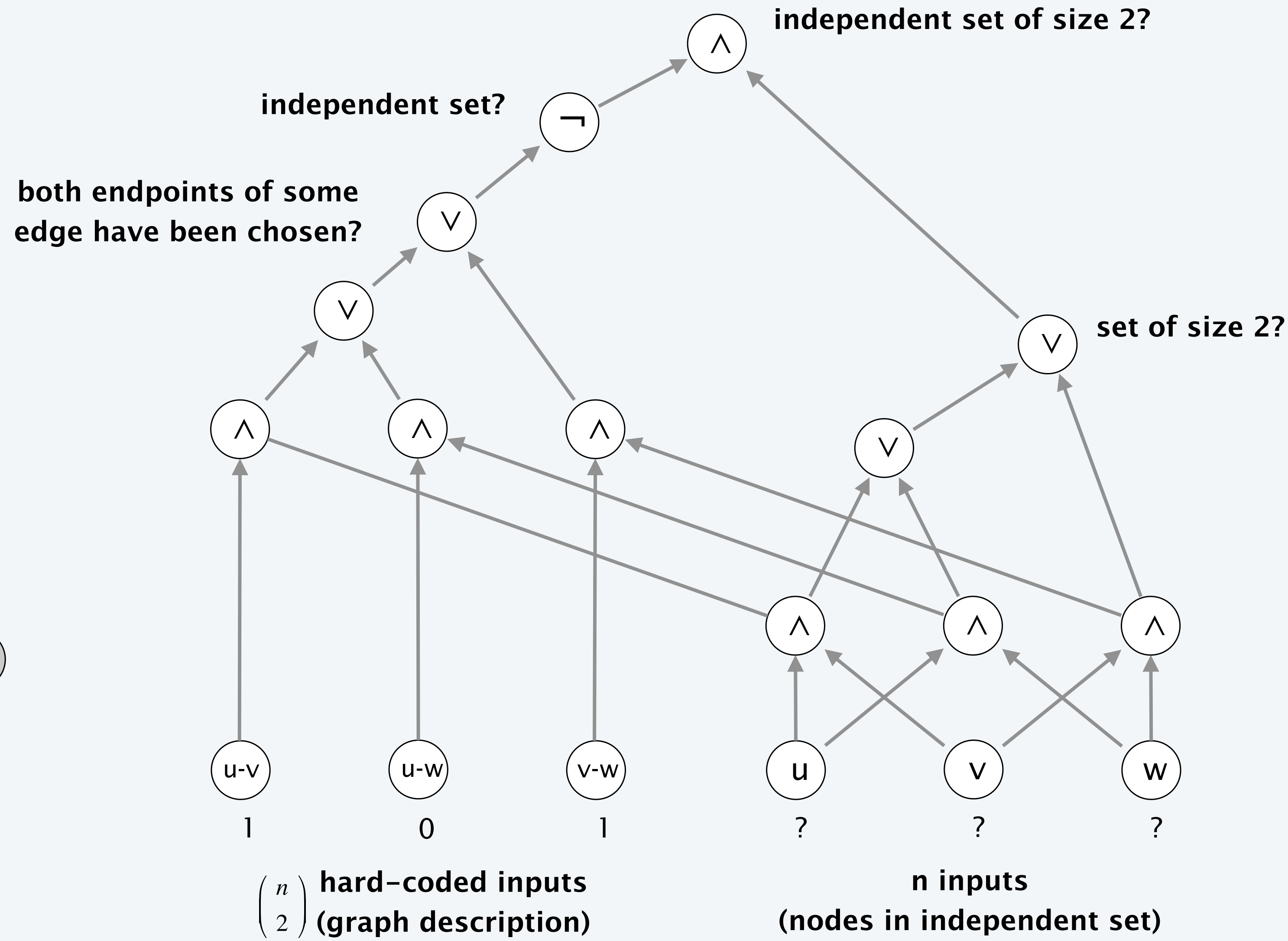
- Consider any problem $X \in$ **NP**. It has a poly-time certifier $C(s, t)$: $s \in X$ iff there exists a certificate $t$ of length $p(|s|)$ such that $C(s, t) = yes$.
- View $C(s, t)$ as an algorithm with $|s| + p(|s|)$ input bits and convert it into a poly-size circuit $K$.
  - first $|s|$ bits are 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**.
- Step 2. Choose an **NP**-complete problem $X$.
- Step 3. Prove that $X \leq_p Y$.

Theorem. 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 $\underbrace{W \leq_p X}_{\text{by definition of NP-complete}}$ and $\underbrace{X \leq_p Y}_{\text{by assumption}}$.

- By transitivity, $W \leq_p Y$.
- Hence $Y \in$ **NP**-complete. ▪

# 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 $\Phi$ 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.

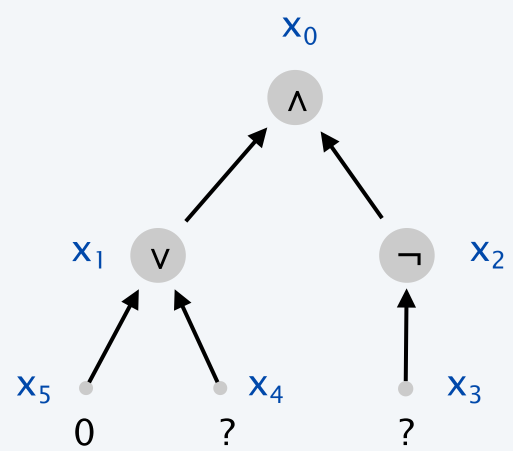
Step 1. Create a 3-SAT variable $x_i$ for each circuit element $i$.

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

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

- $x_5 = 0$ $\Rightarrow$ add 1 clause: $\overline{x_5}$
- $x_0 = 1$ $\Rightarrow$ 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 $z_1 = z_2 = \mathit{false}$:

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

- 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

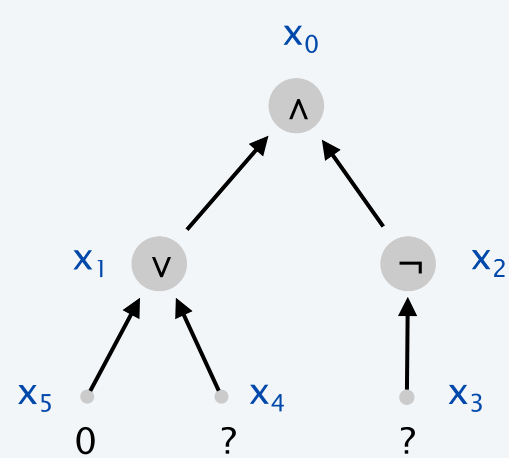
Lemma. $\Phi$ is satisfiable iff the inputs of $K$ can be set so that it outputs $1$.

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

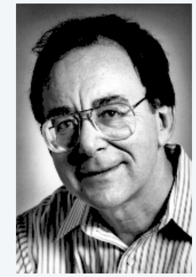
- Can propagate input values to create values at all nodes of $K$.
- This set of values satisfies $\Phi$.

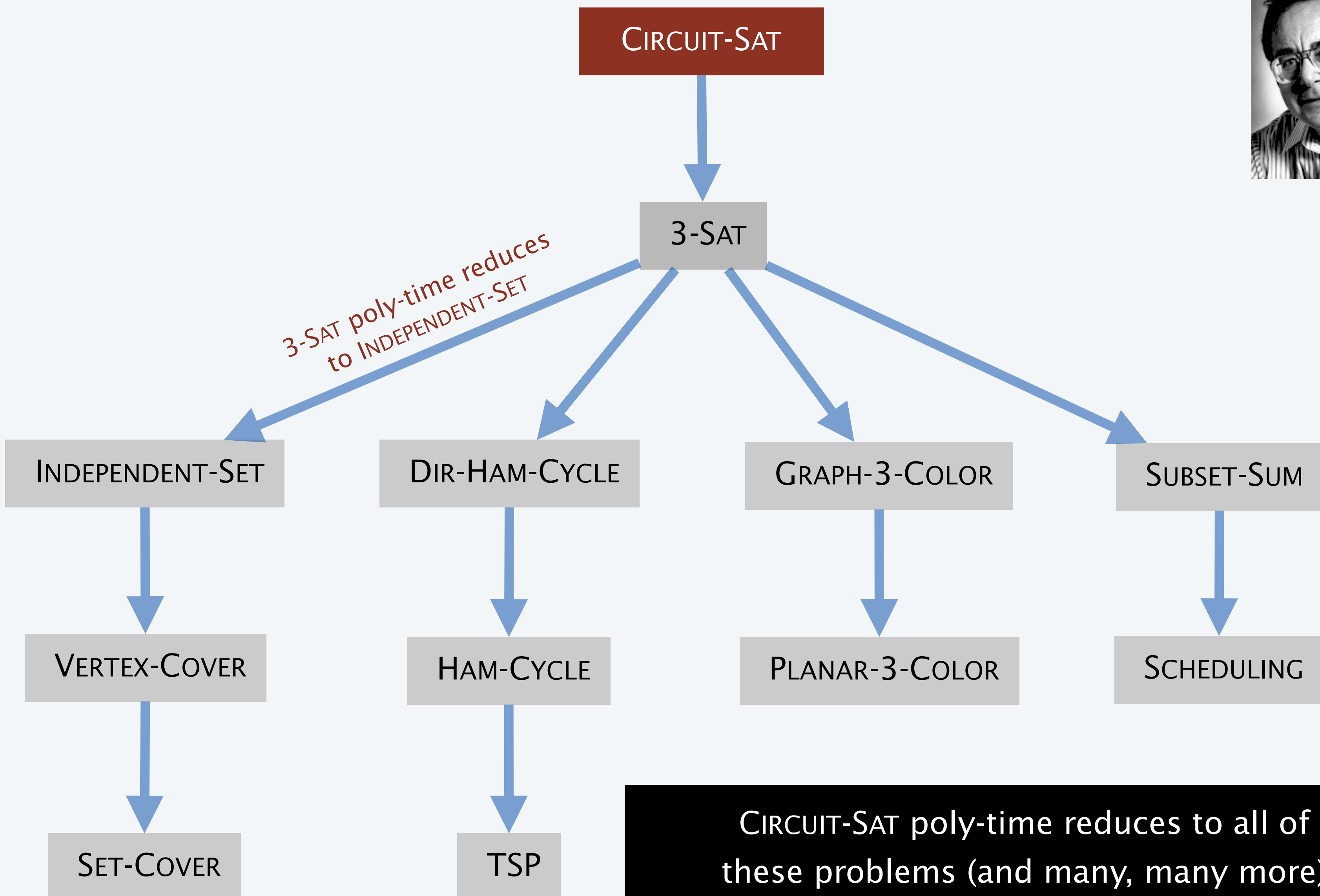
Pf. $\Rightarrow$ Suppose $\Phi$ is satisfiable.

- 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. ▪



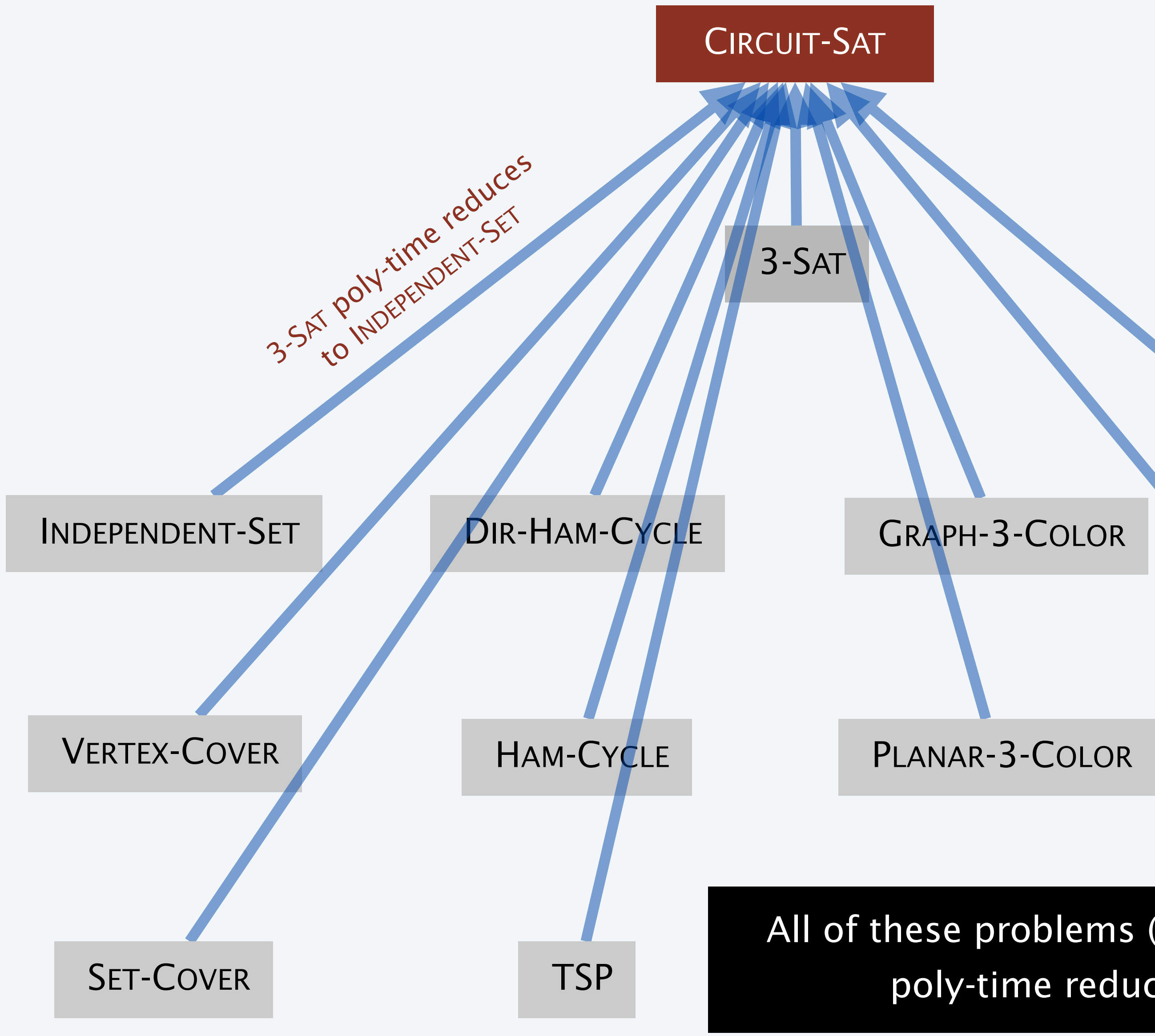
# Implications of Karp

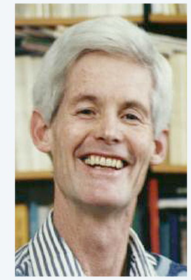


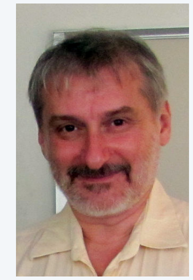



CIRCUIT-SAT poly-time reduces to all of these problems (and many, many more)

# Implications of Cook-Levin

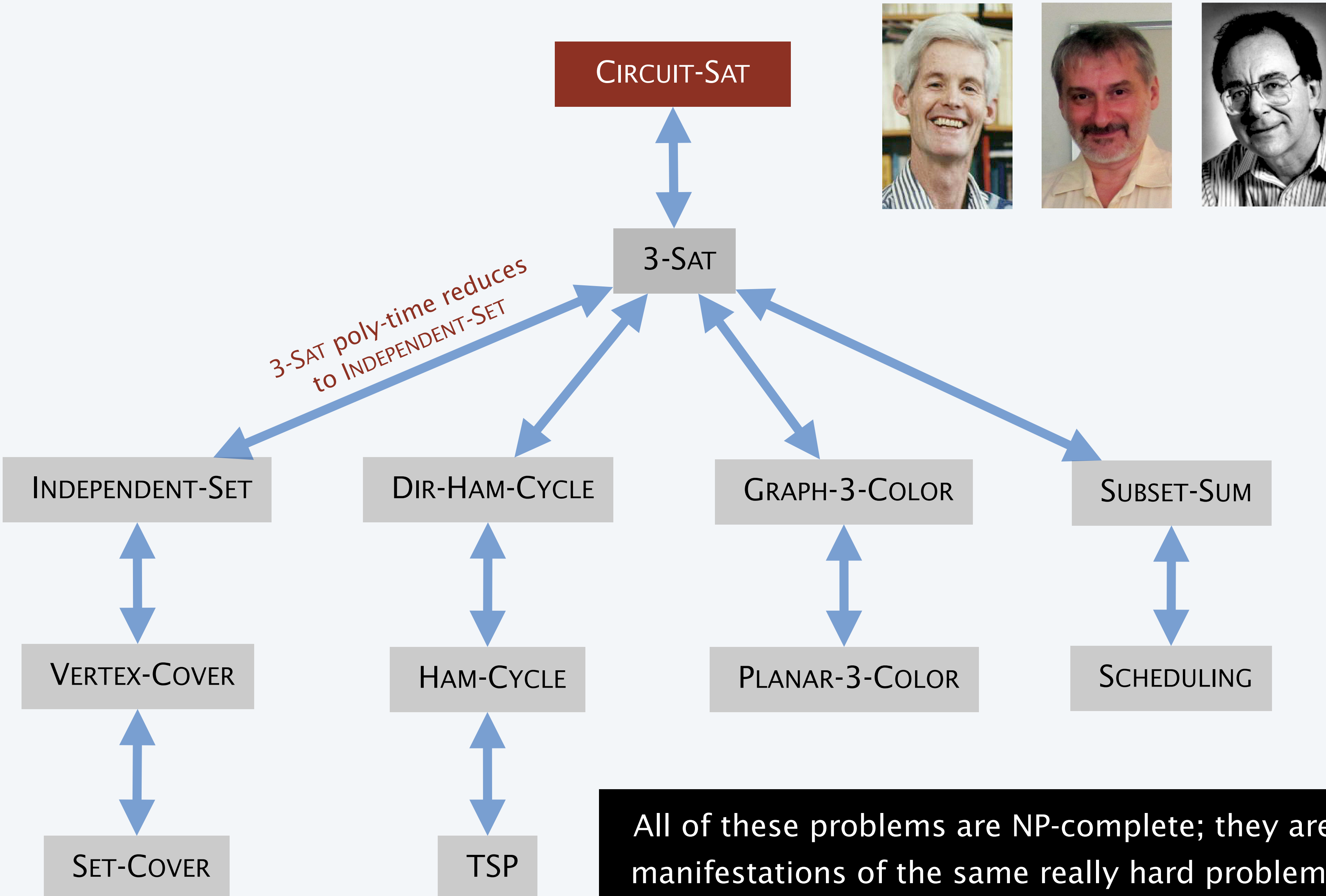




All of these problems (and many, many more) poly-time reduce to CIRCUIT-SAT.

# Implications of Karp + Cook-Levin




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

## Some NP-complete problems

Basic genres of NP-complete problems and paradigmatic examples.

- Packing + covering problems: SET-COVER, VERTEX-COVER, INDEPENDENT-SET.
- Constraint satisfaction problems: CIRCUIT-SAT, SAT, 3-SAT.
- Sequencing problems: HAM-CYCLE, TSP.
- Partitioning problems: 3D-MATCHING, 3-COLOR.
- Numerical problems: SUBSET-SUM, PARTITION.

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

Notable exceptions. FACTOR, GRAPH-ISOMORPHISM, NASH-EQUILIBRIUM.

Theory. [Ladner 1975] Unless **P** = **NP**, there exist problems in **NP** that are neither in **P** nor **NP**-complete.

# More hard computational problems

Garey and Johnson. Computers and Intractability.

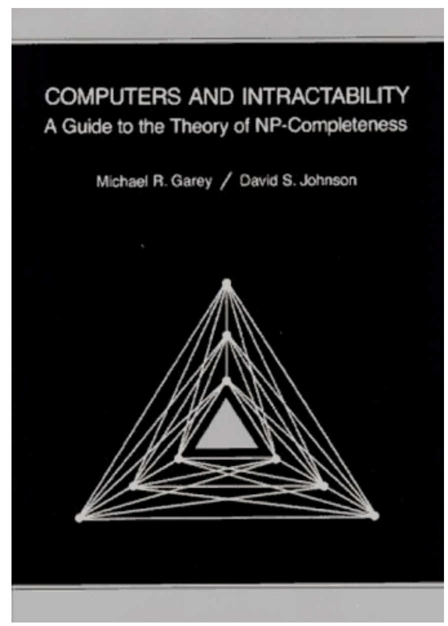
- Appendix includes over 300 **NP**-complete problems.
- Most cited reference in computer science literature.

## Most Cited Computer Science Citations

This list is generated from documents in the CiteSeer[x] database as of January 17, 2013. This list is automatically generated and may contain errors. The list is generated in batch mode and citation counts may differ from those currently in the CiteSeer[x] database, since the database is continuously updated.

All Years | 1990 | 1991 | 1992 | 1993 | 1994 | 1995 | 1996 | 1997 | 1998 | 1999 | 2000 | 2001 | 2002 | 2003 | 2004 | 2005 | 2006 | 2007 | 2008 | 2009 | 2010 | 2011 | 2012 | 2013

1. M R Garey, D S Johnson
   Computers and Intractability. A Guide to the Theory of NP-Completeness 1979
   8665
2. T Cormen, C E Leiserson, R Rivest
   Introduction to Algorithms 1990
   7210
3. V N Vapnik
   The nature of statistical learning theory 1998
   6580
4. A P Dempster, N M Laird, D B Rubin
   Maximum likelihood from incomplete data via the EM algorithm. Journal of the Royal Statistical Society, 1977
   6082
5. T Cover, J Thomas
   Elements of Information Theory 1991
   6075
6. D E Goldberg
   Genetic Algorithms in Search, Optimization, and Machine Learning, 1989
   5998
7. J Pearl
   Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference 1988
   5582
8. E Gamma, R Helm, R Johnson, J Vlissides
   Design Patterns: Elements of Reusable Object-Oriented Software 1995
   4614
9. C E Shannon
   A mathematical theory of communication Bell Syst. Tech. J, 1948
   4118
10. J R Quinlan
    C4.5: Programs for Machine Learning 1993
    4018



# More hard computational problems

Aerospace engineering. Optimal mesh partitioning for finite elements.

Biology. Phylogeny reconstruction.

Chemical engineering. Heat exchanger network synthesis.

Chemistry. Protein folding.

Civil engineering. Equilibrium of urban traffic flow.

Economics. Computation of arbitrage in financial markets with friction.

Electrical engineering. VLSI layout.

Environmental engineering. Optimal placement of contaminant sensors.

Financial engineering. Minimum risk portfolio of given return.

Game theory. Nash equilibrium that maximizes social welfare.

Mathematics. Given integer $a_1$, …, $a_n$, compute $\int_0^{2\pi} \cos(a_1\theta) \times \cos(a_2\theta) \times \cdots \times \cos(a_n\theta)\, d\theta$

Mechanical engineering. Structure of turbulence in sheared flows.

Medicine. Reconstructing 3d shape from biplane angiocardiogram.

Operations research. Traveling salesperson problem.

Physics. Partition function of 3d Ising model.

Politics. Shapley-Shubik voting power.

Recreation. Versions of Sudoko, Checkers, Minesweeper, Tetris.

Statistics. Optimal experimental design.

# Extent and impact of NP-completeness

Extent of NP-completeness. [Papadimitriou 1995]

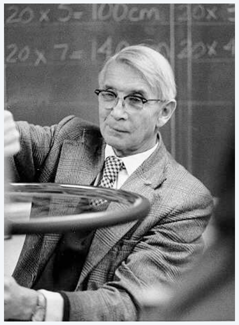
- Prime intellectual export of CS to other disciplines.
- 6,000 citations per year (more than "compiler", "OS", "database").
- Broad applicability and classification power.

NP-completeness can guide scientific inquiry.

- 1926: Ising introduces simple model for phase transitions.
- 1944: Onsager finds closed-form solution to 2D-ISING in tour de force.
- 19xx: Feynman and other top minds seek solution to 3D-ISING.
- 2000: Istrail proves 3D-ISING $\in$ **NP**-complete.

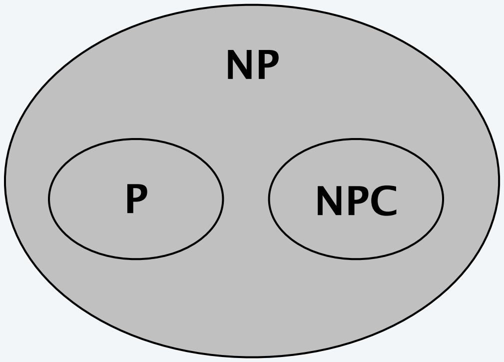
a holy grail of statistical mechanics

search for closed formula appears doomed

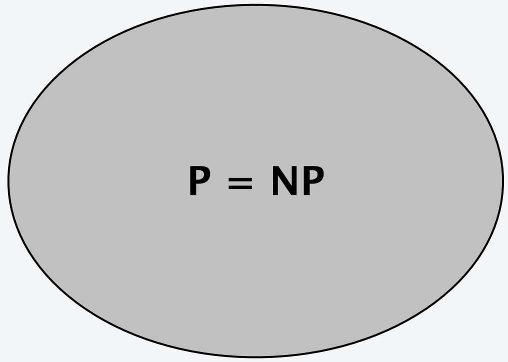



# P vs. NP revisited

Overwhelming consensus (still). P ≠ NP.




P ≠ NP

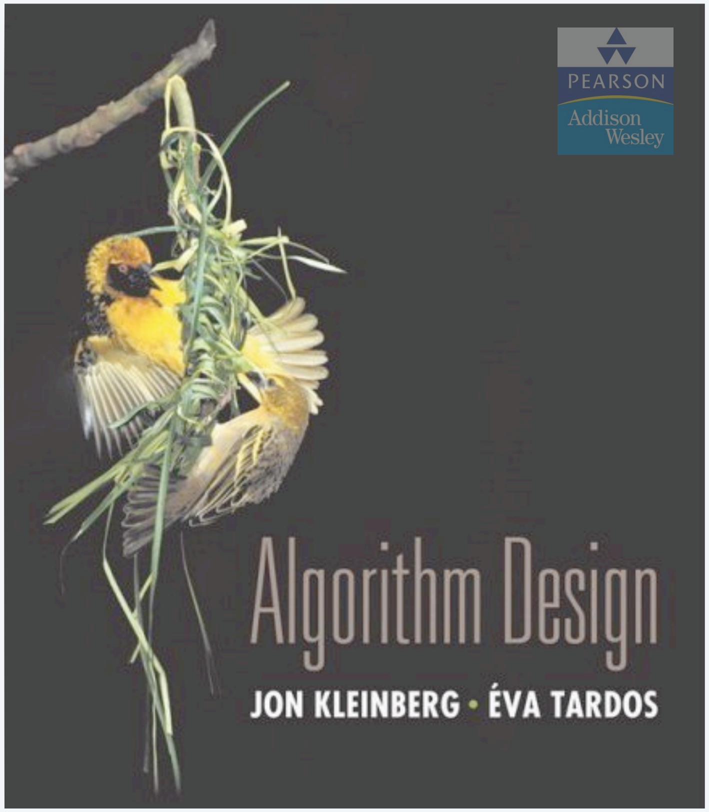



P = NP

Why we believe **P ≠ NP**.

> " *We admire Wiles' proof of Fermat's last theorem, the scientific theories of Newton, Einstein, Darwin, Watson and Crick, the design of the Golden Gate bridge and the Pyramids, precisely because they seem to require a leap which cannot be made by everyone, let alone a by simple mechanical device.* " — *Avi Wigderson*

# You NP-complete me




SECTION 8.9

# 8. INTRACTABILITY II

- *P vs. NP*
- *NP-complete*
- *co-NP*
- *NP-hard*

# Asymmetry of NP

Asymmetry of NP. We only need to have short proofs of *yes* instances.

Ex 1. SAT vs. TAUTOLOGY.

- Can prove a CNF formula is satisfiable by specifying an assignment.
- How could we prove that a formula is not satisfiable?

Ex 2. HAM-CYCLE vs. NO-HAM-CYCLE.

- Can prove a graph is Hamiltonian by specifying a permutation.
- How could we prove that a graph is not Hamiltonian?

Q. How to classify TAUTOLOGY and NO-HAMILTON-CYCLE ?

- SAT $\in$ **NP**-complete and SAT $\equiv_P$ TAUTOLOGY.
- HAM-CYCLE $\in$ **NP**-complete and HAM-CYCLE $\equiv_P$ NO-HAM-CYCLE.
- But neither TAUTOLOGY nor NO-HAM-CYCLE are known to be in NP.

# NP and co-NP

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

Ex. SAT, HAMILTON-CYCLE, and COMPOSITE.

Def. Given a decision problem $X$, its complement $\overline{X}$ is the same problem with the *yes* and *no* answers reverse.

Ex. $X = \{\ 0, 1, 4, 6, 8, 9, 10, 12, 14, 15, \ldots\ \}$

$\overline{X} = \{\ 2, 3, 5, 7, 11, 13, 17, 23, 29, \ldots\ \}$

co-NP. Complements of decision problems in **NP**.

Ex. TAUTOLOGY, NO-HAMILTON-CYCLE, and PRIMES.

# NP = co-NP ?

Fundamental open question. Does **NP** = **co-NP**?

- Do *yes* instances have succinct certificates iff *no* instances do?
- Consensus opinion: no.

Theorem. If **NP** ≠ **co-NP**, then **P** ≠ **NP**.

Pf idea.

- **P** is closed under complementation.
- If **P** = **NP**, then **NP** is closed under complementation.
- In other words, **NP** = **co-NP**.
- This is the contrapositive of the theorem.

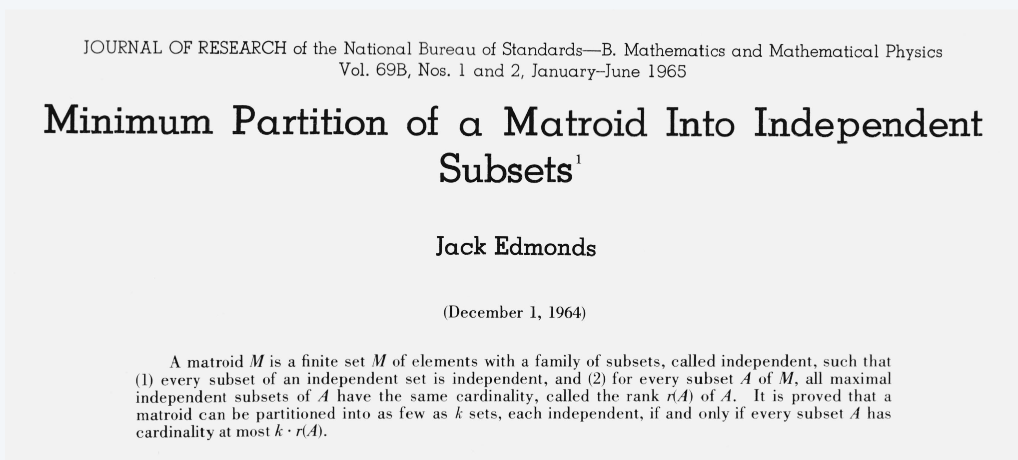
# Good characterizations

Good characterization. [Edmonds 1965] **NP** ∩ **co-NP**.

- If problem $X$ is in both **NP** and **co-NP**, then:
  - for *yes* instance, there is a succinct certificate
  - for *no* instance, there is a succinct disqualifier
- Provides conceptual leverage for reasoning about a problem.

Ex. Given a bipartite graph, is there a perfect matching.

- If yes, can exhibit a perfect matching.
- If no, can exhibit a set of nodes $S$ such that $|N(S)| < |S|$.


JOURNAL OF RESEARCH of the National Bureau of Standards—B. Mathematics and Mathematical Physics
Vol. 69B, Nos. 1 and 2, January–June 1965

Minimum Partition of a Matroid Into Independent Subsets[1]

Jack Edmonds

(December 1, 1964)

A matroid $M$ is a finite set $M$ of elements with a family of subsets, called independent, such that (1) every subset of an independent set is independent, and (2) for every subset $A$ of $M$, all maximal independent subsets of $A$ have the same cardinality, called the rank $r(A)$ of $A$. It is proved that a matroid can be partitioned into as few as $k$ sets, each independent, if and only if every subset $A$ has cardinality at most $k \cdot r(A)$.

# Good characterizations

We seek a good characterization of the minimum number of independent sets into which the columns of a matrix of $M_F$ can be partitioned. As the criterion of "good" for the characterization we apply the "principle of the absolute supervisor." The good characterization will describe certain information about the matrix which the supervisor can require his assistant to search out along with a minimum partition and which the supervisor can then use with ease to verify with mathematical certainty that the partition is indeed minimum. Having a good characterization does not mean necessarily that there is a good algorithm. The assistant might have to kill himself with work to find the information and the partition.

# Good characterizations

Observation. $\mathbf{P} \subseteq \mathbf{NP} \cap \mathbf{co\text{-}NP}$.

- Proof of max-flow min-cut theorem led to stronger result that max-flow and min-cut are in **P**.
- Sometimes finding a good characterization seems easier than finding an efficient algorithm.

Fundamental open question. Does $\mathbf{P} = \mathbf{NP} \cap \mathbf{co\text{-}NP}$?

- Mixed opinions.
- Many examples where problem found to have a nontrivial good characterization, but only years later discovered to be in **P**.

# Linear programming is in NP ∩ co-NP

**Linear programming.** Given $A \in \Re^{m \times n}$, $b \in \Re^m$, $c \in \Re^n$, and $\alpha \in \Re$, does there exist $x \in \Re^n$ such that $Ax \le b$, $x \ge 0$ and $c^{\mathrm{T}} x \ge \alpha$ ?

**Theorem.** [Gale-Kuhn-Tucker 1948] LINEAR-PROGRAMMING $\in$ **NP** $\cap$ **co-NP.**

**Pf sketch.** If (P) and (D) are nonempty, then max = min.

$$
\begin{array}{llrcl}
\text{(P)} & \max c^T x & & & \\
 & \text{s.t.} & Ax & \le & b \\
 & & x & \ge & 0
\end{array}
\qquad\qquad
\begin{array}{llrcl}
\text{(D)} & \min y^T b & & & \\
 & \text{s.t.} & A^T y & \ge & c \\
 & & y & \ge & 0
\end{array}
$$

CHAPTER XIX

LINEAR PROGRAMMING AND THE THEORY OF GAMES [1]

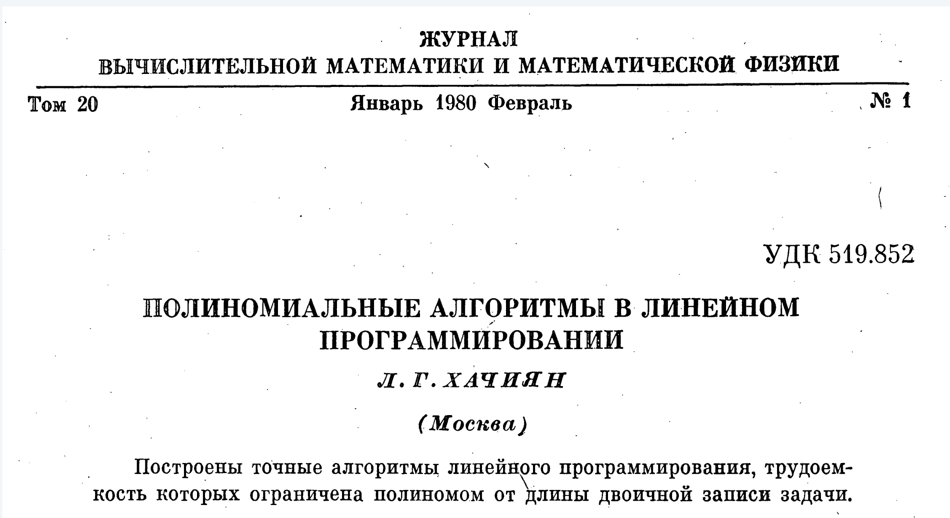
BY DAVID GALE, HAROLD W. KUHN, AND ALBERT W. TUCKER [2]

The basic "scalar" problem of *linear programming* is to maximize (or minimize) a linear function of several variables constrained by a system of linear inequalities [Dantzig, II]. A more general "vector" problem calls for maximizing (in a sense of partial order) a system of linear functions of several variables subject to a system of linear inequalities and, perhaps, linear equations [Koopmans, III]. The purpose of this chapter is to establish theorems of duality and existence for general "matrix" problems of linear programming which contain the "scalar" and "vector" problems as special cases, and to relate these general problems to the theory of zero-sum two-person games.

# Linear programming is in NP ∩ co-NP

Linear programming. Given $A \in \Re^{m \times n}$, $b \in \Re^m$, $c \in \Re^n$, and $\alpha \in \Re$, does there exist $x \in \Re^n$ such that $Ax \le b$, $x \ge 0$ and $c^T x \ge \alpha$ ?

Theorem. [Khachiyan 1979] LINEAR-PROGRAMMING $\in$ **P.**



ЖУРНАЛ
ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И МАТЕМАТИЧЕСКОЙ ФИЗИКИ

Том 20 Январь 1980 Февраль № 1

УДК 519.852

ПОЛИНОМИАЛЬНЫЕ АЛГОРИТМЫ В ЛИНЕЙНОМ ПРОГРАММИРОВАНИИ

*Л. Г. ХАЧИЯН*

*(Москва)*

Построены точные алгоритмы линейного программирования, трудоемкость которых ограничена полиномом от длины двоичной записи задачи.

# Primality testing is in NP ∩ co-NP

**Theorem.** [Pratt 1975] PRIMES $\in$ **NP** $\cap$ **co-NP.**

SIAM J. COMPUT.
Vol. 4, No. 3, September 1975

## EVERY PRIME HAS A SUCCINCT CERTIFICATE*

VAUGHAN R. PRATT†

**Abstract.** To prove that a number $n$ is composite, it suffices to exhibit the working for the multiplication of a pair of factors. This working, represented as a string, is of length bounded by a polynomial in $\log_2 n$. We show that the same property holds for the primes. It is noteworthy that almost no other set is known to have the property that short proofs for membership or nonmembership exist for all candidates without being known to have the property that such proofs are easy to come by. It remains an open problem whether a prime $n$ can be recognized in only $\log_2^\alpha n$ operations of a Turing machine for any fixed $\alpha$.

The proof system used for certifying primes is as follows.

AXIOM. $(x, y, 1)$.

INFERENCE RULES.

$$R_1: \quad (p, x, a), q \vdash (p, x, qa) \quad \text{provided } x^{(p-1)/q} \not\equiv 1 \pmod p \text{ and } q \mid (p-1).$$

$$R_2: \quad (p, x, p-1) \vdash p \quad \text{provided } x^{p-1} \equiv 1 \pmod p.$$

THEOREM 1. *$p$ is a theorem $\equiv$ $p$ is a prime.*

THEOREM 2. *$p$ is a theorem $\supset$ $p$ has a proof of $\lceil 4 \log_2 p \rceil$ lines.*

# Primality testing is in NP ∩ co-NP

**Theorem.** [Pratt 1975] PRIMES ∈ **NP** ∩ **co-NP**.

**Pf sketch.** An odd integer $s$ is prime iff there exists an integer $1 < t < s$ s.t.

$$t^{s-1} \equiv 1 \pmod{s}$$
$$t^{(s-1)/p} \not\equiv 1 \pmod{s}$$

for all prime divisors $p$ of $s$-1

| | |
|---|---|
| **instance s** | 437677 |
| **certificate t** | 17, $2^2 \times 3 \times 36473$ |

prime factorization of s–1
also need a recursive certificate
to assert that 3 and 36,473 are prime

CERTIFIER ($s$)

CHECK $s - 1 = 2 \times 2 \times 3 \times 36473$.

CHECK $17^{s-1} = 1 \pmod{s}$.

CHECK $17^{(s-1)/2} \equiv 437676 \pmod{s}$.

CHECK $17^{(s-1)/3} \equiv 329415 \pmod{s}$.

CHECK $17^{(s-1)/36,473} \equiv 305452 \pmod{s}$.

use repeated squaring

# Primality testing is in P

**Theorem.** [Agrawal-Kayal-Saxena 2004] PRIMES $\in$ **P**.

Annals of Mathematics, **160** (2004), 781–793

## PRIMES is in P

By Manindra Agrawal, Neeraj Kayal, and Nitin Saxena*

**Abstract**

We present an unconditional deterministic polynomial-time algorithm that determines whether an input number is prime or composite.

# Factoring is in NP ∩ co-NP

FACTORIZE. Given an integer $x$, find its prime factorization.

FACTOR. Given two integers $x$ and $y$, does $x$ have a nontrivial factor $< y$?

Theorem. FACTOR $\equiv_P$ FACTORIZE.

Pf.

- $\leq_P$ trivial.
- $\geq_P$ binary search to find a factor; divide out the factor and repeat. ▪

Theorem. FACTOR $\in$ **NP** $\cap$ **co-NP**.

Pf.

- Certificate: a factor $p$ of $x$ that is less than $y$.
- Disqualifier: the prime factorization of $x$ (where each prime factor is less than $y$), along with a Pratt certificate that each factor is prime. ▪

# Is factoring in P ?

**Fundamental question.** Is FACTOR $\in$ **P**.

**Challenge.** Factor this number.

74037563479561712828046796097429573142593188889231289
08493623263897276503402826627689199641962511784399589
43305021275853701189680982867331732731089309005525051
16877063299072396380786710086096962537934650563796359

**RSA-704**
**($30,000 prize if you can factor)**

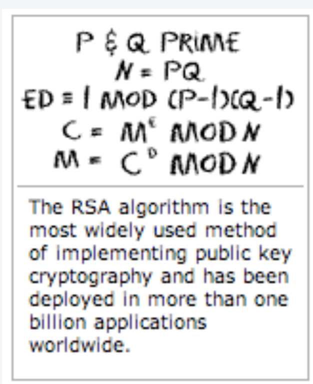
# Exploiting intractability

Modern cryptography.

- Ex. Send your credit card to Amazon.
- Ex. Digitally sign an e-document.
- Enables freedom of privacy, speech, press, political association.

RSA. Based on dichotomy between complexity of two problems.

- To use: generate two random $n$-bit primes and multiply.
- To break: suffices to factor a $2n$-bit integer.



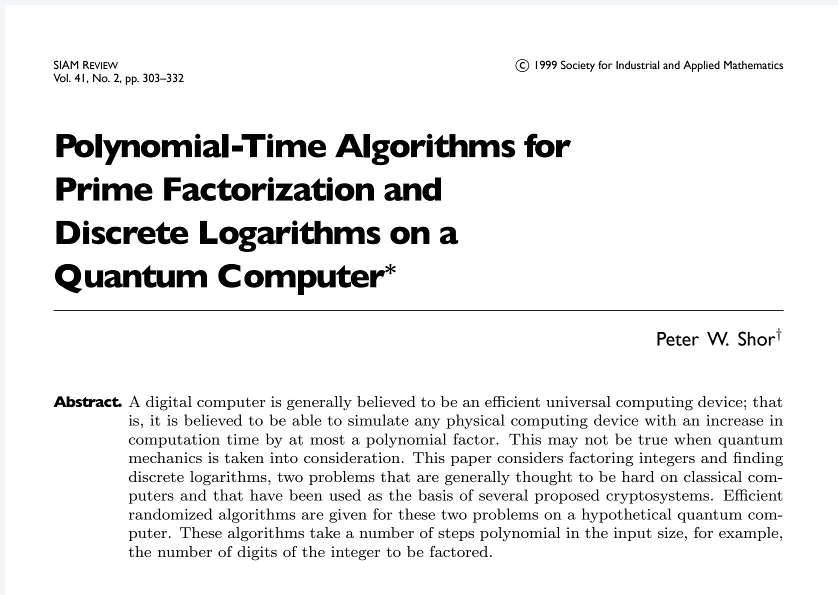

**RSA algorithm**

**RSA sold for \$2.1 billion**


**or design a t-shirt**

# Factoring on a quantum computer

Theorem. [Shor 1994] Can factor an $n$-bit integer in $O(n^3)$ steps on a "quantum computer."


SIAM Review
Vol. 41, No. 2, pp. 303–332

© 1999 Society for Industrial and Applied Mathematics

**Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer***

Peter W. Shor†

**Abstract.** A digital computer is generally believed to be an efficient universal computing device; that is, it is believed to be able to simulate any physical computing device with an increase in computation time by at most a polynomial factor. This may not be true when quantum mechanics is taken into consideration. This paper considers factoring integers and finding discrete logarithms, two problems that are generally thought to be hard on classical computers and that have been used as the basis of several proposed cryptosystems. Efficient randomized algorithms are given for these two problems on a hypothetical quantum computer. These algorithms take a number of steps polynomial in the input size, for example, the number of digits of the integer to be factored.


2001. Factored $15 = 3 \times 5$ (with high probability) on a quantum computer.

2012. Factored $21 = 3 \times 7$.

Fundamental question. Does **P** = **BQP** ?

# 8. Intractability II

- *P vs. NP*
- *NP-complete*
- *co-NP*
- *NP-hard*

# A note on terminology

SIGACT News 12 January 1974

A TERMINOLOGICAL PROPOSAL

D. E. Knuth

While preparing a book on combinatorial algorithms, I felt a strong need for a new technical term, a word which is essentially a one-sided version of polynomial complete. A great many problems of practical interest have the property that they are at least as difficult to solve in polynomial time as those of the Cook-Karp class NP. I needed an adjective to convey such a degree of difficulty, both formally and informally; and since the range of practical applications is so broad, I felt it would be best to establish such a term as soon as possible.

The goal is to find an adjective x that sounds good in sentences like this:

The covering problem is x .
It is x to decide whether a given graph has a Hamiltonian circuit.
It is unknown whether or not primality testing is an x problem.

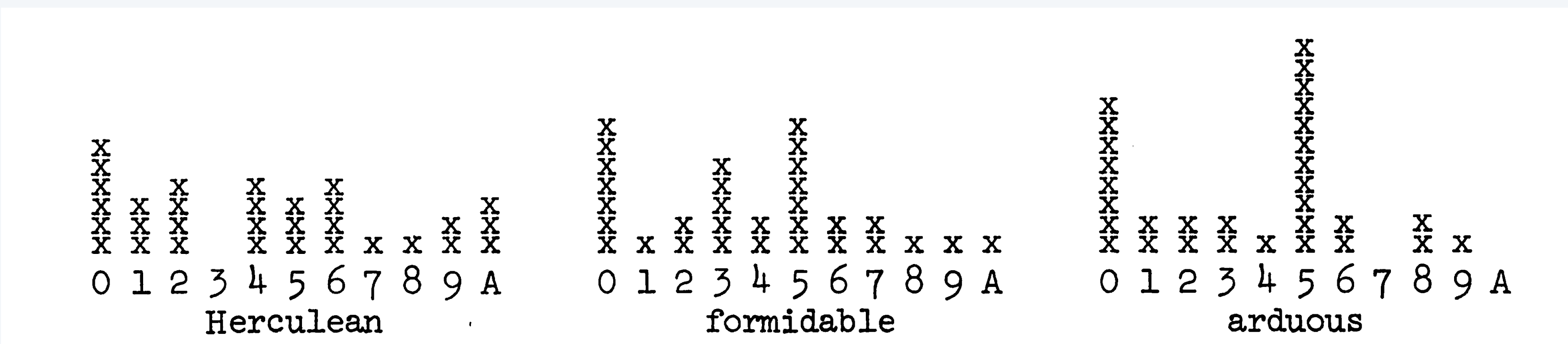
Note. The term $x$ does not necessarily imply that a problem is in **NP**, just that every problem in **NP** poly-time reduces to $x$.

# A note on terminology

Knuth's original suggestions.

- Hard.
- Tough.
- Herculean.
- Formidable.
- Arduous.

so common that it is unclear whether
it is being used in a technical sense




**assign a real number between 0 and 1 to each choice**

# A note on terminology

Some English word write-ins.

- Impractical.
- Bad.
- Heavy.
- Tricky.
- Intricate.
- Prodigious.
- Difficult.
- Intractable.
- Costly.
- Obdurate.
- Obstinate.
- Exorbitant.
- Interminable.

# A note on terminology

**Hard-boiled.** [Ken Steiglitz] In honor of Cook.

**Hard-ass.** [Al Meyer] Hard as satisfiability.

**Sisyphean.** [Bob Floyd] Problem of Sisyphus was time-consuming.

**Ulyssean.** [Don Knuth] Ulysses was known for his persistence.

> *" creative research workers are as full of ideas for new terminology as they are empty of enthusiasm for adopting it. "*
>
> *— Donald Knuth*

## A note on terminology: acronyms

PET. [Shen Lin] Probably exponential time.

- If **P** ≠ **NP**, provably exponential time.
- If **P** = **NP**, previously exponential time.

GNP. [Al Meyer] Greater than or equal to **NP** in difficulty.

- And costing more than the GNP to solve.

# A note on terminology: made-up words

Exparent. [Mike Paterson] Exponential + apparent.

Perarduous. [Mike Paterson] Through (in space or time) + completely.

Supersat. [Al Meyer] Greater than or equal to satisfiability.

Polychronious. [Ed Reingold] Enduringly long; chronic.

# A note on terminology: consensus

NP-complete. A problem in **NP** such that every problem in **NP** poly-time reduces to it.

NP-hard. [Bell Labs, Steve Cook, Ron Rivest, Sartaj Sahni]
A problem such that every problem in **NP** polynomial-time reduces to it.

One final criticism (which applies to all the terms suggested) was stated nicely by Vaughan Pratt: "If the Martians know that P = NP for Turing Machines and they kidnap me, I would lose face calling these problems 'formidable'." Yes; if P = NP , there's no need for any term at all. But I'm willing to risk such an embarrassment, and in fact I'm willing to give a prize of one live turkey to the first person who proves that P = NP .