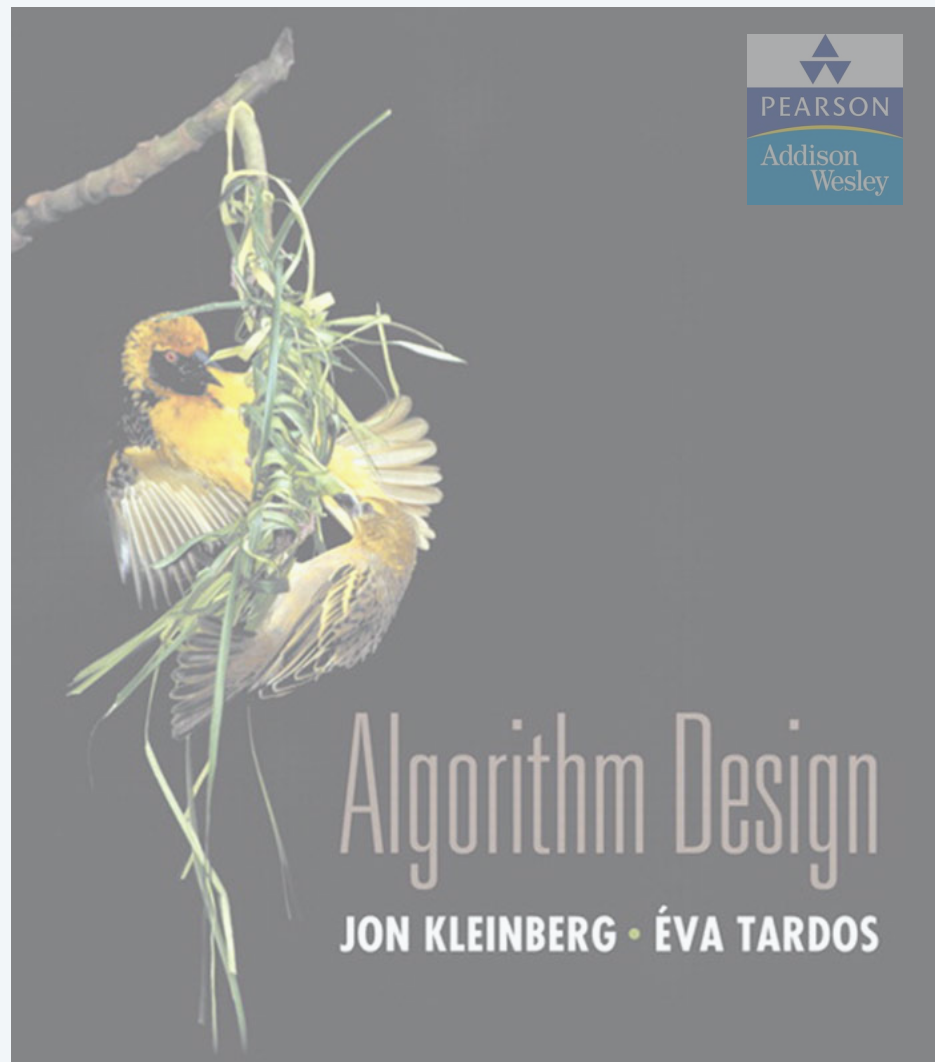


# INTRACTABILITY I

---

- ▶ *poly-time reductions*
- ▶ *packing and covering problems*
- ▶ *constraint satisfaction problems*



# INTRACTABILITY I

---

- ▶ *poly-time reductions*
- ▶ *packing and covering problems*
- ▶ *constraint satisfaction 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.
- 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.

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

# Poly-time reductions

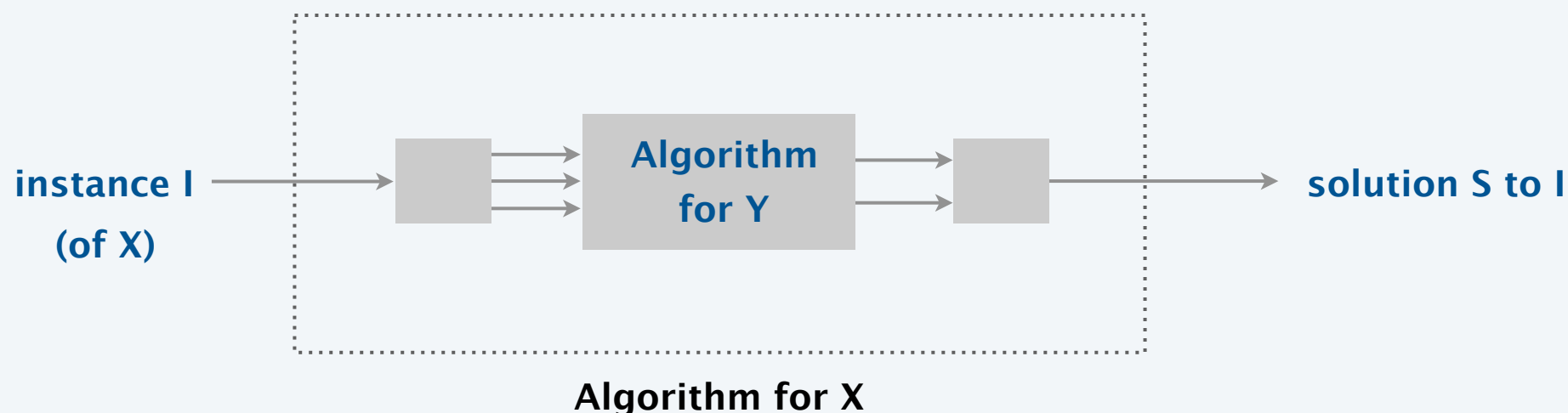
---

**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



# 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.**  $X \leq_p Y$ .

**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  $Y \leq_p X$ .

# Poly-time reductions

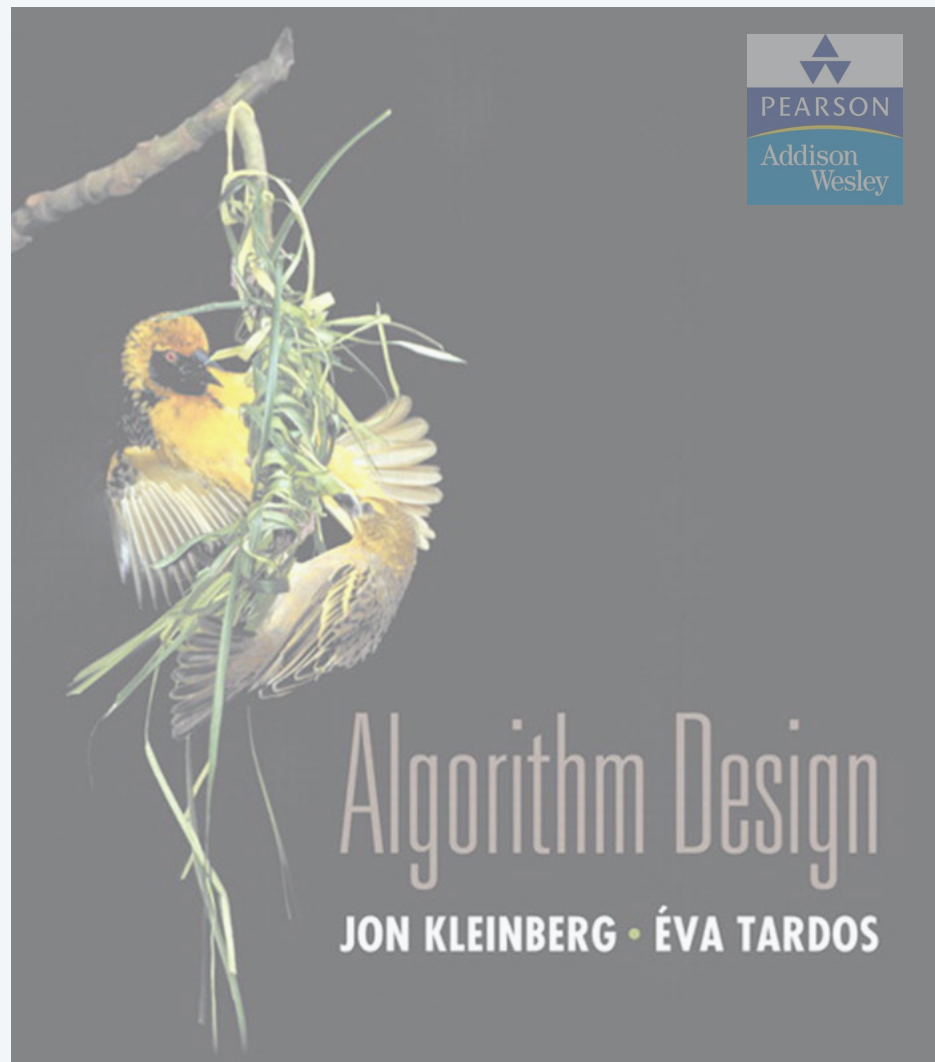
---

**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.



# INTRACTABILITY I

---

- ▶ *poly-time reductions*
- ▶ *packing and covering problems*
- ▶ *constraint satisfaction problems*



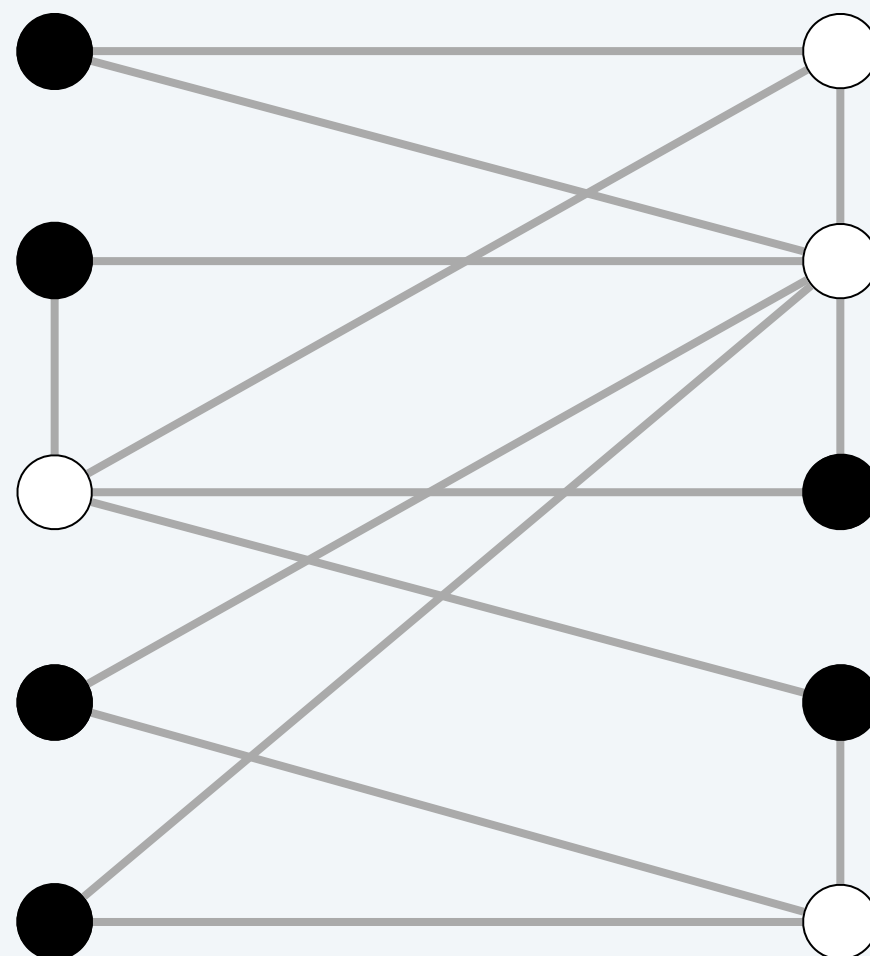
# Independent set

---

**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  $\geq 6$ ?

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



● independent set of size 6

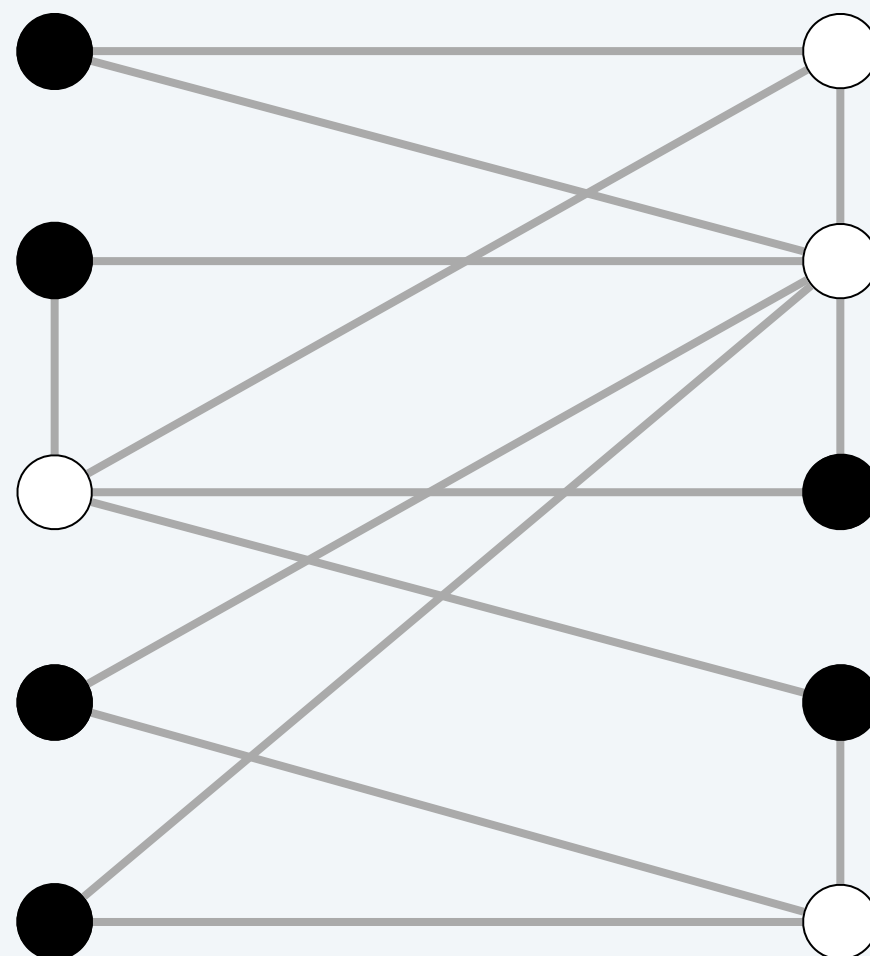
# Vertex cover

---

**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  $\leq 4$ ?

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



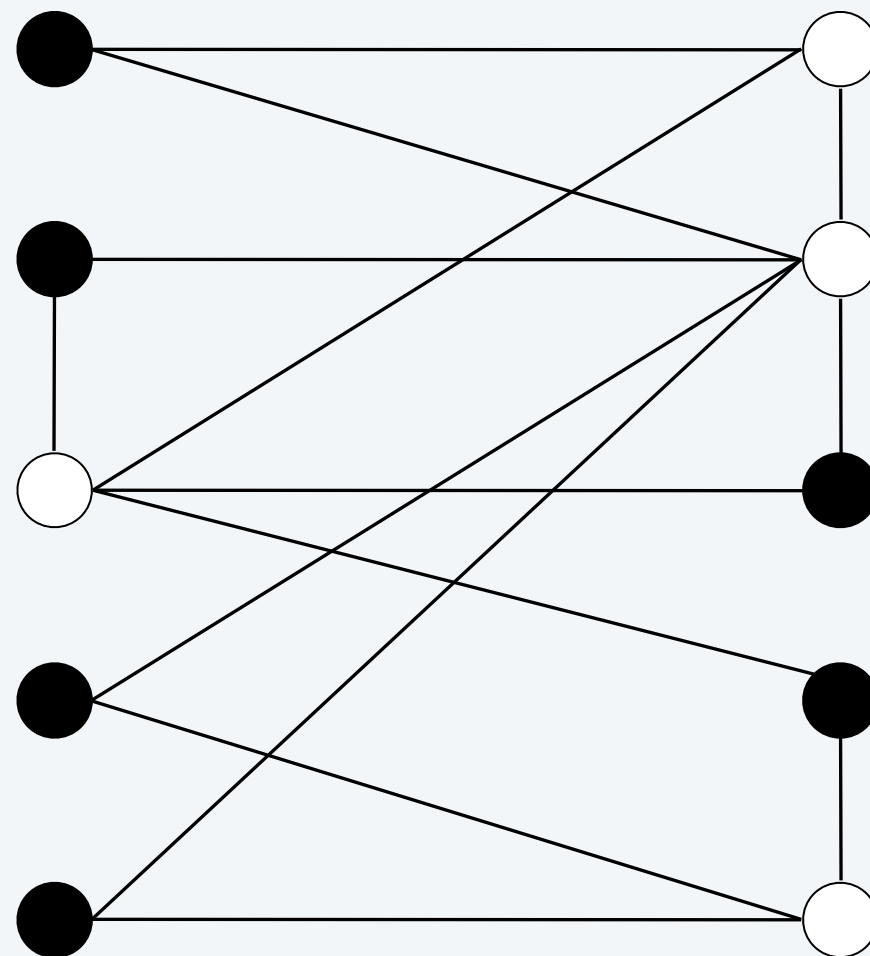
● independent set of size 6  
○ vertex cover of size 4

# Vertex cover and independent set reduce to one another

---

**Theorem.**  $\text{INDEPENDENT-SET} \equiv_P \text{VERTEX-COVER}$ .

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



● independent set of size 6  
○ vertex cover of size 4

# Vertex cover and independent set reduce to one another

---

**Theorem.**  $\text{INDEPENDENT-SET} \equiv_P \text{VERTEX-COVER}$ .

**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.**  $\text{INDEPENDENT-SET} \equiv_P \text{VERTEX-COVER}$ .

**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  $U$ ?

## 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

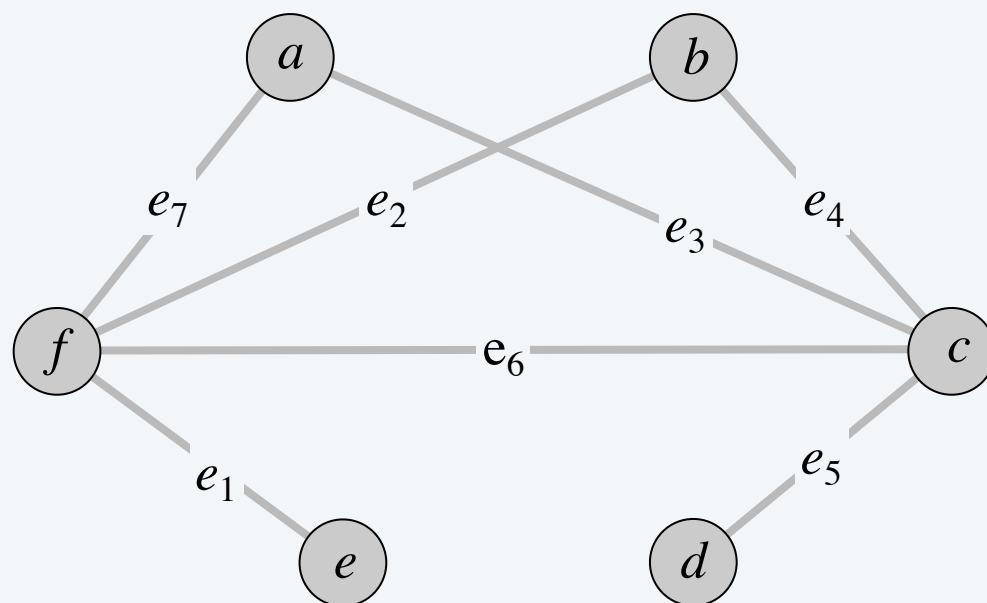
# Vertex cover reduces to set cover

**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**  
( $k = 2$ )

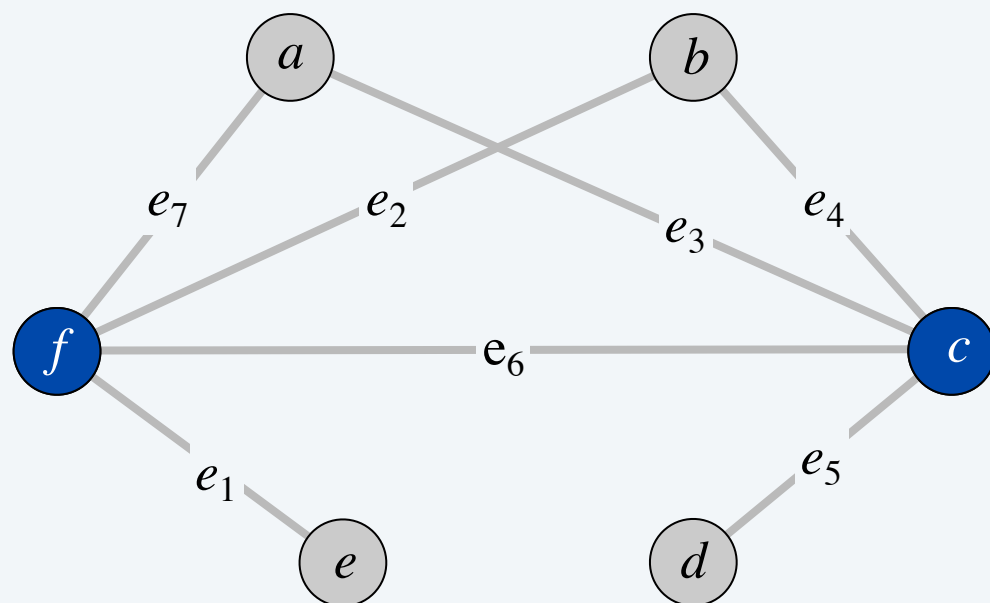
# Vertex cover reduces to set cover

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

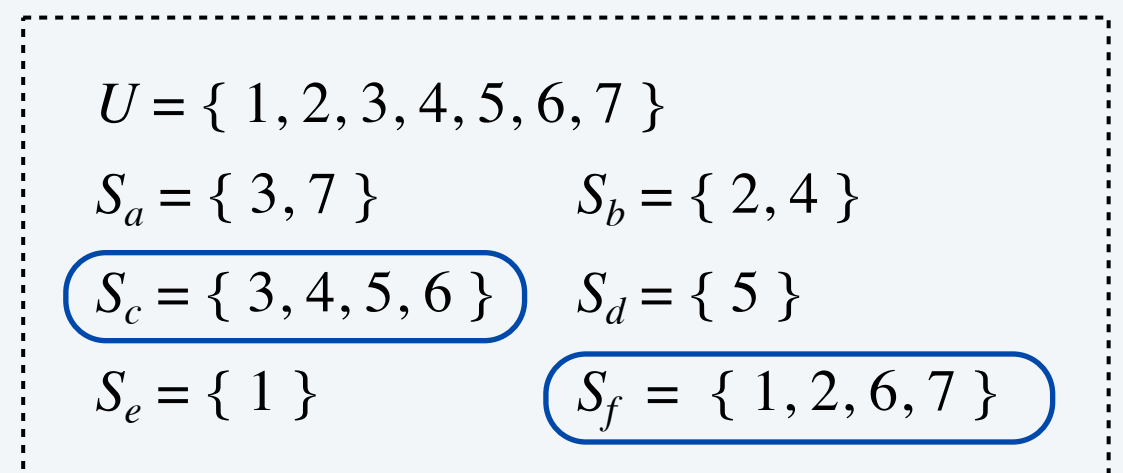
**Pf.**  $\Rightarrow$  Let  $X \subseteq V$  be a vertex cover of size  $k$  in  $G$ .

- 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  
( $k = 2$ )



set cover instance  
( $k = 2$ )



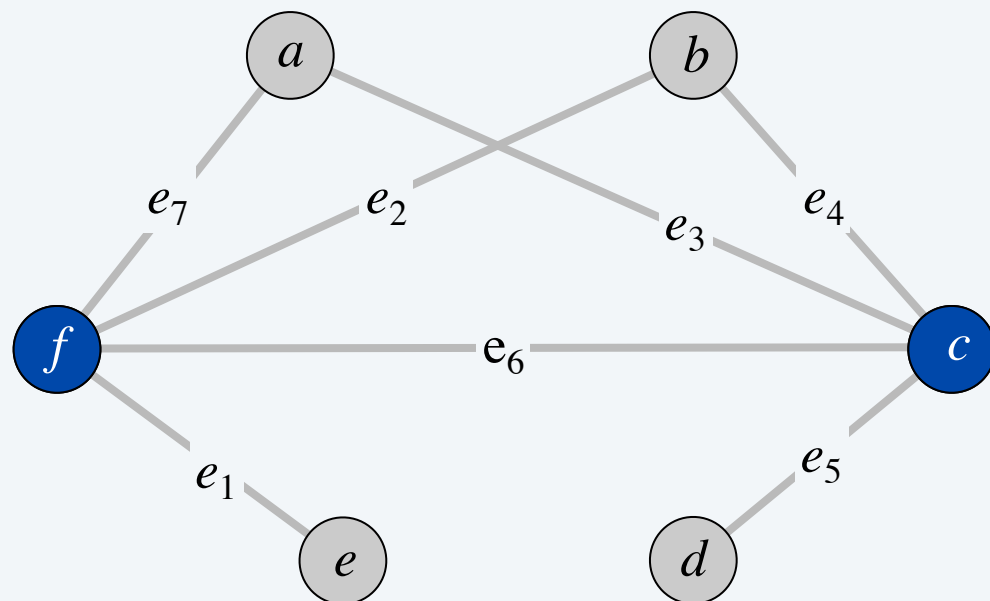
# Vertex cover reduces to set cover

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

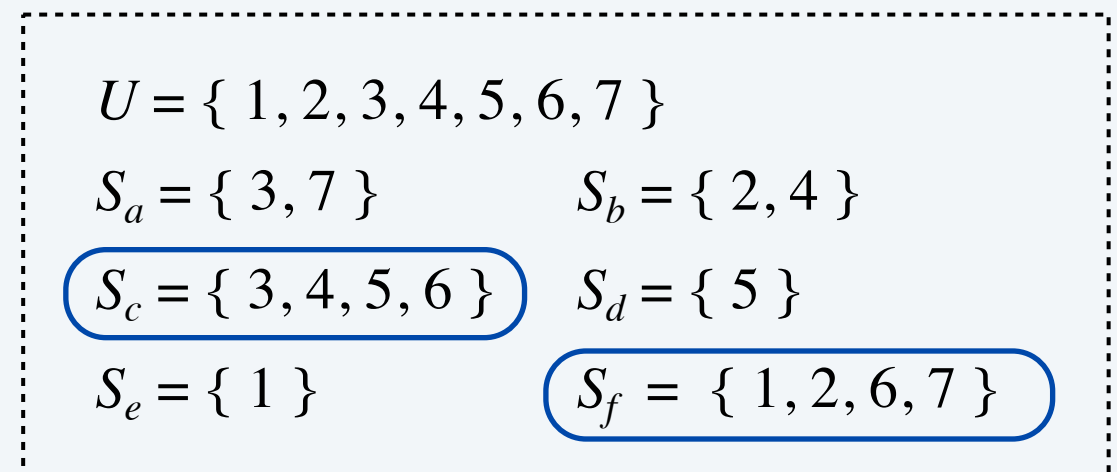
**Pf.**  $\Leftarrow$  Let  $Y \subseteq S$  be a set cover of size  $k$  in  $(U, S, k)$ .

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

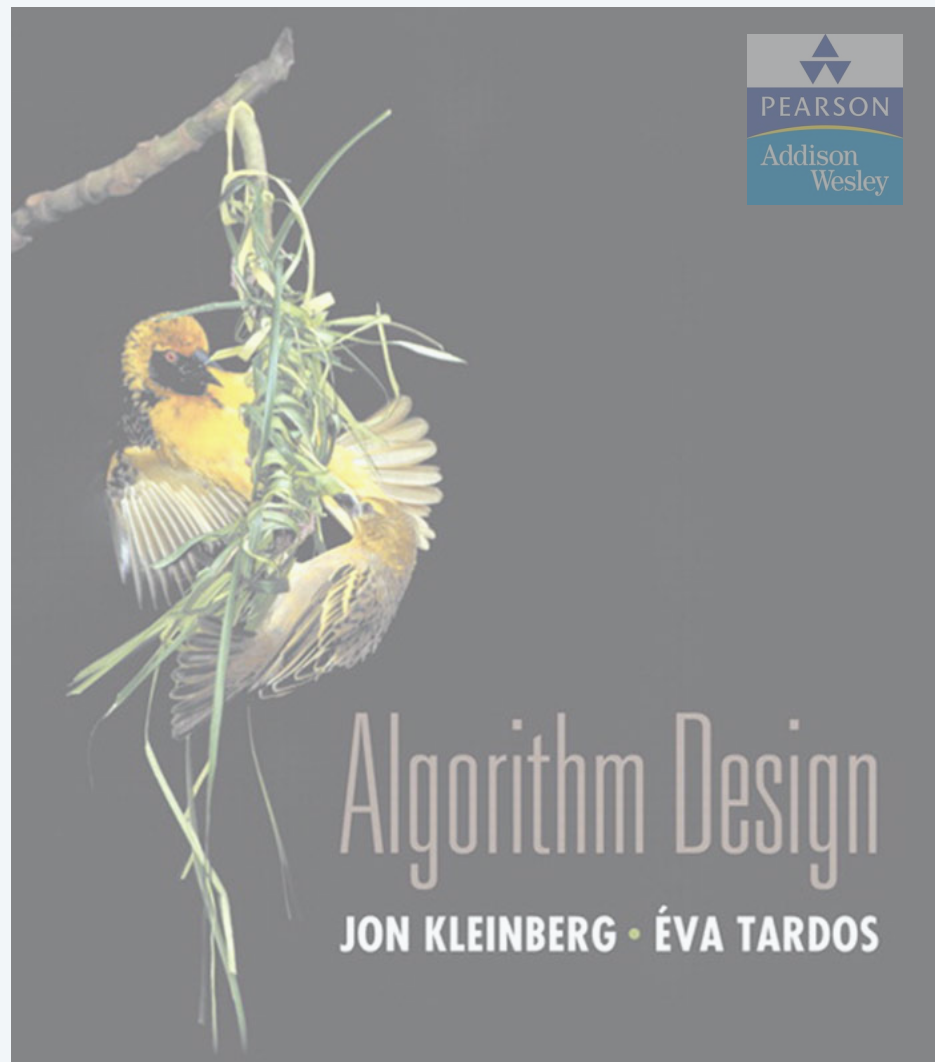
“no” instances of VERTEX-COVER are solved correctly



vertex cover instance  
( $k = 2$ )



set cover instance  
( $k = 2$ )



# INTRACTABILITY I

---

- ▶ *poly-time reductions*
- ▶ *packing and covering problems*
- ▶ *constraint satisfaction problems*

# Satisfiability

---

**Literal.** A Boolean variable or its negation.

$$x_i \text{ or } \overline{x_i}$$

**Clause.** A disjunction of literals.

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

**Conjunctive normal form (CNF).** A propositional formula  $\Phi$  that is a conjunction of clauses.

$$\Phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$$

**SAT.** Given a 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).

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

**yes instance:**  $x_1 = \text{true}$ ,  $x_2 = \text{true}$ ,  $x_3 = \text{false}$ ,  $x_4 = \text{false}$

**Key application.** Electronic design automation (EDA).

# Review

---

## Basic reduction strategies.

- Simple equivalence:  $\text{INDEPENDENT-SET} \equiv_p \text{VERTEX-COVER}$ .
- Special case to general case:  $\text{VERTEX-COVER} \leq_p \text{SET-COVER}$ .
- Encoding with gadgets:  $3\text{-SAT} \leq_p \text{INDEPENDENT-SET}$ .

**Transitivity.** If  $X \leq_p Y$  and  $Y \leq_p Z$ , then  $X \leq_p Z$ .

**Pf idea.** Compose the two algorithms.

**Ex.**  $3\text{-SAT} \leq_p \text{INDEPENDENT-SET} \leq_p \text{VERTEX-COVER} \leq_p \text{SET-COVER}$ .

# DECISION, SEARCH, AND OPTIMIZATION PROBLEMS

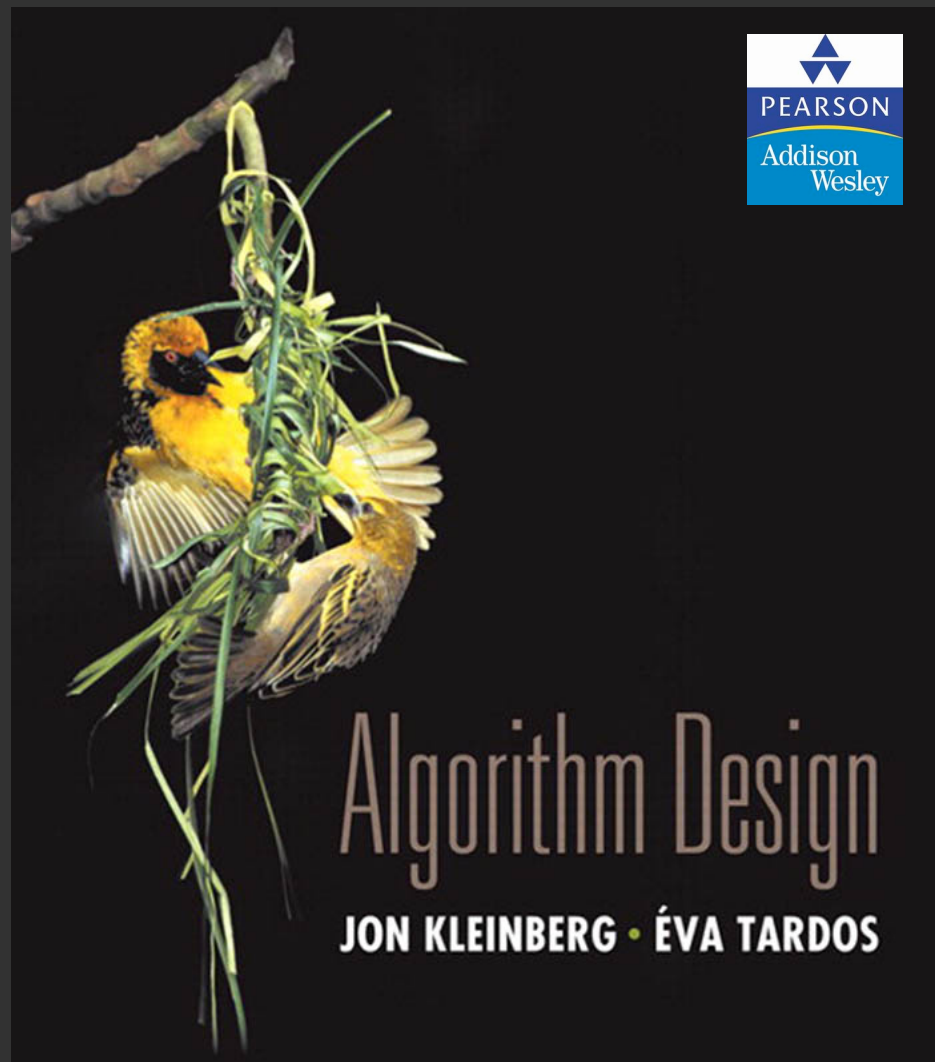


**Decision problem.** Does there **exist** a vertex cover of size  $\leq k$ ?

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

**Optimization problem.** **Find** a vertex cover of **minimum** size.

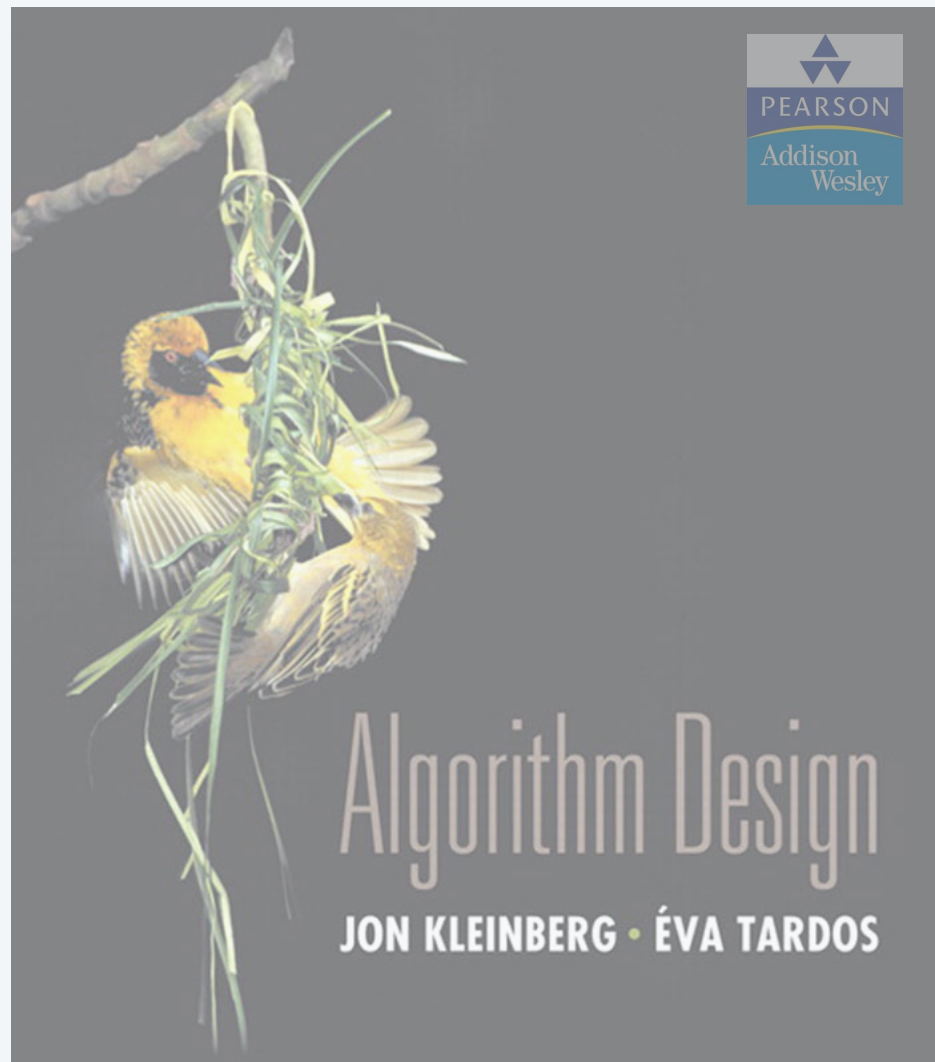
**Goal.** Show that all three problems poly-time reduce to one another.



## INTRACTABILITY II

---

- ▶  $P$  vs.  $NP$
- ▶  $NP$ -complete
- ▶  $NP$ -hard



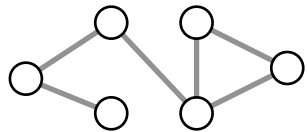
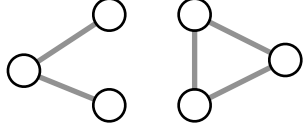
# INTRACTABILITY II

---

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

# Some problems in P

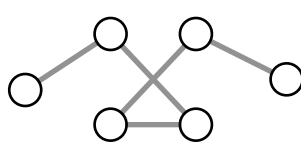
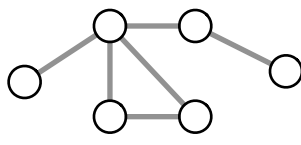
P. Decision problems for which there exists a poly-time algorithm.

problem	description	poly-time 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's algorithm	34, 39	34, 51
PRIMES	Is $x$ prime?	Agrawal–Kayal–Saxena	53	51
EDIT-DISTANCE	Is the edit distance between $x$ and $y$ less than 5?	Needleman–Wunsch	niether neither	acgggt ttttta
L-SOLVE	Is there a vector $x$ that satisfies $Ax = b$ ?	Gauss–Edmonds elimination	$\begin{bmatrix} 0 & 1 & 1 \\ 2 & 4 & -2 \\ 0 & 3 & 15 \end{bmatrix}, \begin{bmatrix} 4 \\ 2 \\ 36 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$
U-CONN	Is an undirected graph $G$ connected?	depth-first search		



# Some problems in NP

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

problem	description	poly-time 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?	Agrawal–Kayal–Saxena	51	53
FACTOR	Does $x$ have a nontrivial factor less than $y$ ?	???	(56159, 50)	(55687, 50)
SAT	Given a CNF formula, does it have a satisfying truth assignment?	???	$\neg x_1 \vee x_2 \vee \neg x_3$ $x_1 \vee \neg x_2 \vee x_3$ $\neg x_1 \vee \neg x_2 \vee x_3$	$\neg x_2$ $x_1 \vee x_2$ $\neg x_1 \vee x_2$
HAMILTON-PATH	Is there a simple path between $u$ and $v$ that visits every node?	???		

# The main question: P vs. NP

---

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

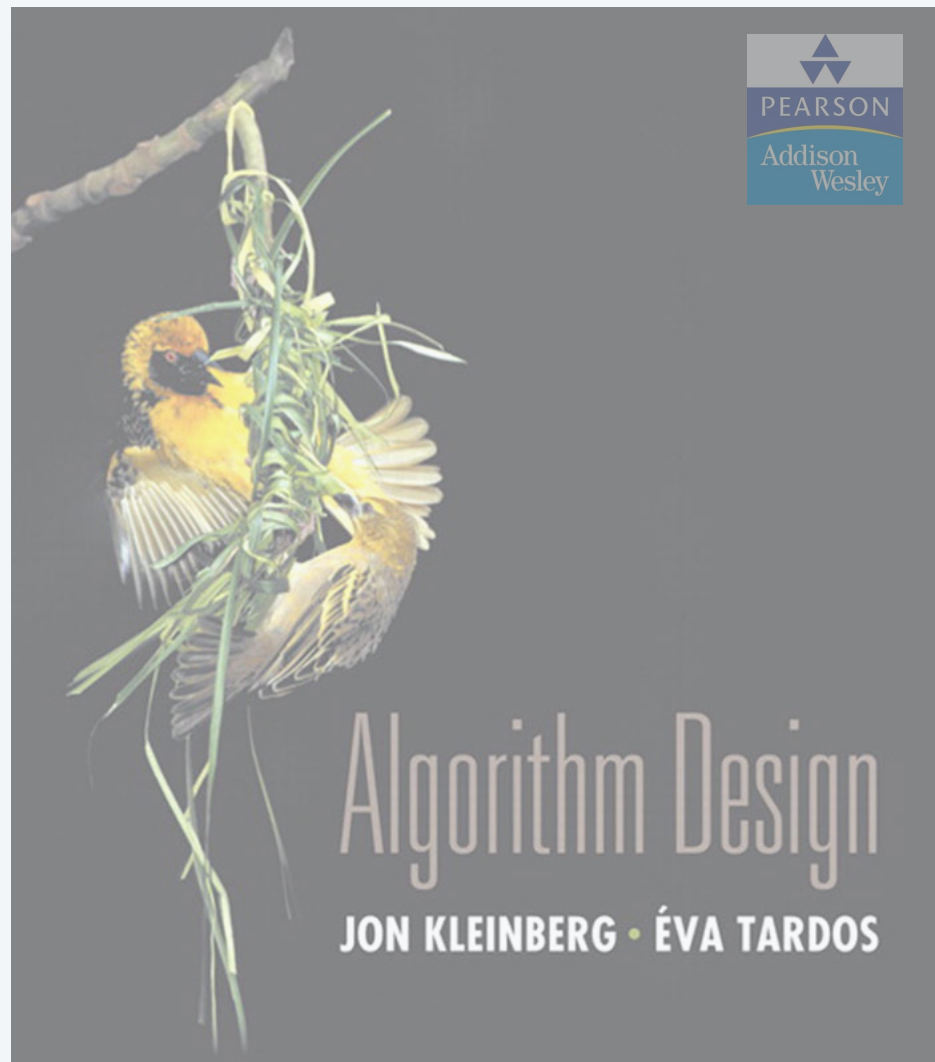
A. Exhaustive search: try all  $2^n$  truth assignments.

Q. Can we do anything substantially more clever?

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

“intractable”





# INTRACTABILITY II

---

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

# 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  $Y \in \mathbf{NP}$  with the property that for every problem  $X \in \mathbf{NP}$ ,  $X \leq_p Y$ .

**Proposition.** Suppose  $Y \in \mathbf{NP}$ -complete. Then,  $Y \in \mathbf{P}$  iff  $\mathbf{P} = \mathbf{NP}$ .

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

**Pf.**  $\Rightarrow$  Suppose  $Y \in \mathbf{P}$ .

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

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

# The "first" NP-complete problem

Theorem. [Cook 1971, Levin 1973]  $\text{SAT} \in \text{NP-complete}$ .

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  $\Sigma$ . 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  $\Sigma$ . Since we will require infinitely many proposition symbols (atoms), each such symbol will consist of a member of  $\Sigma$  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),  $\vee$  (or), and  $\neg$  (not).

The set of tautologies (denoted by  $\{\text{tautologies}\}$ ) 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  $\{\text{tautologies}\}$  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  $u \in T$  and the no state if  $u \notin T$ . 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  $w \in S$ .

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, предсказываемого 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  $k$

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

Аналогично будем понимать термин «меньше или сравнимо».

**О п р е д е л е н и е.** 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 на 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.

# Establishing NP-completeness

---

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

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

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

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

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

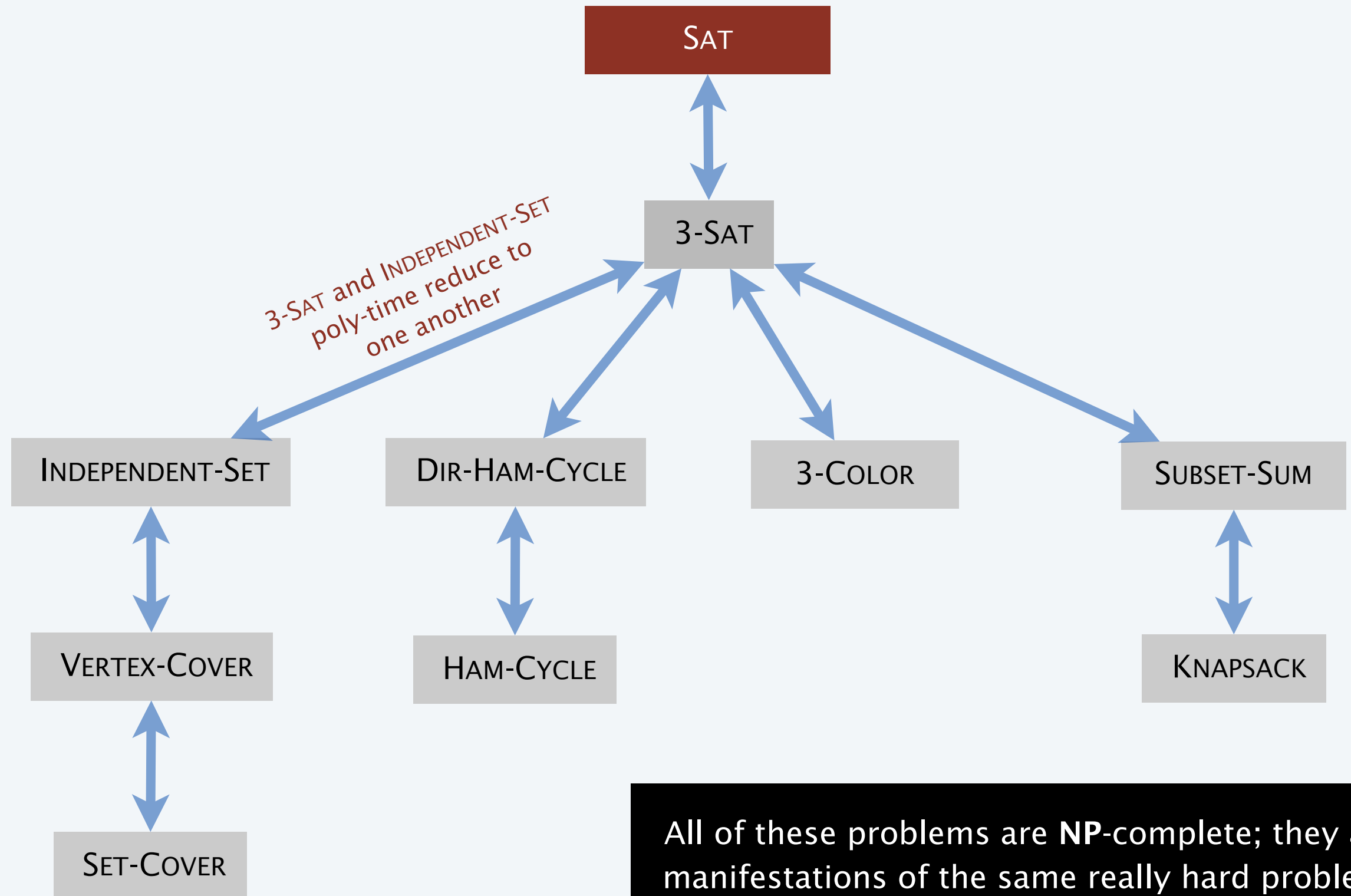
- By transitivity,  $W \leq_P Y$ .
- Hence  $Y \in \mathbf{NP}$ -complete. ■

↑  
by definition of  
**NP**-complete

↑  
by assumption



# Implications of Karp + Cook-Levin





# 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: HAMILTON-CYCLE, TSP.
- Partitioning problems: 3D-MATCHING, 3-COLOR.
- Numerical problems: SUBSET-SUM, KNAPSACK.

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

**NP-intermediate?** FACTOR, DISCRETE-LOG, GRAPH-ISOMORPHISM, ....

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

On the Structure of Polynomial Time Reducibility

RICHARD E. LADNER

*University of Washington, Seattle, Washington*



# INTRACTABILITY II

---

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

# A note on terminology

---

SIGACT News

12

January 1974

## A TERMINOLOGICAL PROPOSAL

D. F. 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: 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** poly-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  $P = NP$ .