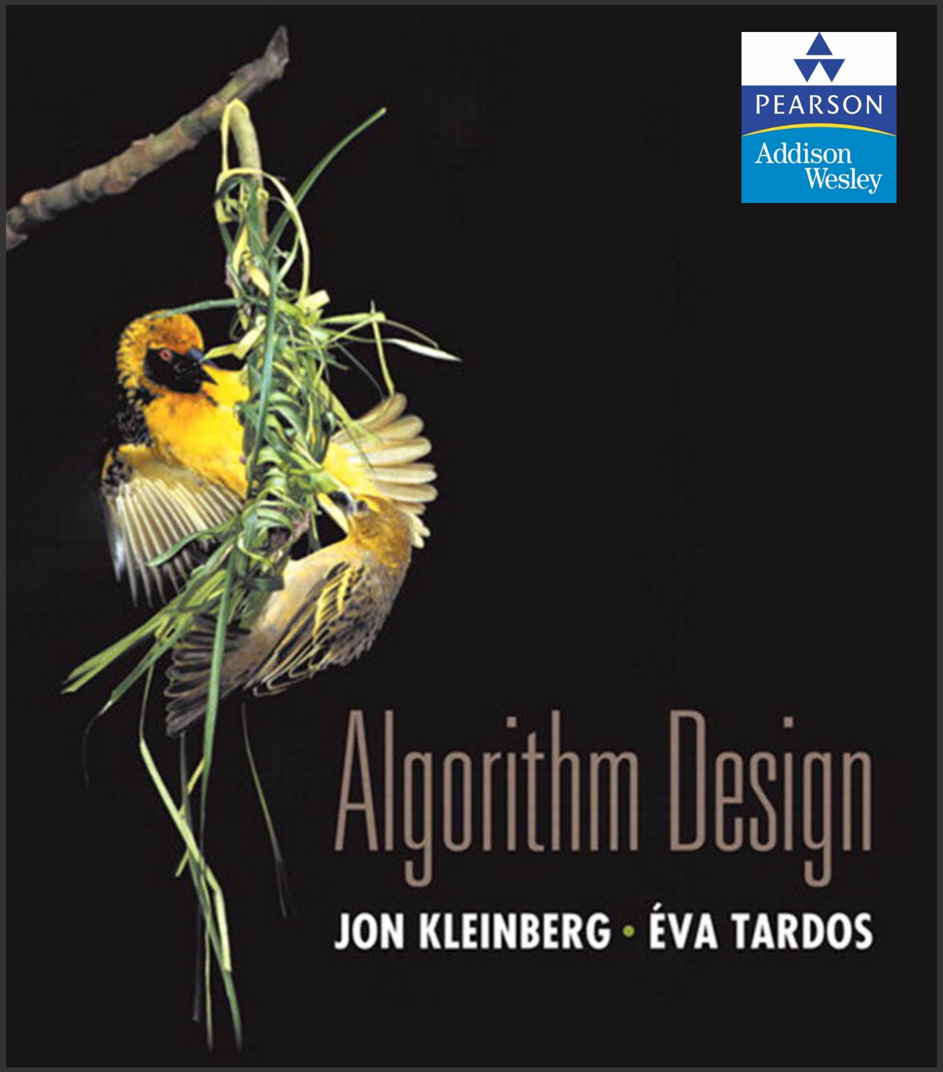
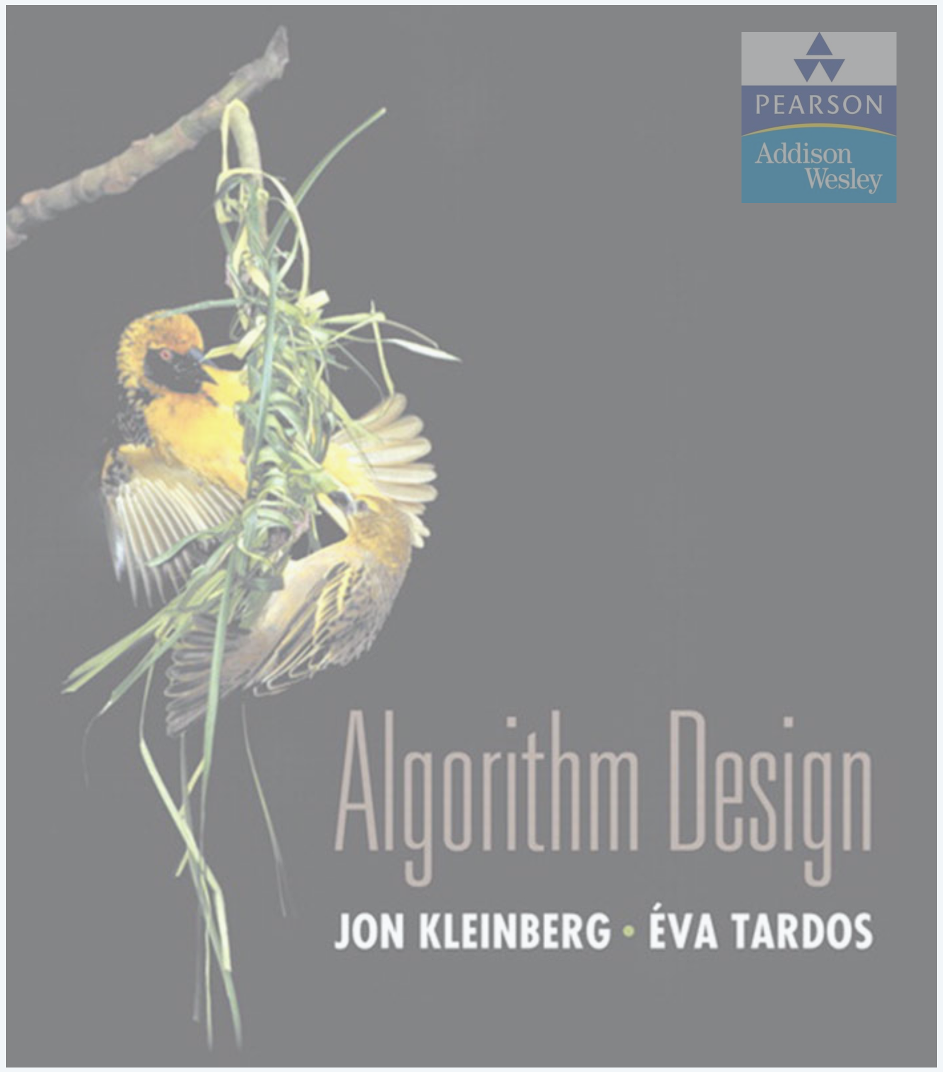


8. Intractability I

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



SECTION 8.1

8. 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.

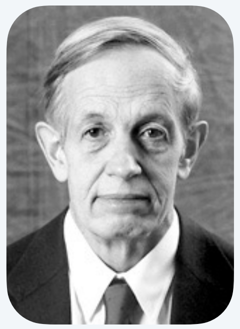
• **NP**-completeness.	$O(n^k)$ algorithm unlikely.
• **PSPACE**-completeness.	$O(n^k)$ certification algorithm unlikely.
• Undecidability.	No algorithm possible.

Classify problems according to computational requirements

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

A working definition. Those with poly-time algorithms.

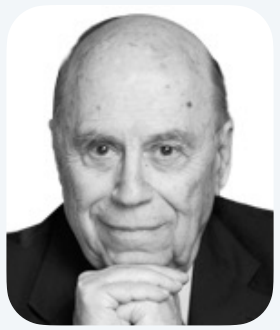
von Neumann (1953)


Nash (1955)

Gödel (1956)

Cobham (1964)

Edmonds (1965)


Rabin (1966)

Turing machine, word RAM, uniform circuits, …

Theory. Definition is broad and robust.

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

Practice. Poly-time algorithms scale to huge problems.

Classify problems

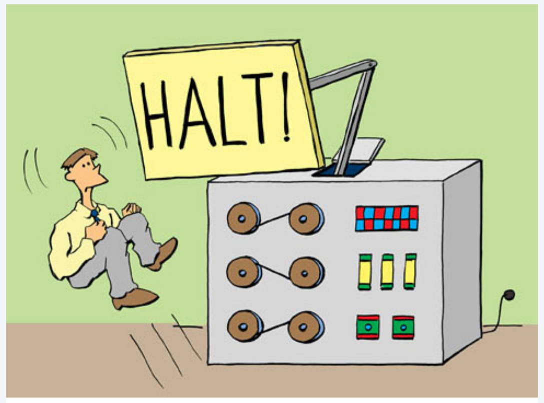
Desiderata. Classify problems according to those that can be solved in polynomial time and those that cannot.

input size = $c + \log k$

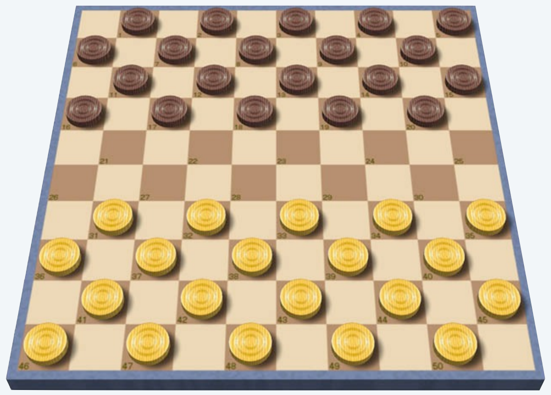
Provably requires exponential time.

- 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.

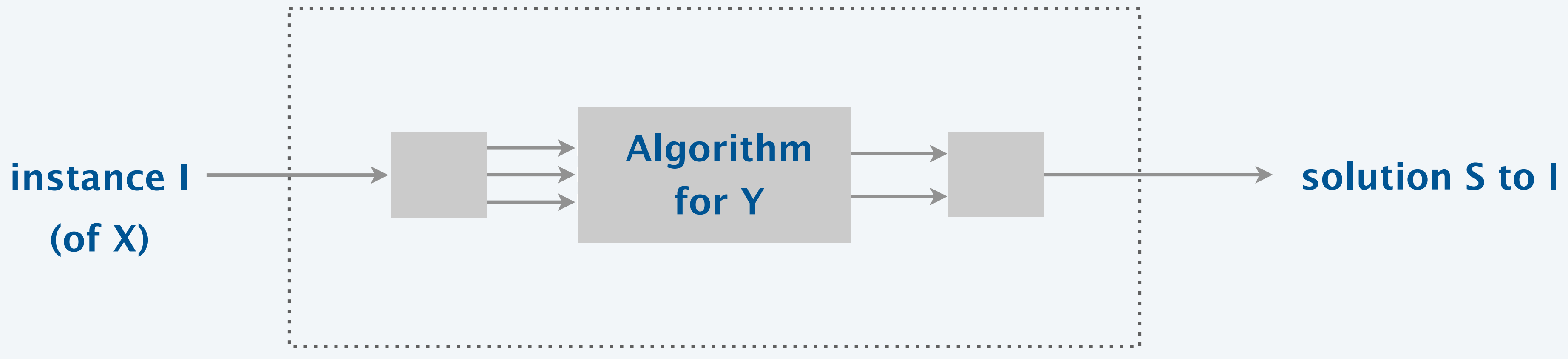
Poly-time reductions

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

Reduction. Problem X polynomial-time (Cook) reduces to problem Y if arbitrary instances of problem X can be solved using:

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

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



Algorithm for X

Poly-time reductions

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

Reduction. Problem X polynomial-time (Cook) reduces to problem Y if arbitrary instances of problem X can be solved using:

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

Notation. $X \leq_P Y$.

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

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

Intractability: quiz 1

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

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

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

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

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

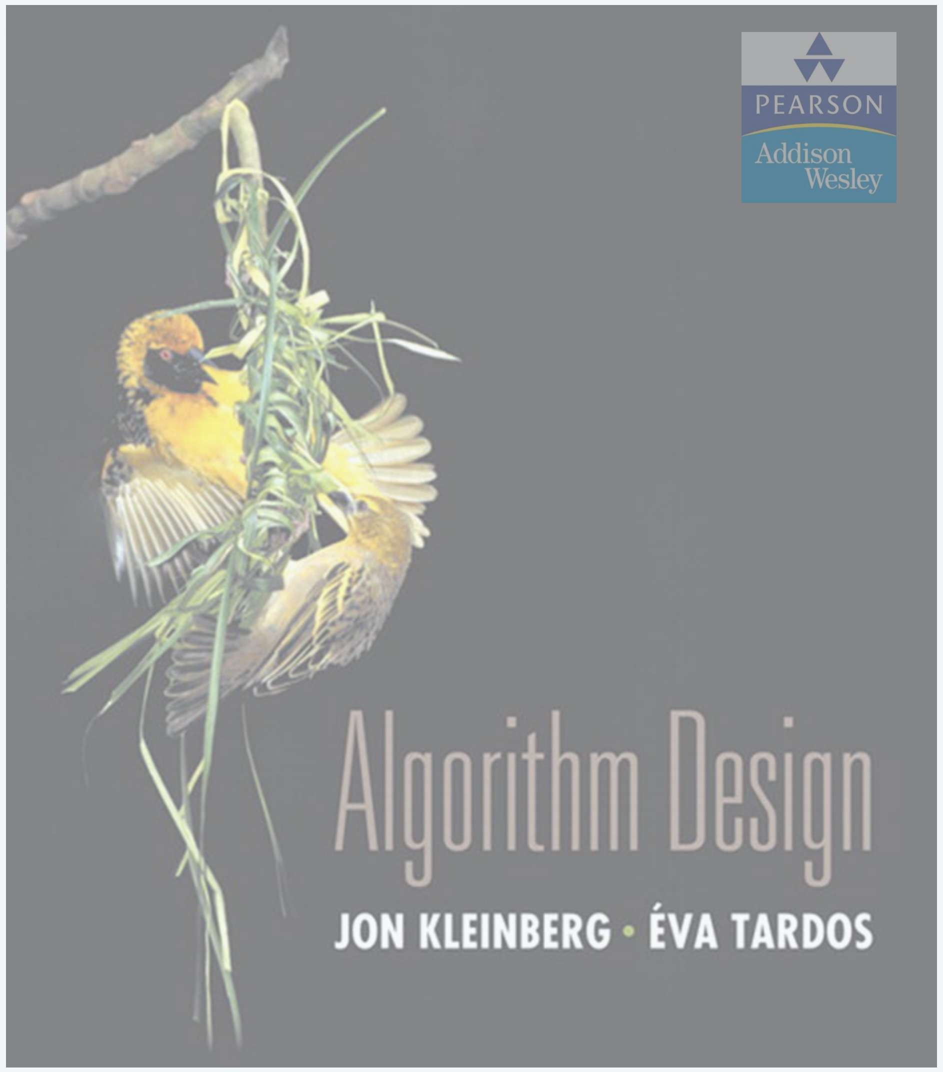
Poly-time reductions

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

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

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

Bottom line. Reductions classify problems according to relative difficulty.



SECTION 8.1

8. INTRACTABILITY I

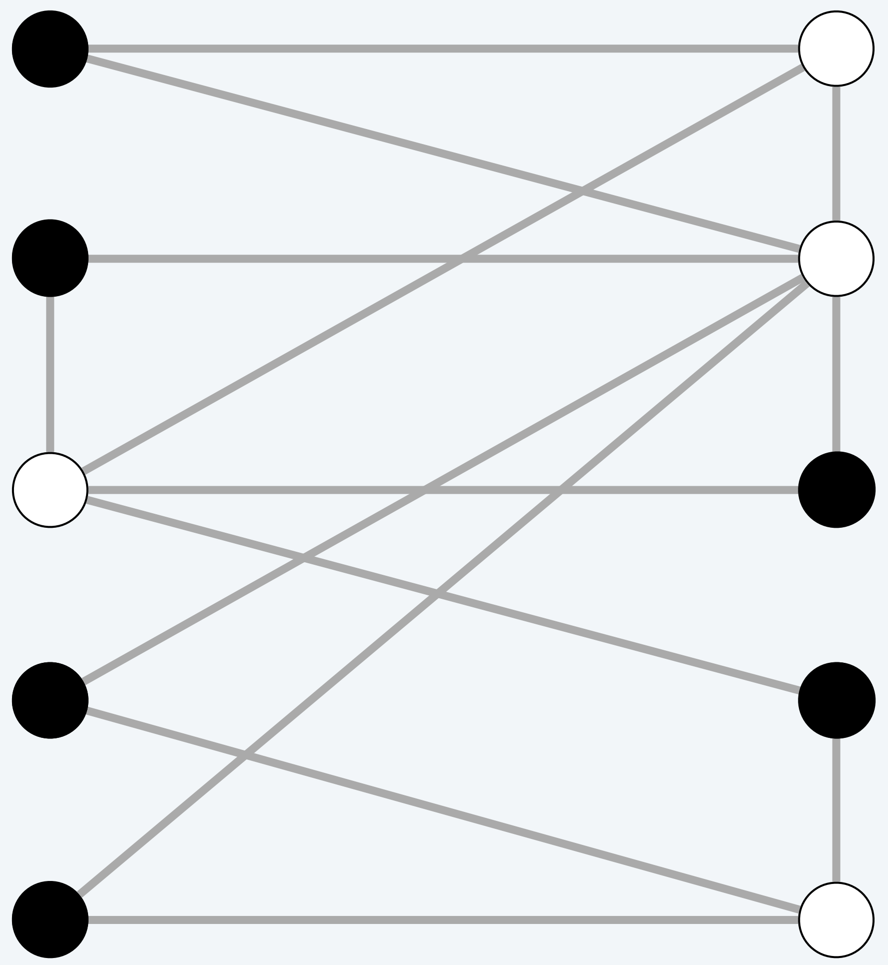
- *poly-time reductions*
- *packing and covering problems*
- *constraint satisfaction problems*

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 ≥ 6 ?

Ex. Is there an independent set of size ≥ 7 ?



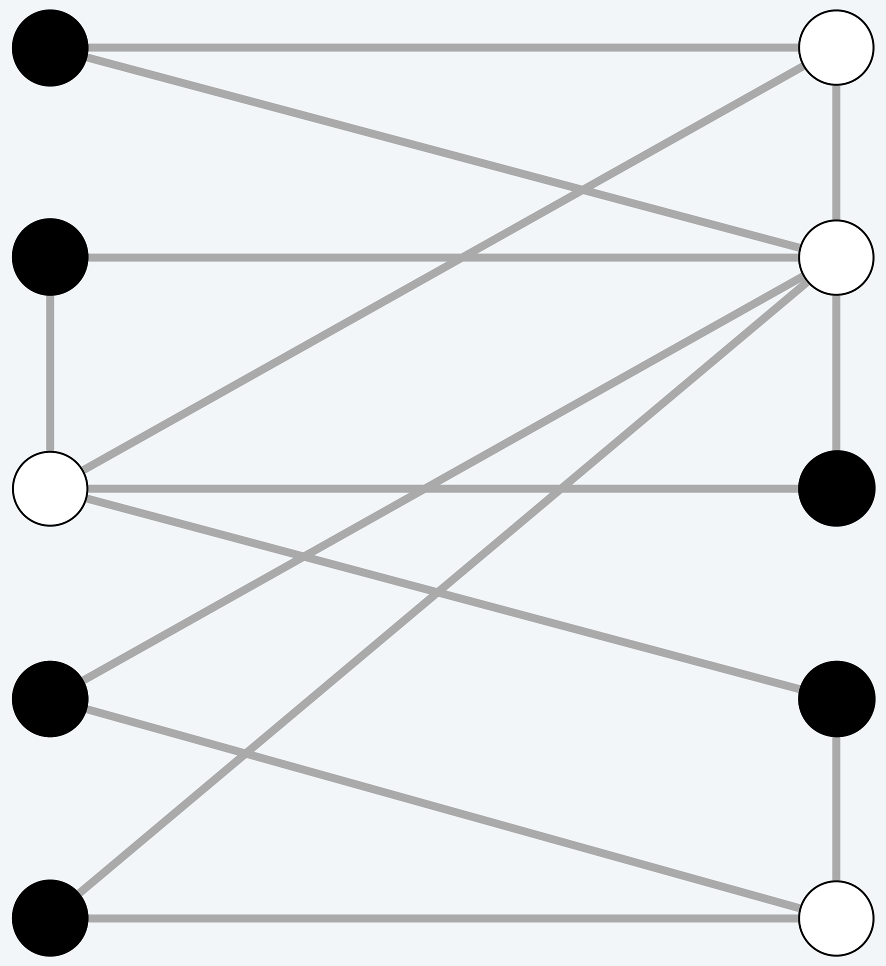
● independent set of size 6

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 ≤ 4 ?

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



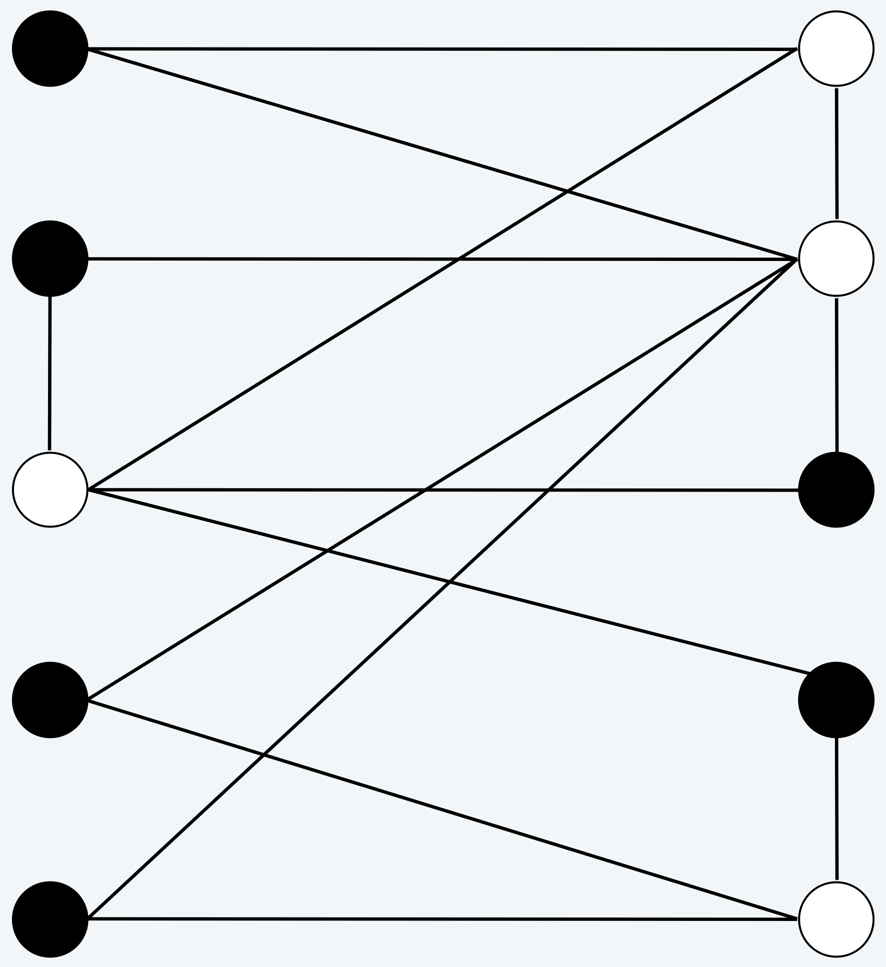
● independent set of size 6

○ vertex cover of size 4

Vertex cover and independent set reduce to one another

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

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



● independent set of size 6

○ vertex cover of size 4

Vertex cover and independent set reduce to one another

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

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

$\Rightarrow$

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

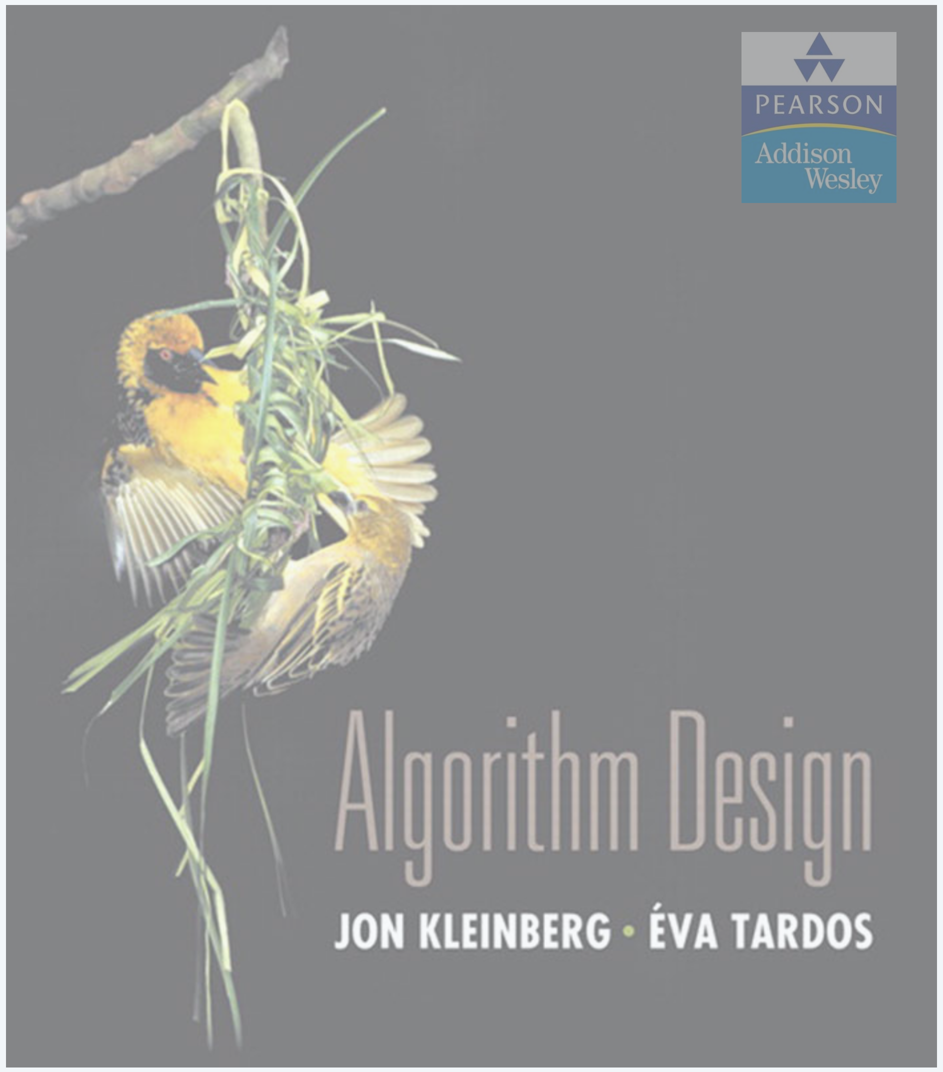
Vertex cover and independent set reduce to one another

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

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

$\Leftarrow$

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



SECTION 8.2

8. INTRACTABILITY I

- *poly-time reductions*
- *packing and covering problems*
- *constraint satisfaction 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 (CNF). A propositional formula Φ that is a conjunction of clauses. $\Phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$

SAT. Given a CNF formula Φ, does it have a satisfying truth assignment?
3-SAT. SAT where each clause contains exactly 3 literals
(and each literal corresponds to a different variable).

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

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

Key application. Electronic design automation (EDA).

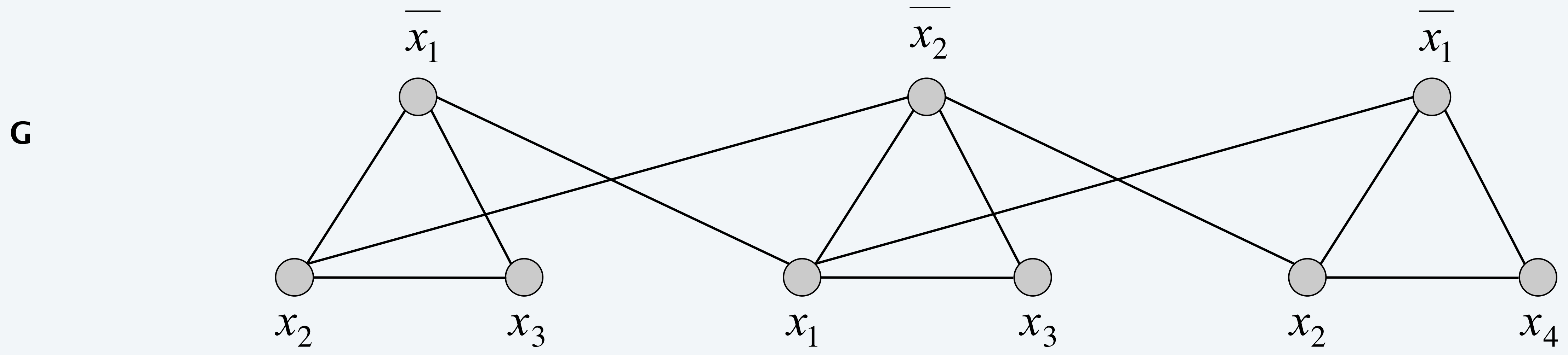
3-satisfiability reduces to independent set

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

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

Construction.

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



k = 3

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

Review

Basic reduction strategies.

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

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

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

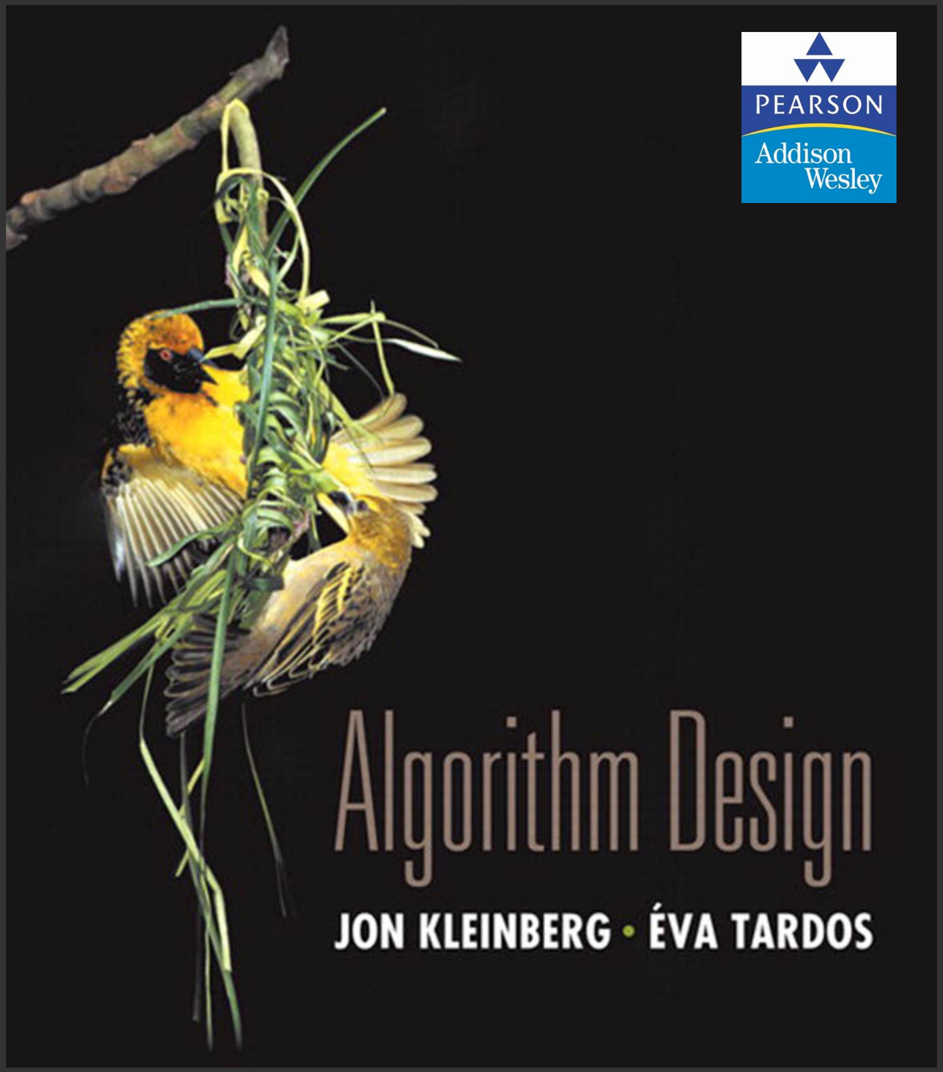
Decision, search, and optimization problems

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

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

Optimization problem. Find a vertex cover of minimum size.

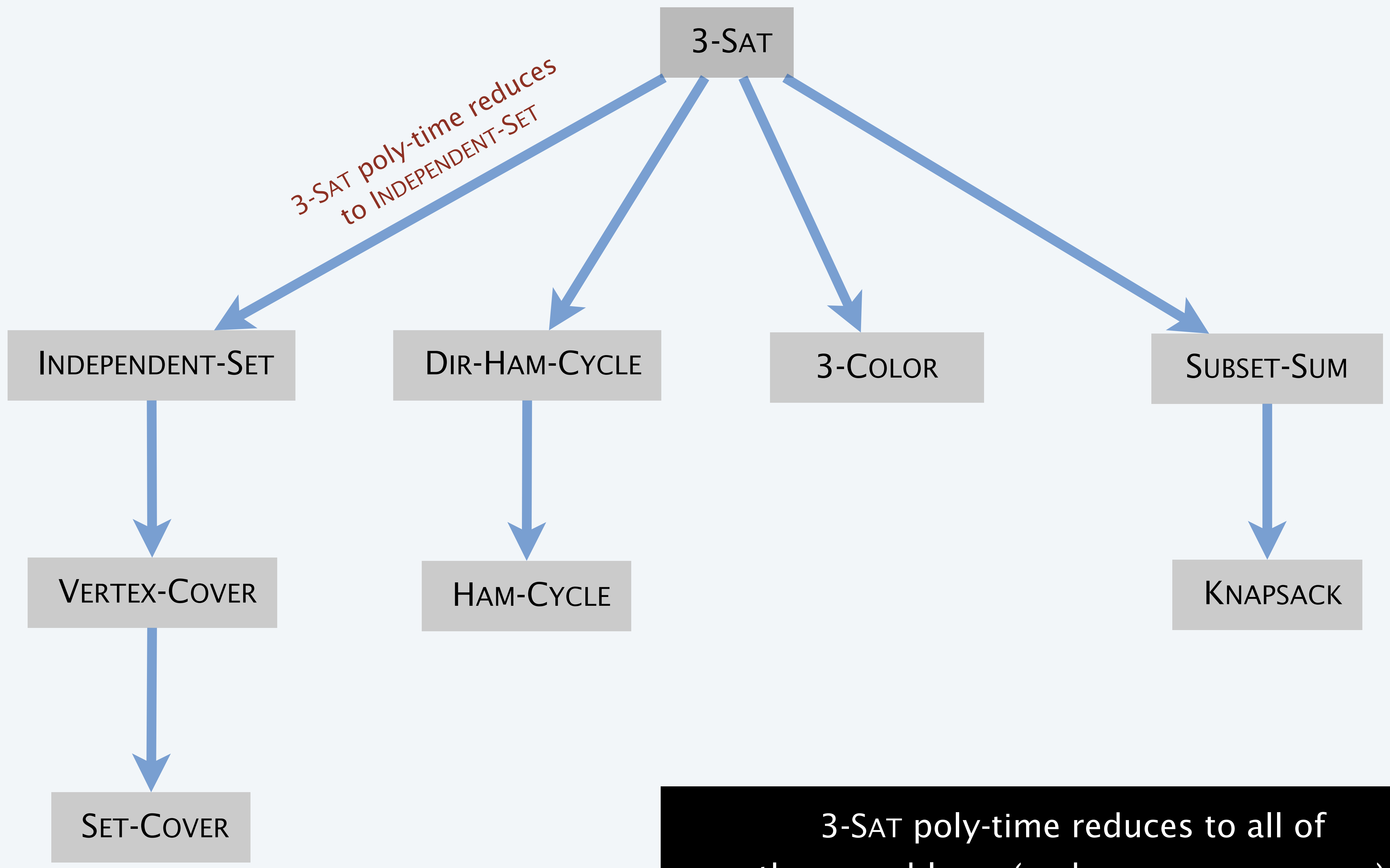
all three problems poly-time reduce to one another.



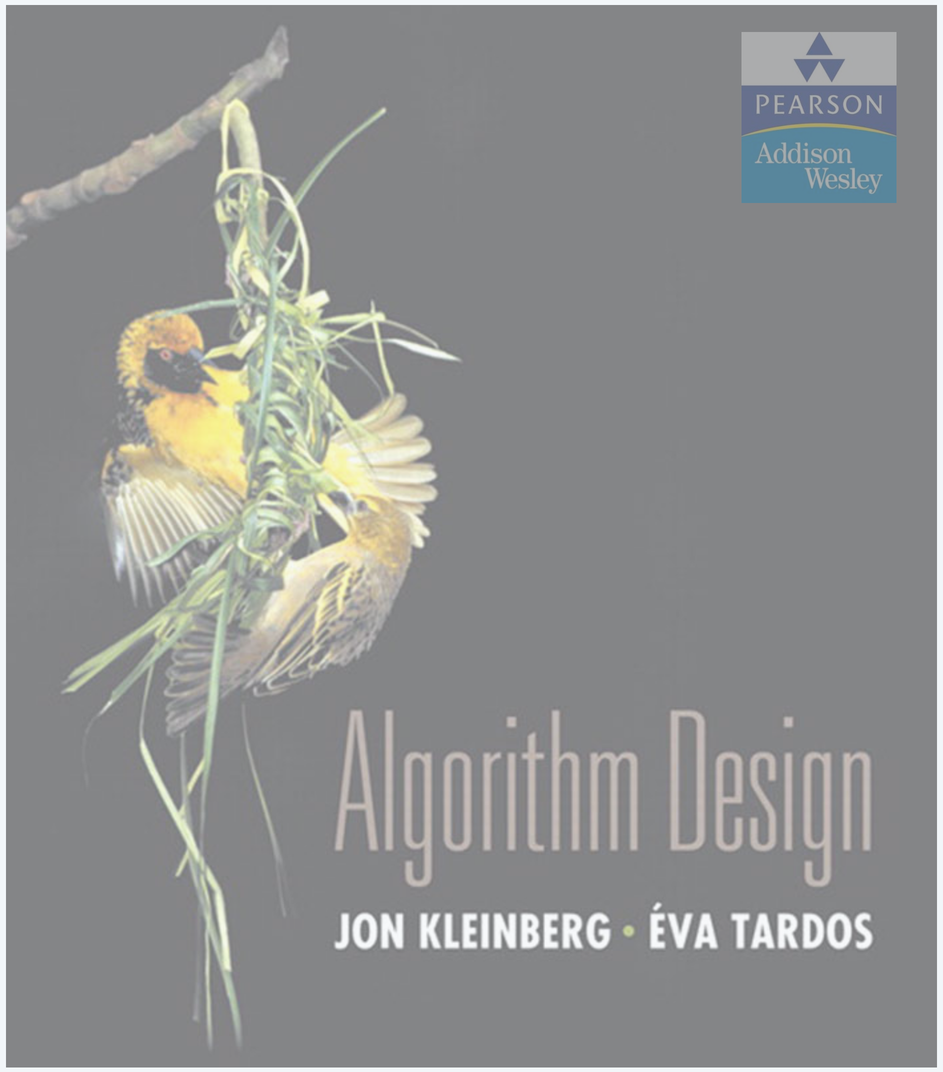
8. Intractability II

- *P vs. NP*
- *NP-complete*

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*

P

Decision problem.

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

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

length of s

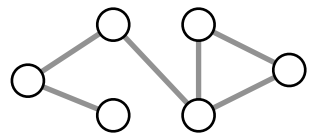
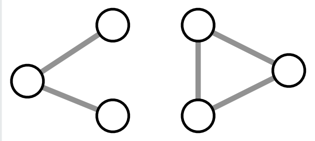
Def. **P** = set of decision problems for which there exists a poly-time algorithm.

on a deterministic Turing machine

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

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		

NP

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

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

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

"certificate" or "witness"

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

Certifiers and certificates: satisfiability

SAT. Given a CNF formula Φ, does it have a satisfying truth assignment?
3-SAT. SAT where each clause contains exactly 3 literals.

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

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

instance s $\Phi = \left(\overline{x_1} \lor x_2 \lor x_3\right) \land \left(x_1 \lor \overline{x_2} \lor x_3\right) \land \left(\overline{x_1} \lor x_2 \lor x_4\right)$

certificate t $x_1 = \mathit{true},\ x_2 = \mathit{true},\ x_3 = \mathit{false},\ x_4 = \mathit{false}$

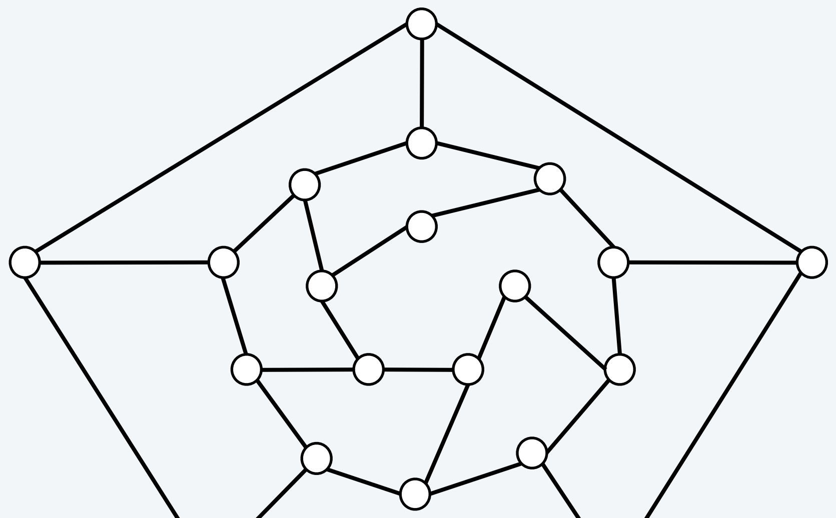
Conclusions. SAT $\in$ **NP**, 3-SAT $\in$ **NP**.

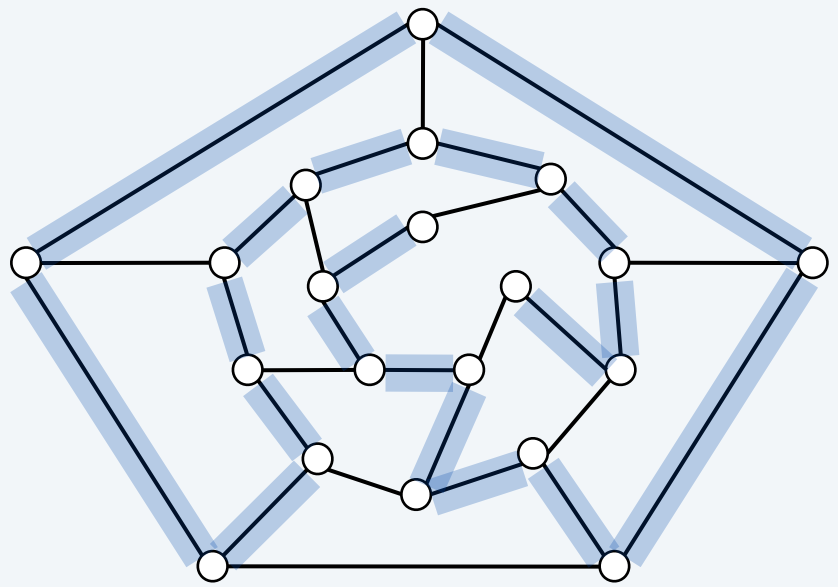
Certifiers and certificates: Hamilton path

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

Certificate. A permutation π of the n nodes.

Certifier. Check that π contains each node in V exactly once, and that G contains an edge between each pair of adjacent nodes.

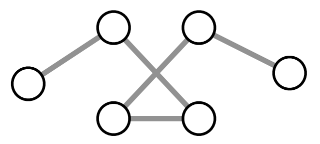
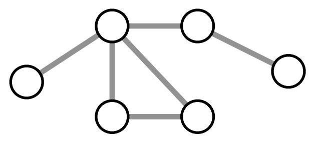

instance s


certificate t

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

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?	???	$\begin{array}{rrr} \neg x_1 \vee & x_2 \vee & \neg x_3 \\ x_1 \vee & \neg x_2 \vee & x_3 \\ \neg x_1 \vee & \neg x_2 \vee & x_3 \end{array}$	$\begin{array}{rr} & \neg x_2 \\ x_1 \vee & x_2 \\ \neg x_1 \vee & x_2 \end{array}$
HAMILTON-PATH	Is there a simple path between u and v that visits every node?	???		

Significance of NP

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

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

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

P, NP, and EXP

P. Decision problems for which there exists a poly-time algorithm.
NP. Decision problems for which there exists a poly-time certifier.
EXP. Decision problems for which there exists an exponential-time algorithm.

Proposition. **P** ⊆ **NP**.
Pf. Consider any problem $X \in$ **P**.

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

Proposition. **NP** ⊆ **EXP**.
Pf. Consider any problem $X \in$ **NP**.

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

Fact. **P** ≠ **EXP** ⇒ either **P** ≠ **NP**, or **NP** ≠ **EXP**, or both.

The main question: P vs. NP

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

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

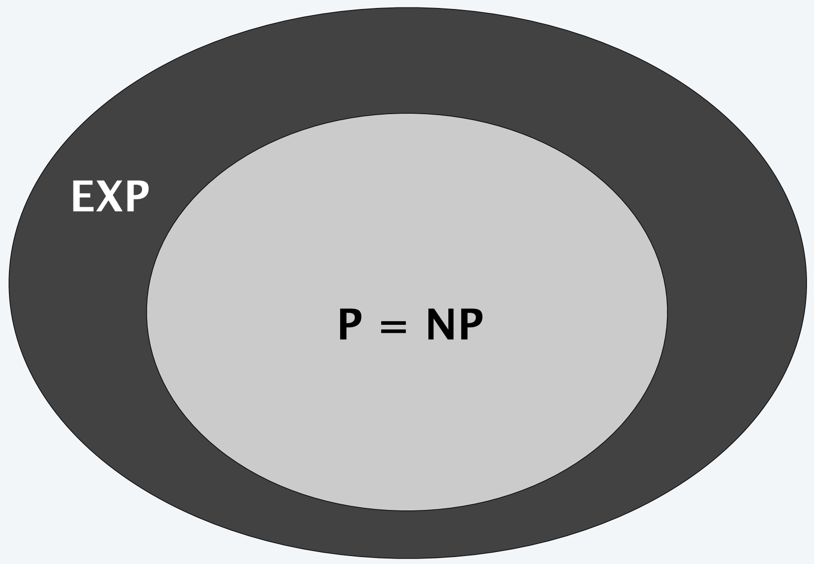
Q. Can we do anything substantially more clever?

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

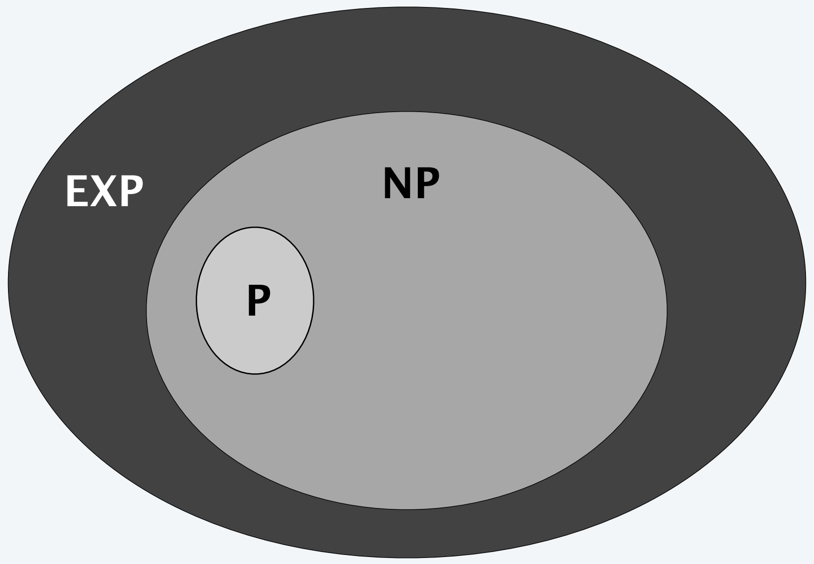
The main question: P vs. NP

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, VERTEX-COVER, FACTOR, ...

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

Consensus opinion. Probably no.

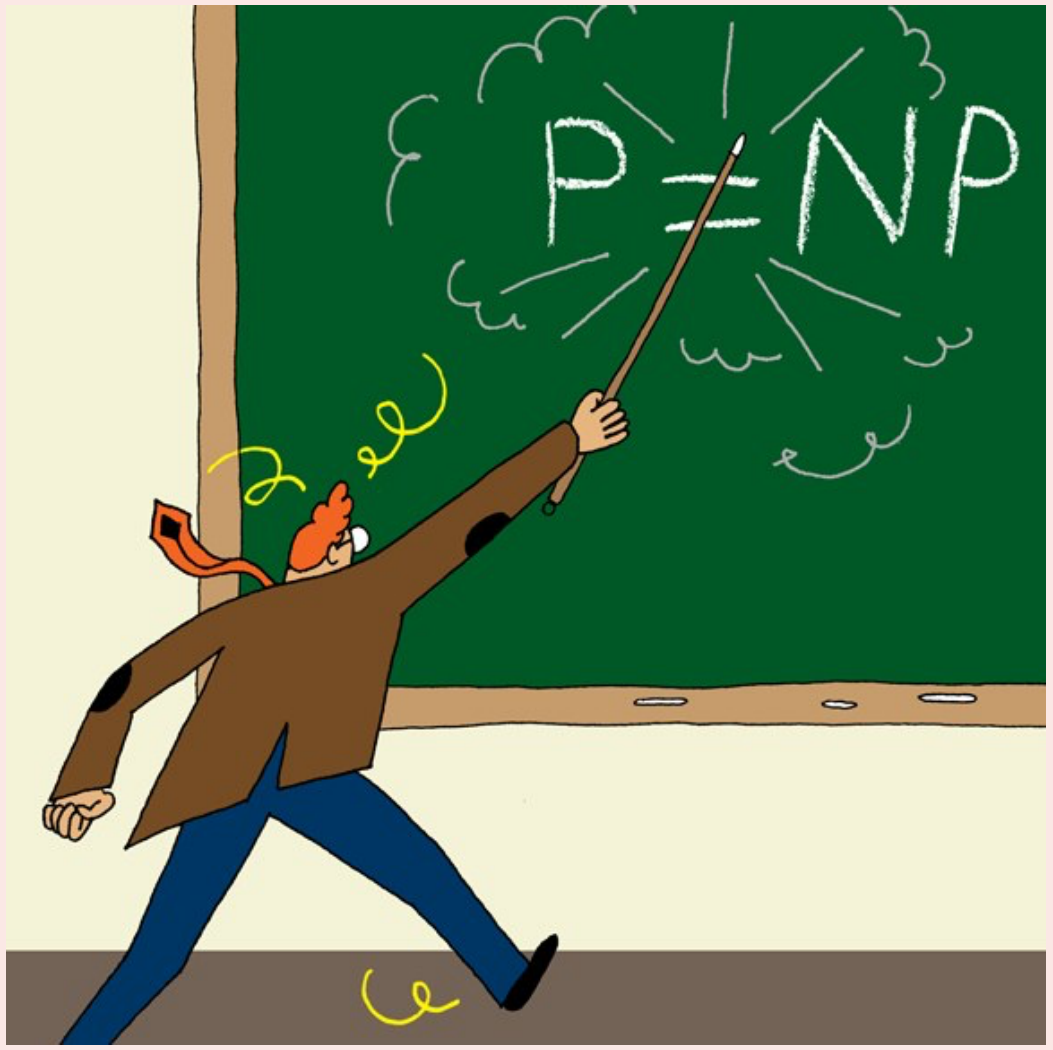
Intractability: quiz 4

Does P = NP?

A. Yes.

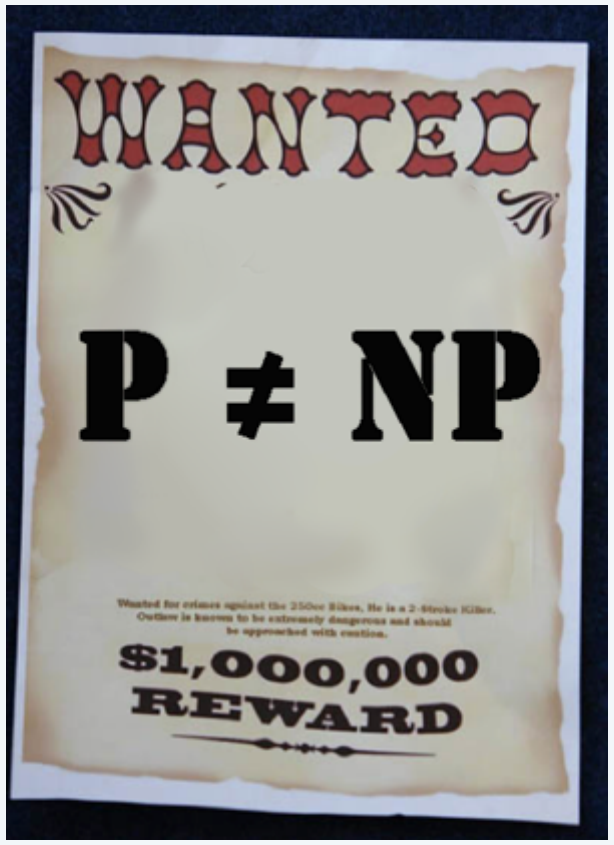
B. No.

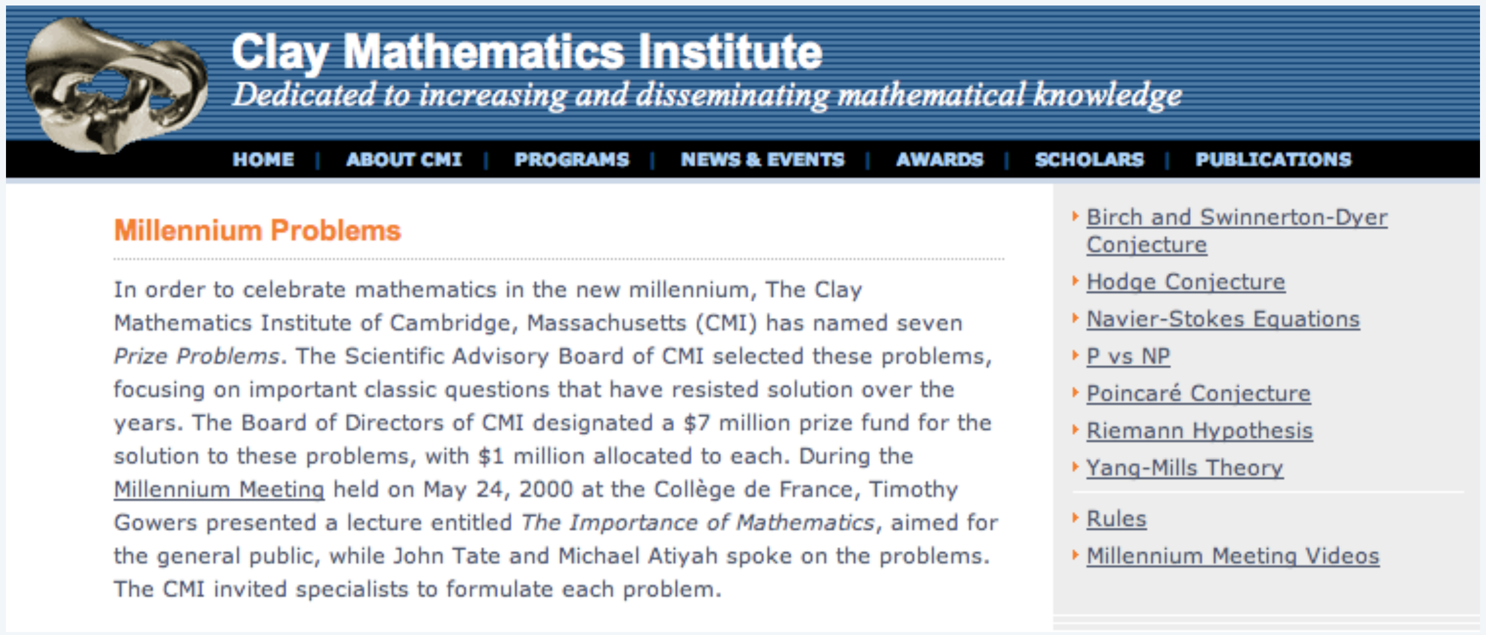
C. None of the above.



Millennium prize

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


WANTED
P ≠ NP
Wanted for crimes against the 250cc Bikes, He is a 2-Stroke Killer.
Outlaw is known to be extremely dangerous and should be approached with caution.
$1,000,000
REWARD


Clay Mathematics Institute
Dedicated to increasing and disseminating mathematical knowledge
HOME | ABOUT CMI | PROGRAMS | NEWS & EVENTS | AWARDS | SCHOLARS | PUBLICATIONS

Millennium Problems

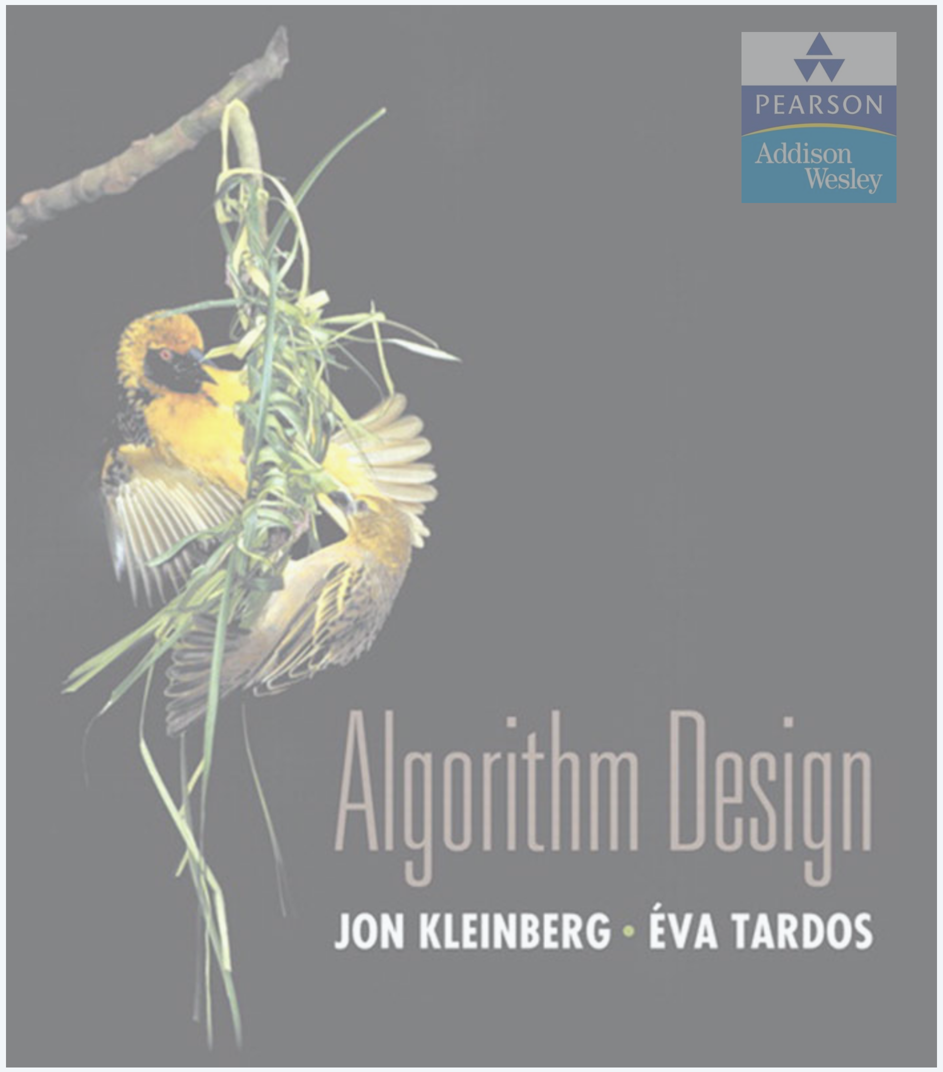
In order to celebrate mathematics in the new millennium, The Clay Mathematics Institute of Cambridge, Massachusetts (CMI) has named seven *Prize Problems*. The Scientific Advisory Board of CMI selected these problems, focusing on important classic questions that have resisted solution over the years. The Board of Directors of CMI designated a $7 million prize fund for the solution to these problems, with $1 million allocated to each. During the Millennium Meeting held on May 24, 2000 at the Collège de France, Timothy Gowers presented a lecture entitled *The Importance of Mathematics*, aimed for the general public, while John Tate and Michael Atiyah spoke on the problems. The CMI invited specialists to formulate each problem.

- Birch and Swinnerton-Dyer Conjecture
- Hodge Conjecture
- Navier-Stokes Equations
- P vs NP
- Poincaré Conjecture
- Riemann Hypothesis
- Yang-Mills Theory

- Rules
- Millennium Meeting Videos

8. Intractability II

- *P vs. NP*
- *NP-complete*



SECTION 8.4

NP-complete

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

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

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

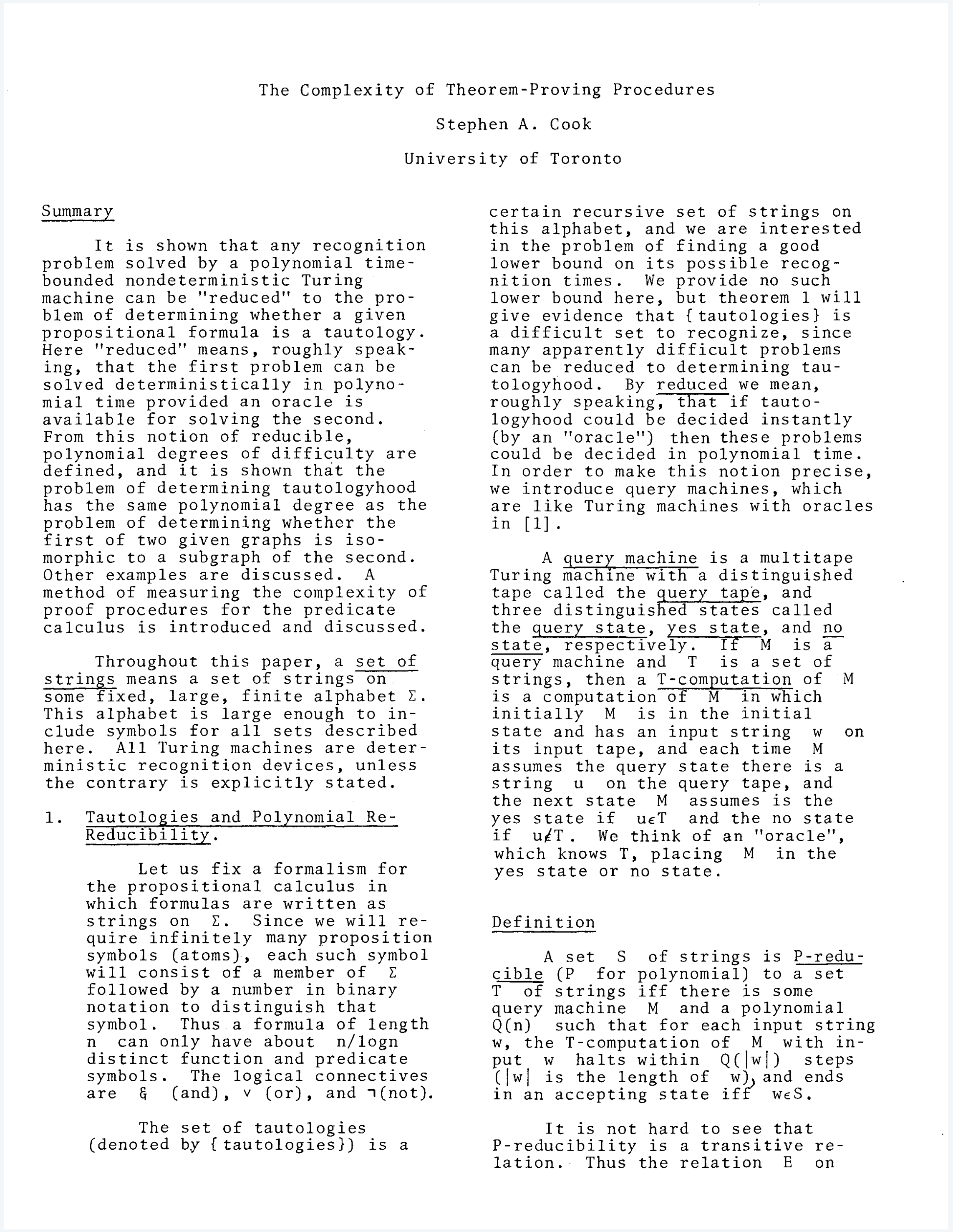
Pf. $\Rightarrow$ Suppose $Y \in$ **P**.

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

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

The “first” NP-complete problem

Theorem. [Cook 1971, Levin 1973] $\textsc{Sat} \in \textbf{NP}$-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 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 $n/\log n$ distinct function and predicate symbols. The logical connectives are & (and), $\vee$ (or), and $\neg$(not).

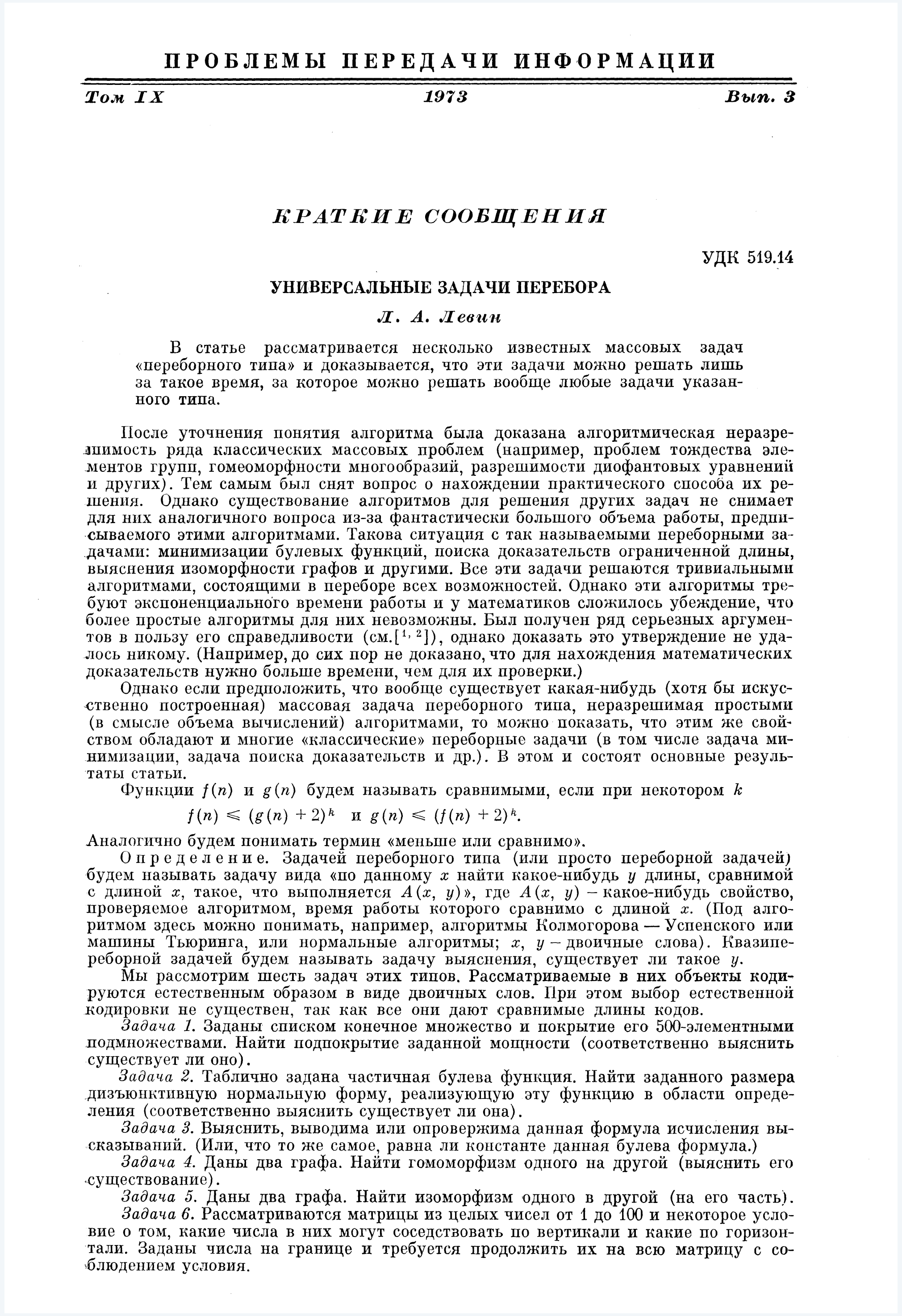
The set of tautologies (denoted by {tautologies}) is a certain recursive set of strings on this alphabet, and we are interested in the problem of finding a good lower bound on its possible recognition times. We provide no such lower bound here, but theorem 1 will give evidence that {tautologies} is a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

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

Establishing NP-completeness

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

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

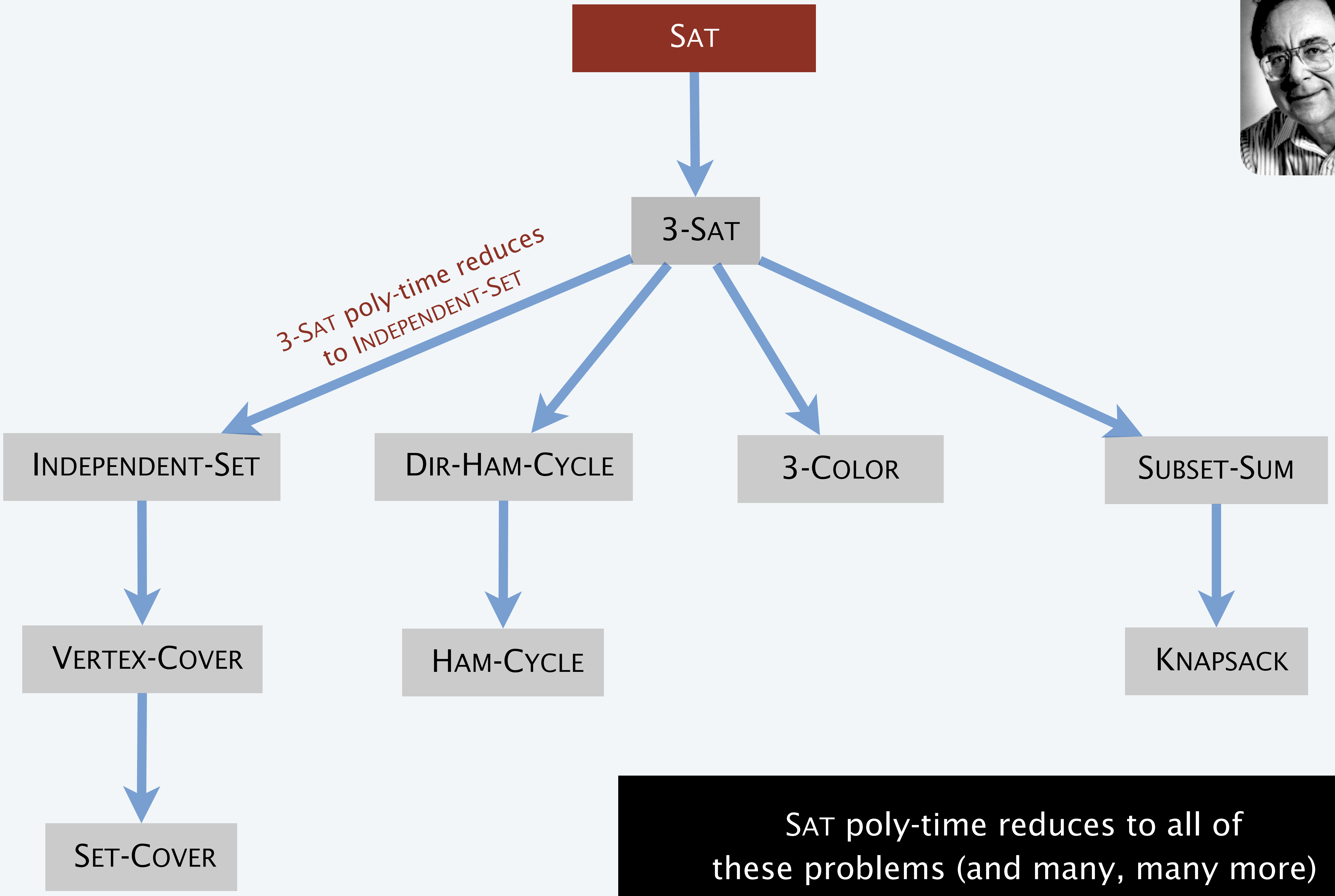
- Step 1. Show that $Y \in$ **NP**.
- Step 2. Choose an **NP**-complete problem X.
- Step 3. Prove that $X \le_P Y$.

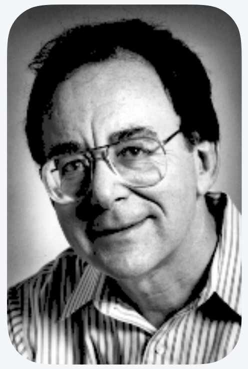
Proposition. If $X \in$ **NP**-complete, $Y \in$ **NP**, and $X \le_P Y$, then $Y \in$ **NP**-complete.

Pf. Consider any problem $W \in$ **NP**. Then, both $\underset{\substack{\uparrow \\ \text{by definition of} \\ \textbf{NP}\text{-complete}}}{W \le_P X}$ and $\underset{\substack{\uparrow \\ \text{by assumption}}}{X \le_P Y}$.

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

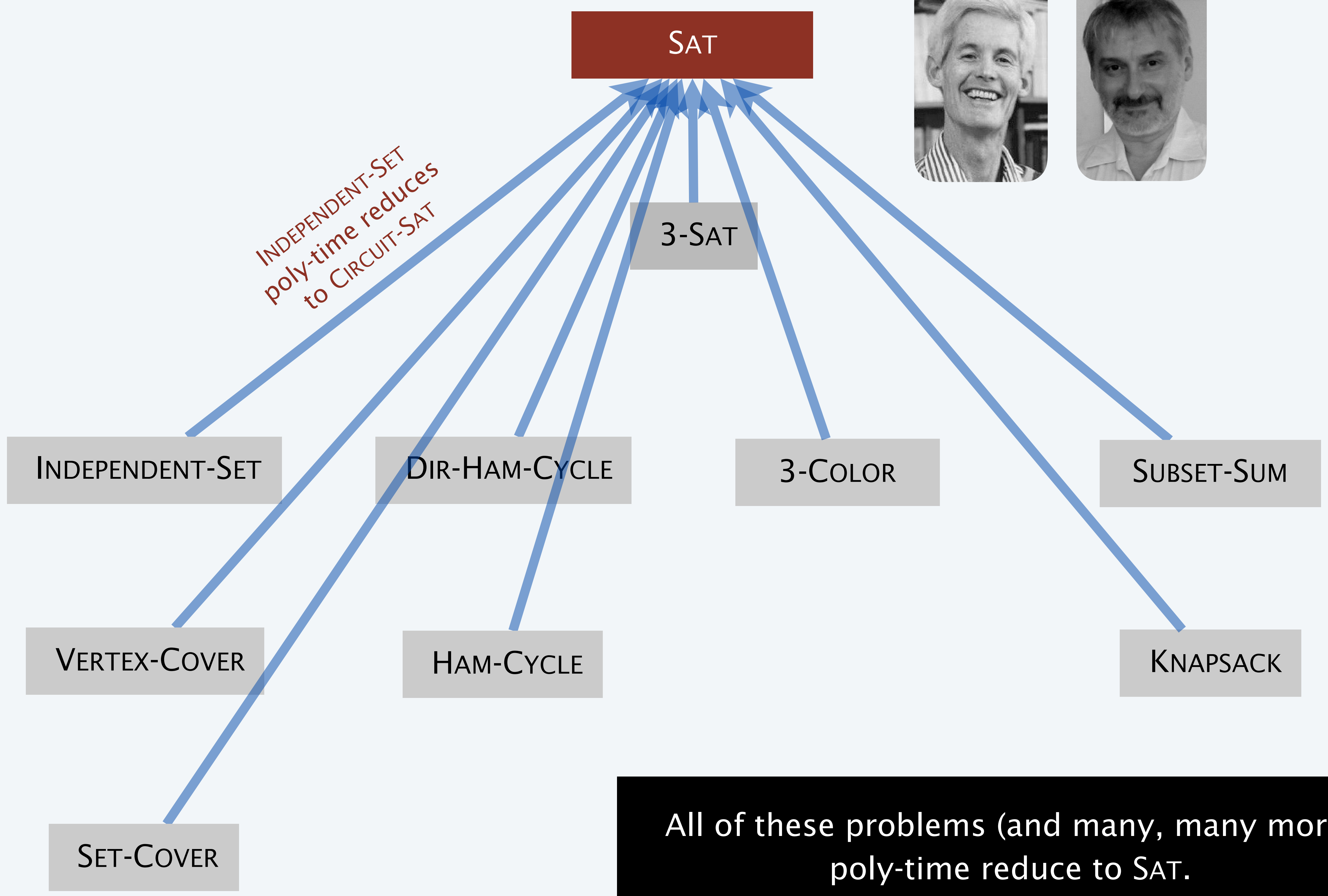
Implications of Karp





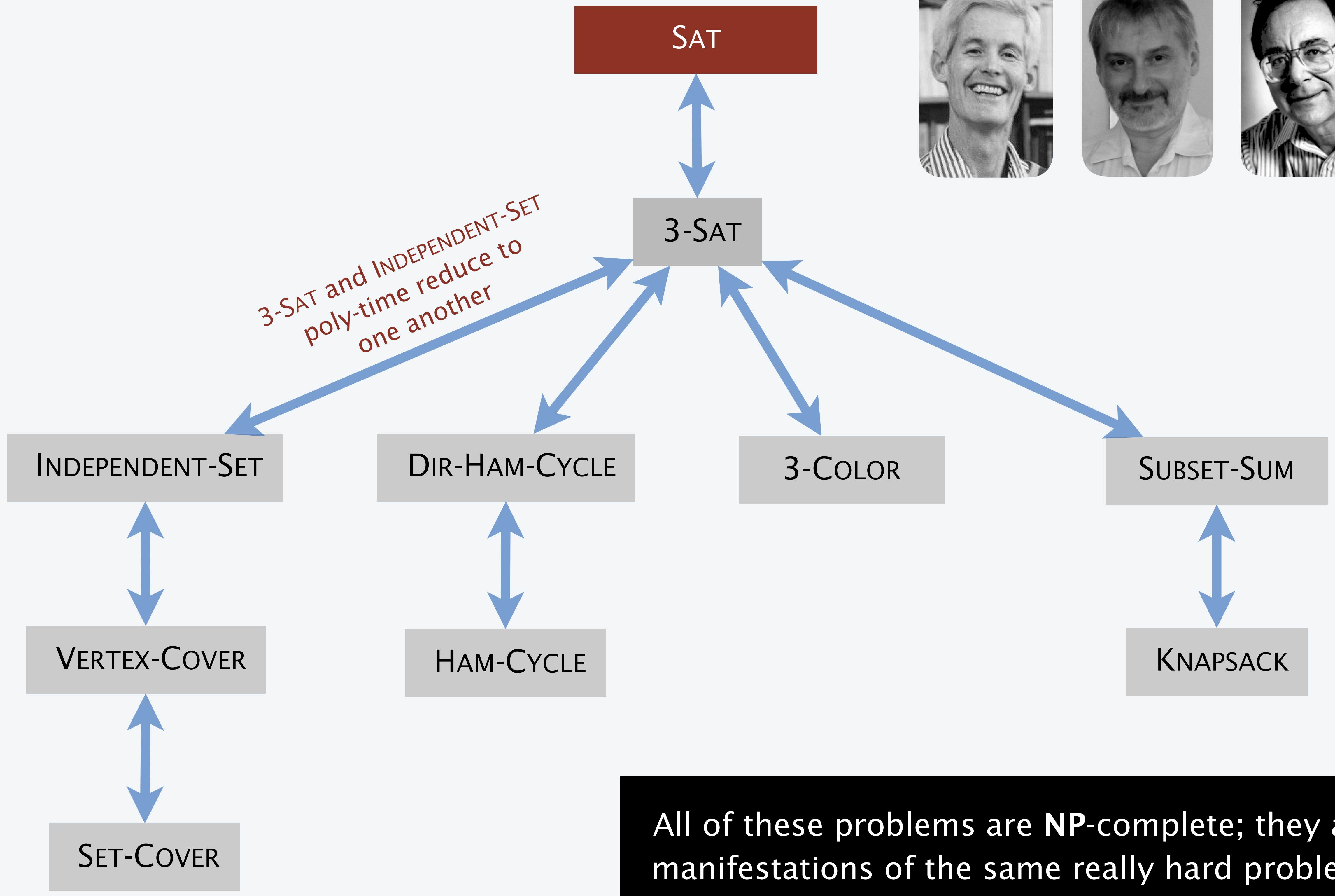
SAT poly-time reduces to all of these problems (and many, many more)

Implications of Cook–Levin



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

Implications of Karp + Cook–Levin



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

I'D TELL YOU ANOTHER
NP-COMPLETE JOKE,
BUT ONCE YOU'VE HEARD
ONE,

YOU'VE HEARD THEM
ALL.