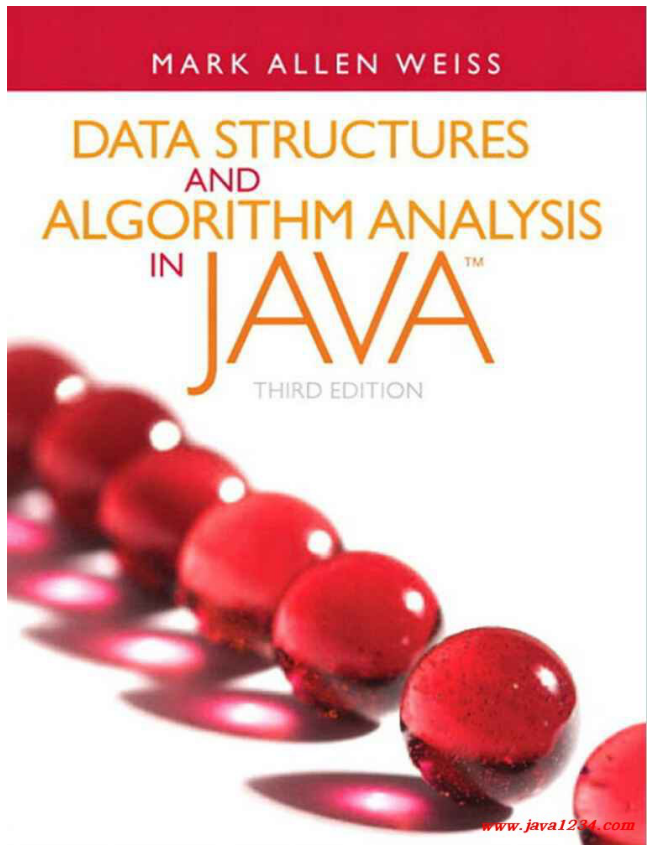
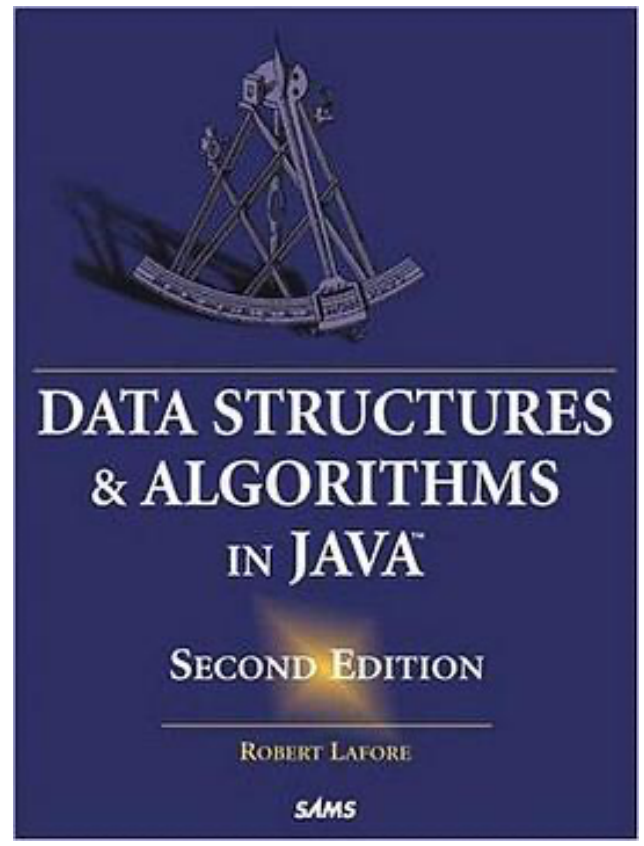
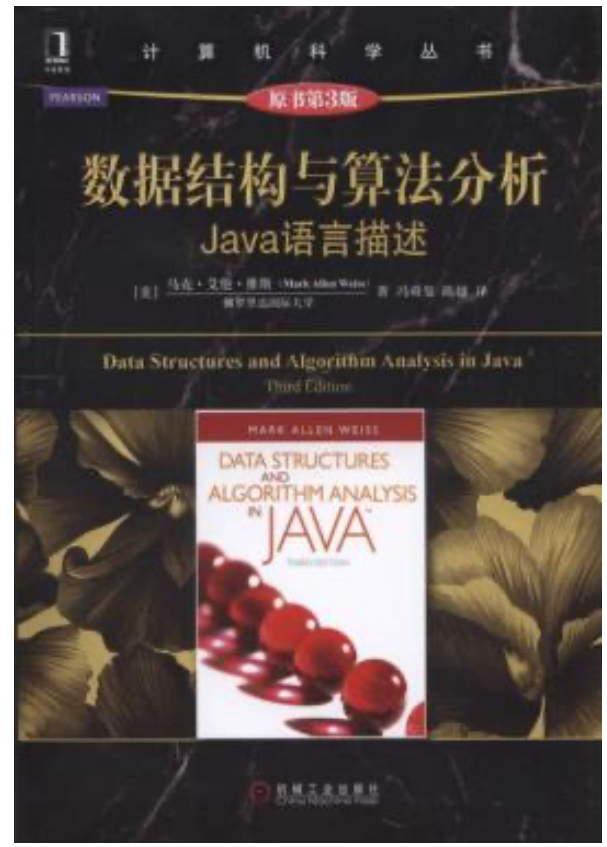
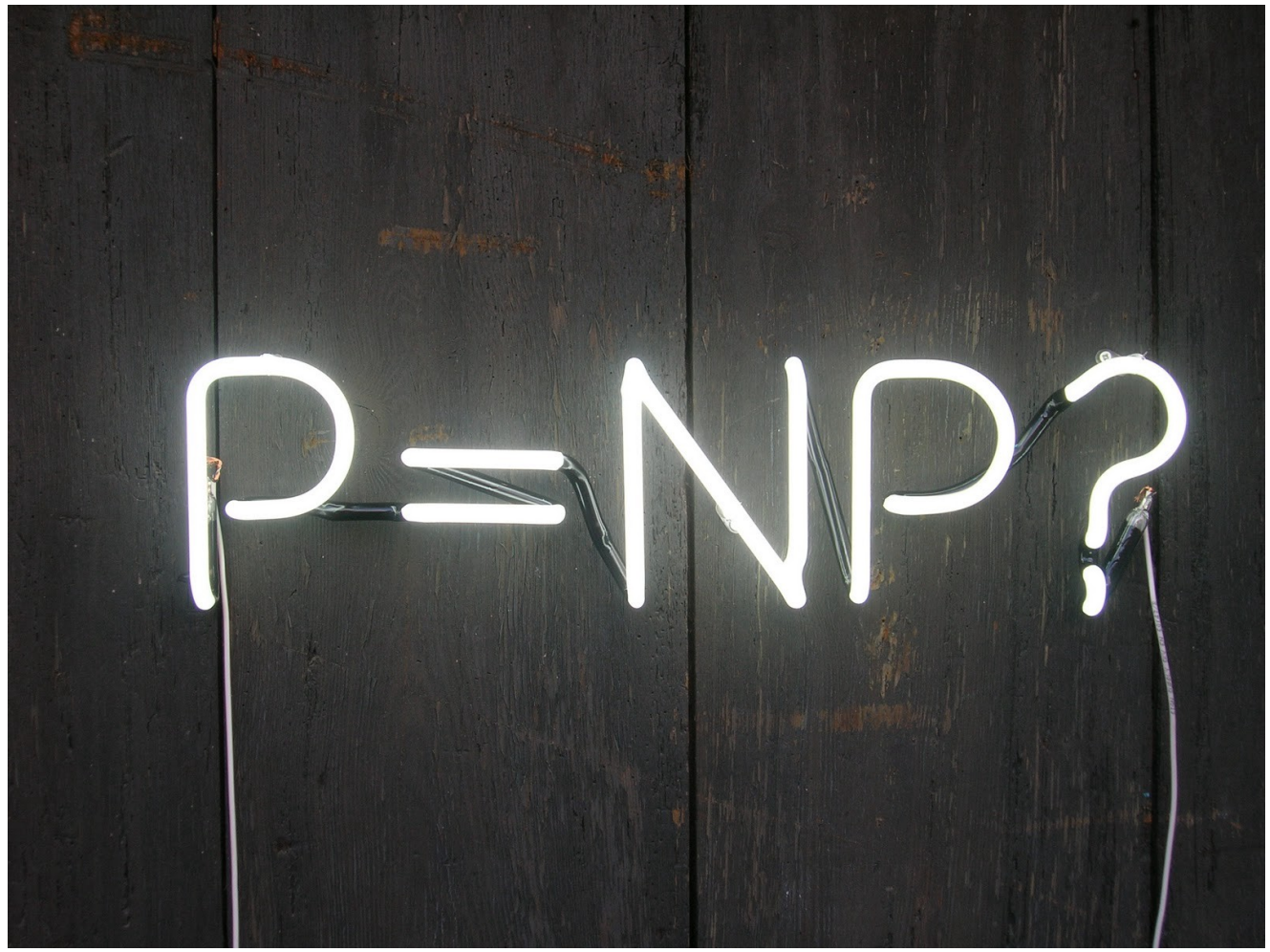
Topic 20 –
NP Problem







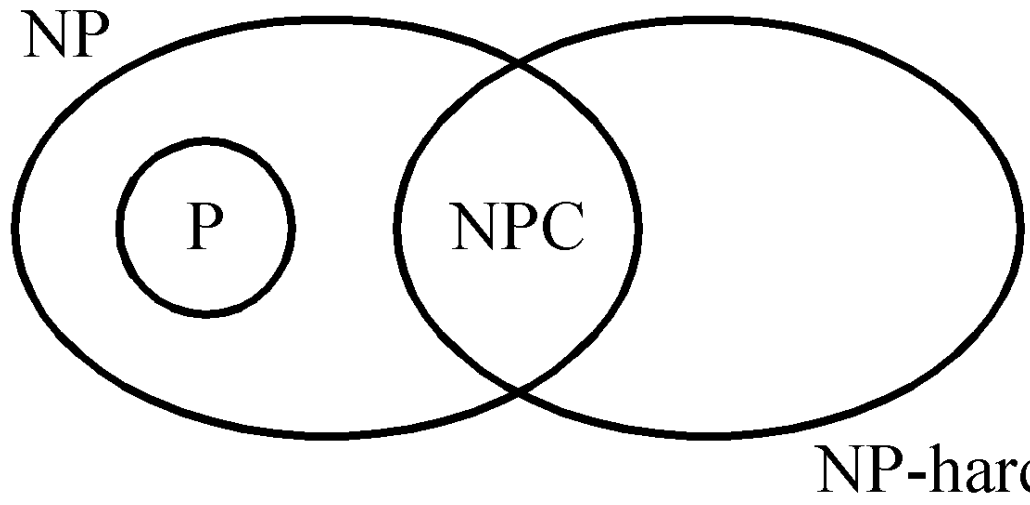
“Can every problem whose solution can be quickly verified by a computer also be quickly solved by a computer?”.

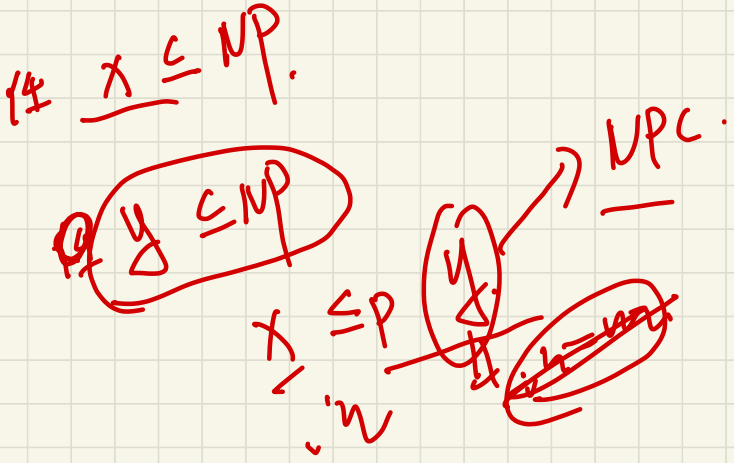


- *P vs. NP*
- *NP-complete*
- *co-NP*
- *NP-hard*

P, NP

- **P**: the class of problems which can be solved by a deterministic polynomial algorithm.
- **NP** : the class of decision problem which can be solved by a non-deterministic polynomial algorithm.
- **NP-hard**: the class of problems to which every NP problem reduces.
- **NP-complete (NPC)**: the class of problems which are NP-hard and belong to NP.



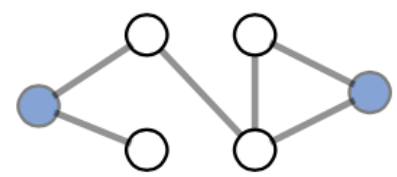
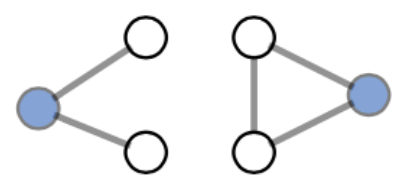

任 $X \in NP$.
$Y \in NP$
$X \leq_P Y$
$Y \in NPC$.

Decision problems

- Decision problem.
  - Problem X is a set of strings.
  - Instance s is one string.
  - Algorithm A solves problem X: $A(s) = yes$ if and only if $s \in X$.
- Definition:
  - 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.
    - Note. |s| = length of s
- Ex.
  - Problem PRIMES = { 2, 3, 5, 7, 11, 13, 17, 23, 29, 31, 37, ... }.
  - 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 $s \in X$.
- **Definition.** 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 $C(s, t) = yes$.
  - Note t is called "certificate" or "witness"
- **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 if and only if s is composite. Moreover $|t| \leq |s|$.
- **Certifier.** Check that $1 < t < s$ and that s is a multiple of t.

- **Example.**
  - **instance s** 437669
  - **certificate t** 541 or 809
  - 437,669 = 541 × 809

- **Conclusion.** COMPOSITES $\in$ **NP**.

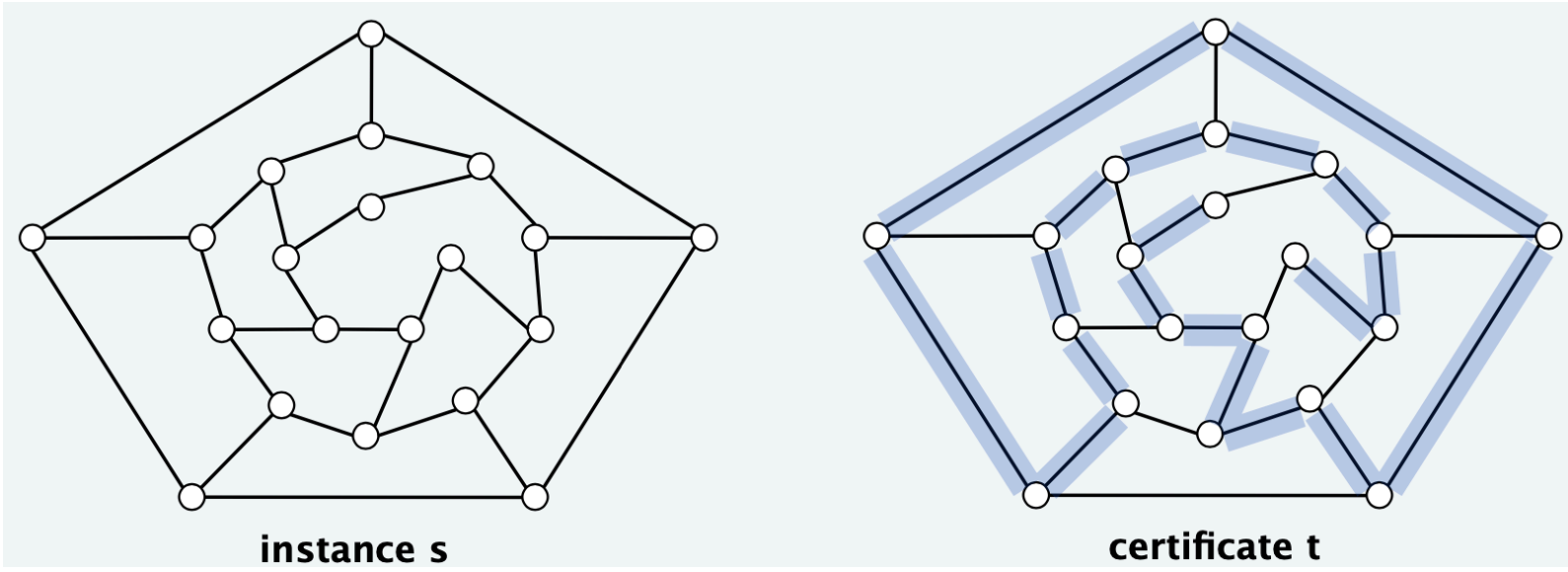
Certifiers and certificates: 3-satisfiability

- **3-SAT.** Given a CNF formula Φ, is there a satisfying assignment?
- **Certificate.** An assignment of truth values to the *n* boolean variables.
- **Certifier.** Check that each clause in Φ has at least one true literal.
- Example.
  - **instances** Φ $=(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_4)$
  - **certificate t** $x_1 = true, x_2 = true, x_3 = false, x4 = false$

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

Certifiers and certificates: Hamilton path

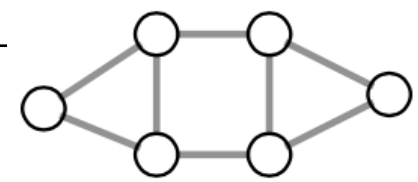
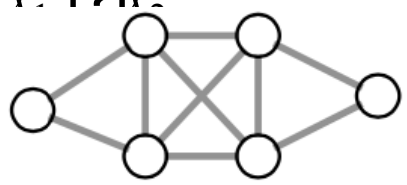
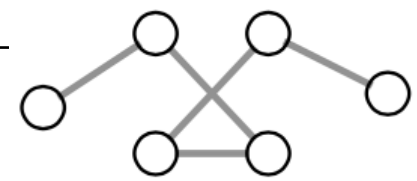
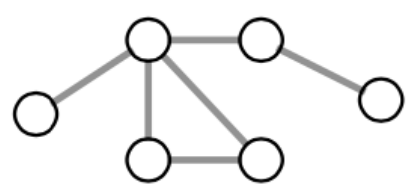
- **HAM-PATH.** Given an undirected graph $G = (V, E)$, does there exist a simple path P that visits every node?
- **Certificate.** A permutation of the n nodes.
- **Certifier.** Check that the permutation contains each node in V exactly once, and that there is an edge between each pair of adjacent nodes.



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

Definition of NP

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

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 ?	?	$\neg x_1 \; ? \; x_2$ $x_1 \; ? \; x_2$	$\neg x_2$ $\neg x_1 \; ? \; x_2$ $x_1 \; ? \; x_2$
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?	?		

Decision problems

- **Decision problems**. The solution is simply "Yes" or "No".
- Optimization problems are more difficult.

- e.g. the traveling salesperson problem (TSP)
  - Optimization version:
    - Find the shortest tour
  - Decision version:
    - Is there a tour whose total length is less than or equal to a constant c ?

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 $C(s, t) = A(s)$.

P, NP, and EXP

- **NP.** Decision problems for which there is a poly-time certifier.
- **EXP.** Decision problems for which there is an exponential-time algorithm.
- 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 $|t| \leq p(|s|)$.
  - Return *yes* if $C(s, t)$ returns *yes* for any of these potential certificates.
- Remark. Time-hierarchy theorem implies **P** $\subsetneq$ **EXP**.

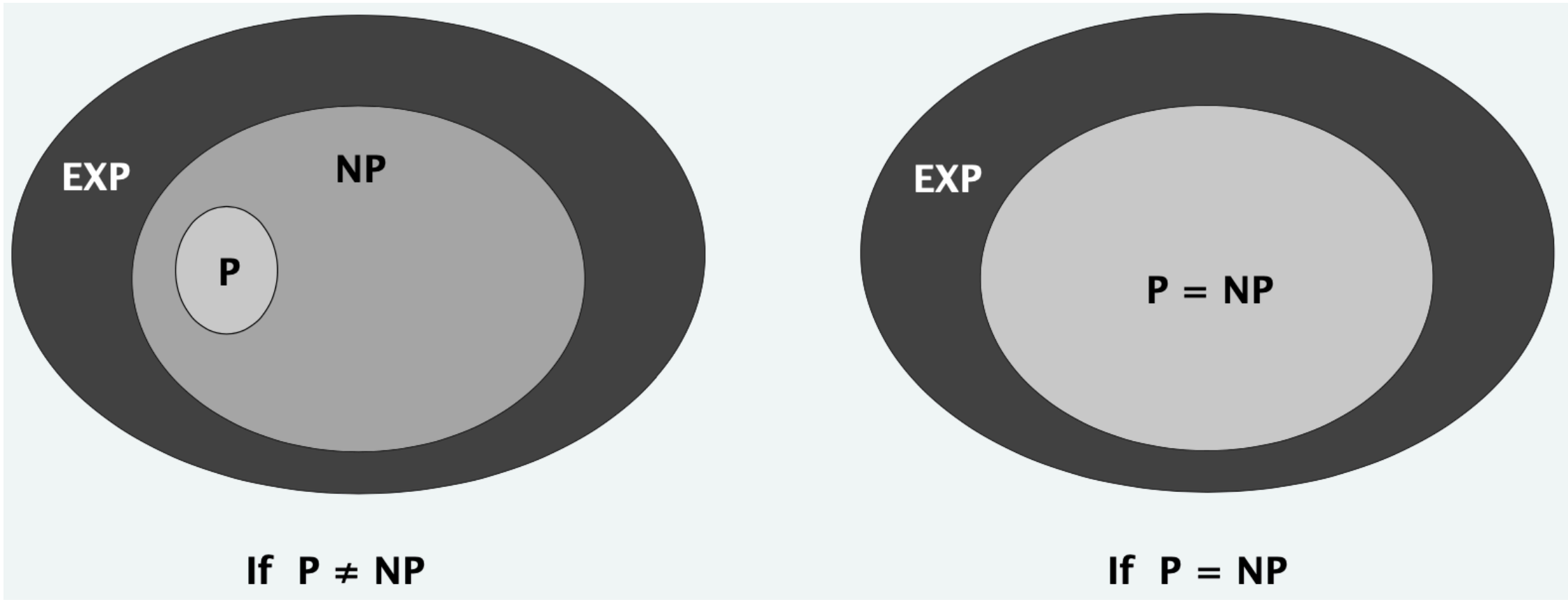
The main question: P vs. NP

- Q. How to solve an instance of 3-SAT with n variables?
- A. Exhaustive search: try all $2n$ truth assignments.

- Q. Can we do anything substantially more clever?
- Conjecture. No poly-time algorithm 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 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 ≠ NP.

“ I conjecture that there is no good algorithm for the traveling salesman problem. My reasons are the same as for any mathematical conjecture:
(i) It is a legitimate mathematical possibility and (ii) I do not know.”
— Jack Edmonds 1966

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

Possible outcomes

- $P \neq NP$.

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

Possible outcomes

- P = NP.

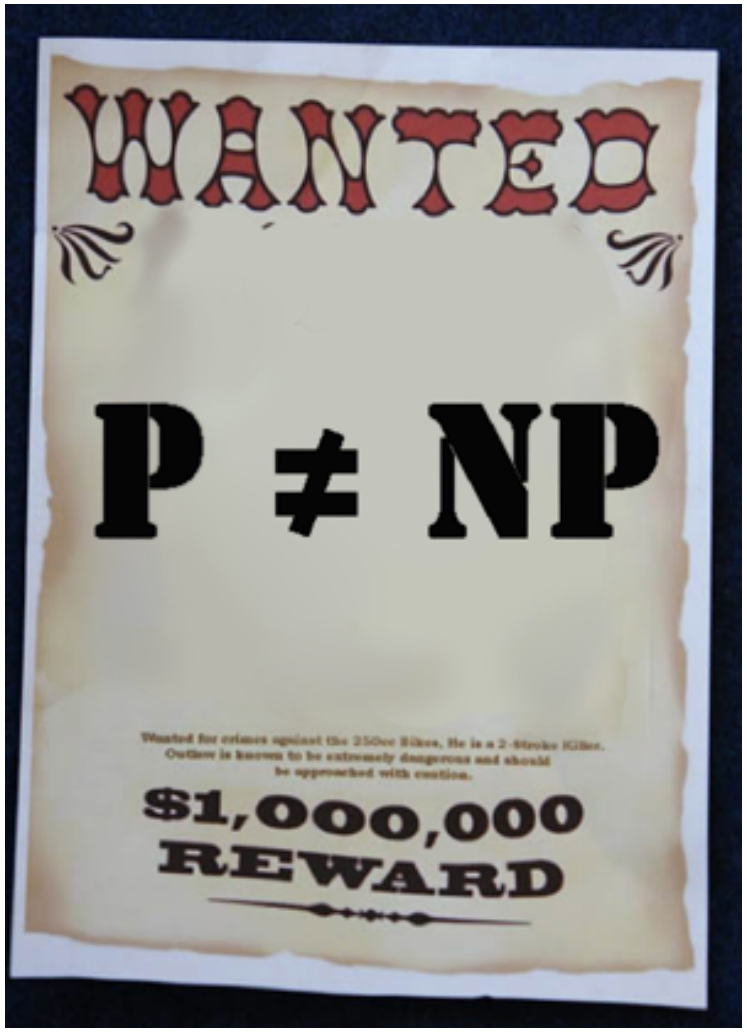
> *“ P = NP. In my opinion this shouldn’t really be a hard problem; it’s just that we came late to this theory, and haven’t yet developed any techniques for proving computations to be hard. Eventually, it will just be a footnote in the books. ” — John Conway*

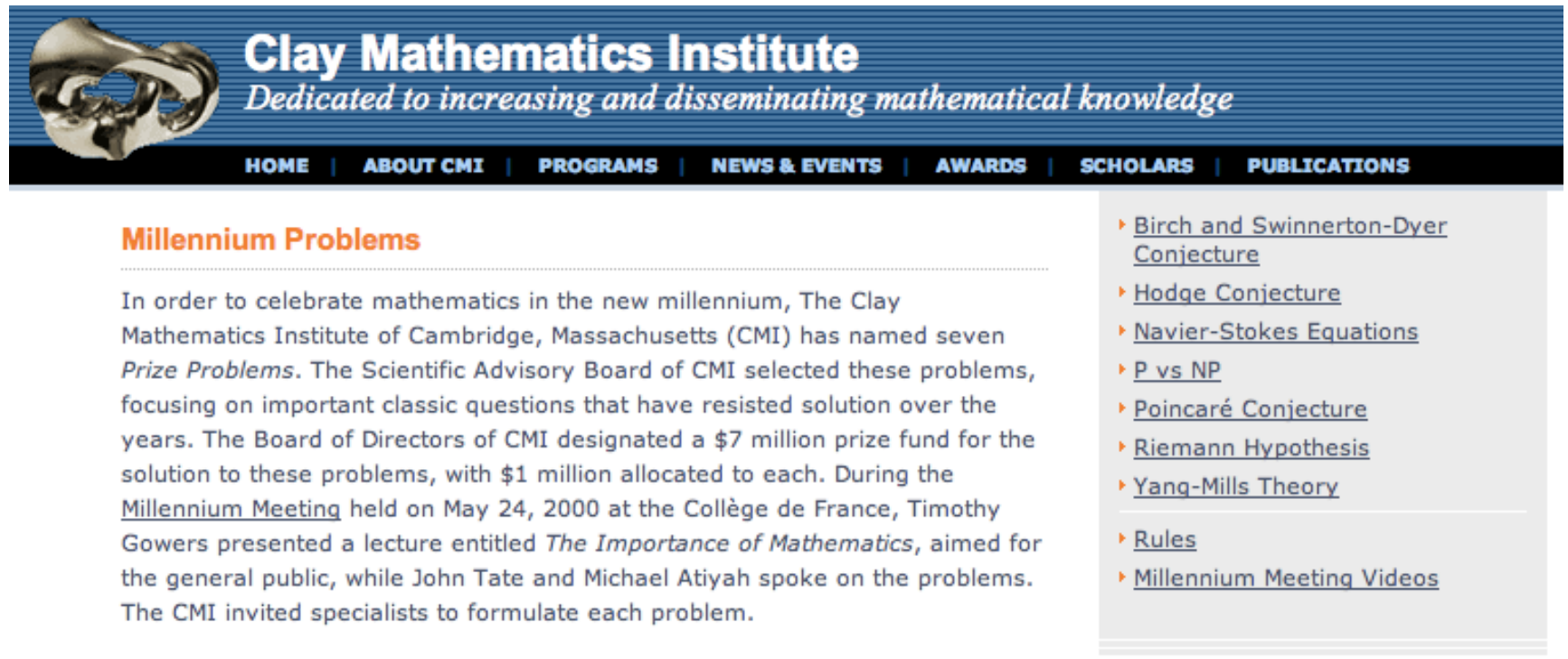
Other possible outcomes

- P = NP, but only $\Omega(n100)$ 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





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

Polynomial transformation

- Definition.
  - 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*.
- Definition.
  - 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* if and only if *y* is a *yes* instance of *Y*.
    - we require |y| to be of size polynomial in |x|

Polynomial transformation

- Definition.
  - 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* if and only if *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.

Polynomial transformation

- Definition.
  - 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*.
- Definition.
  - 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* if and only if *y* is a *yes* instance of *Y*.
    - we require |y| to be of size polynomial in |x|
- **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**, $X \leq_p Y$.

- Definition of reduction:
  - Problem A reduces to problem B (A $\leq_p$ B) if and only if A can be solved by a deterministic polynomial time algorithm using a deterministic algorithm that solves B in polynomial time.

Reduction

- Up to now, none of the NPC problems can be solved by a deterministic polynomial time algorithm in the worst case.
- It does not seem to have any polynomial time algorithm to solve the NPC problems.
- The theory of NP-completeness always considers the worst case.
- The lower bound of any NPC problem seems to be in the order of an exponential function.
- Not all NP problems are difficult. (e.g. the MST problem is an NP problem.)
- If A, B $\in$ NPC, then A $\leq_p$ B and B $\leq_p$ A.

P = NP ?

- Theorem.
  - Suppose $Y \in$ **NP**-complete. Then $Y \in$ **P** if and only if **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**.

P = NP ?

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

Cook's theorem

NP = P iff the satisfiability problem is a P problem.

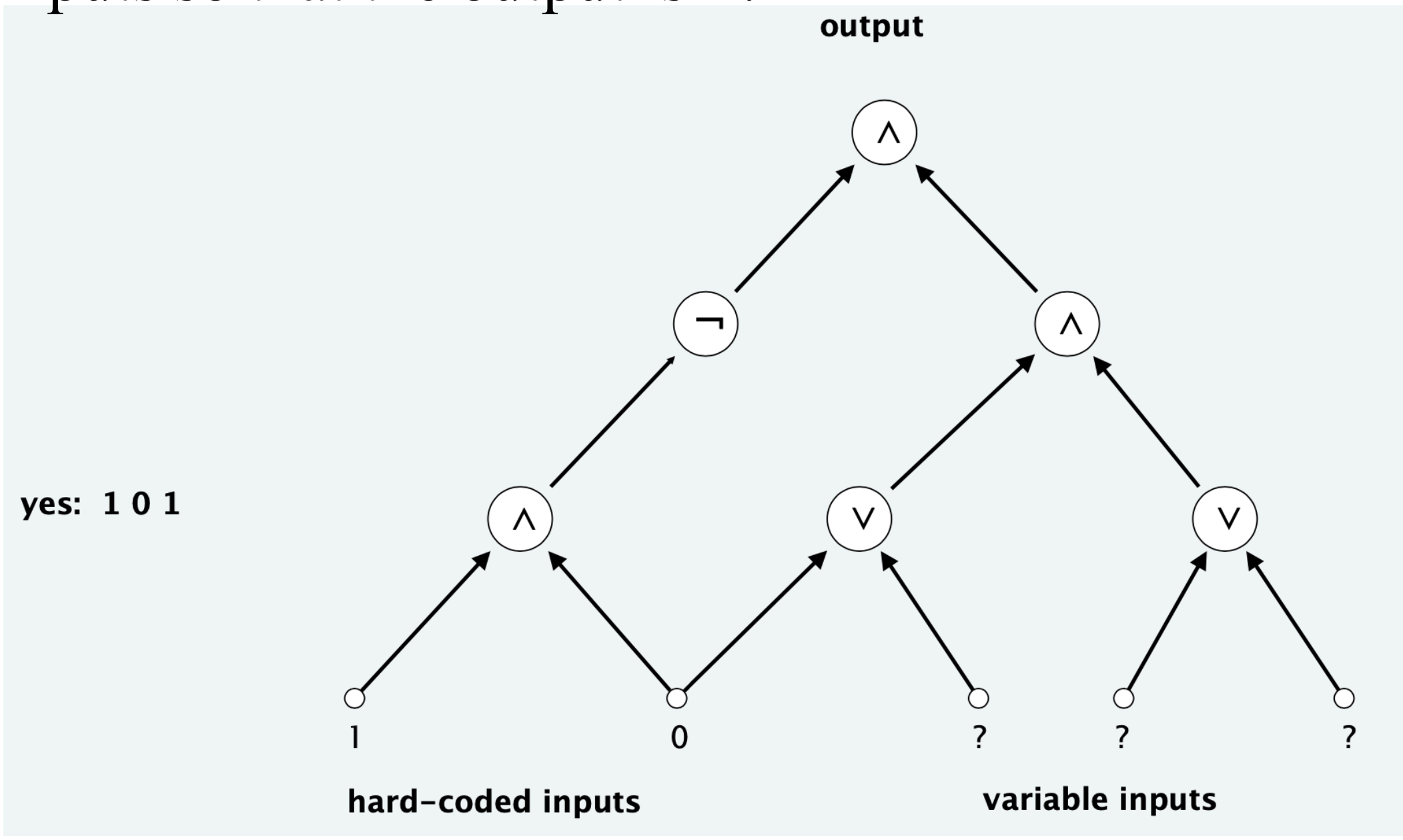
- SAT is NP-complete.
- It is the first NP-complete problem.
- Every NP problem reduces to SAT.

Stephen Arthur Cook

(1939~)

Circuit satisfiability

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



The "first" NP-complete problem

- Theorem. CIRCUIT-SAT $\in$ **NP**-complete. [Cook 1971, Levin 1973]

The Complexity of Theorem-Proving Procedures

Stephen A. Cook

University of Toronto

Summary

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

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

1. Tautologies and Polynomial Reducibility.

Let us fix a formalism for the propositional calculus in which formulas are written as strings on Σ. Since we will require infinitely many proposition symbols (atoms), each such symbol will consist of a member of Σ followed by a number in binary notation to distinguish that symbol. Thus a formula of length n can only have about $n/\log n$ distinct function and predicate symbols. The logical connectives are $\&$ (and), $\vee$ (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».

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 в другой (на его часть).

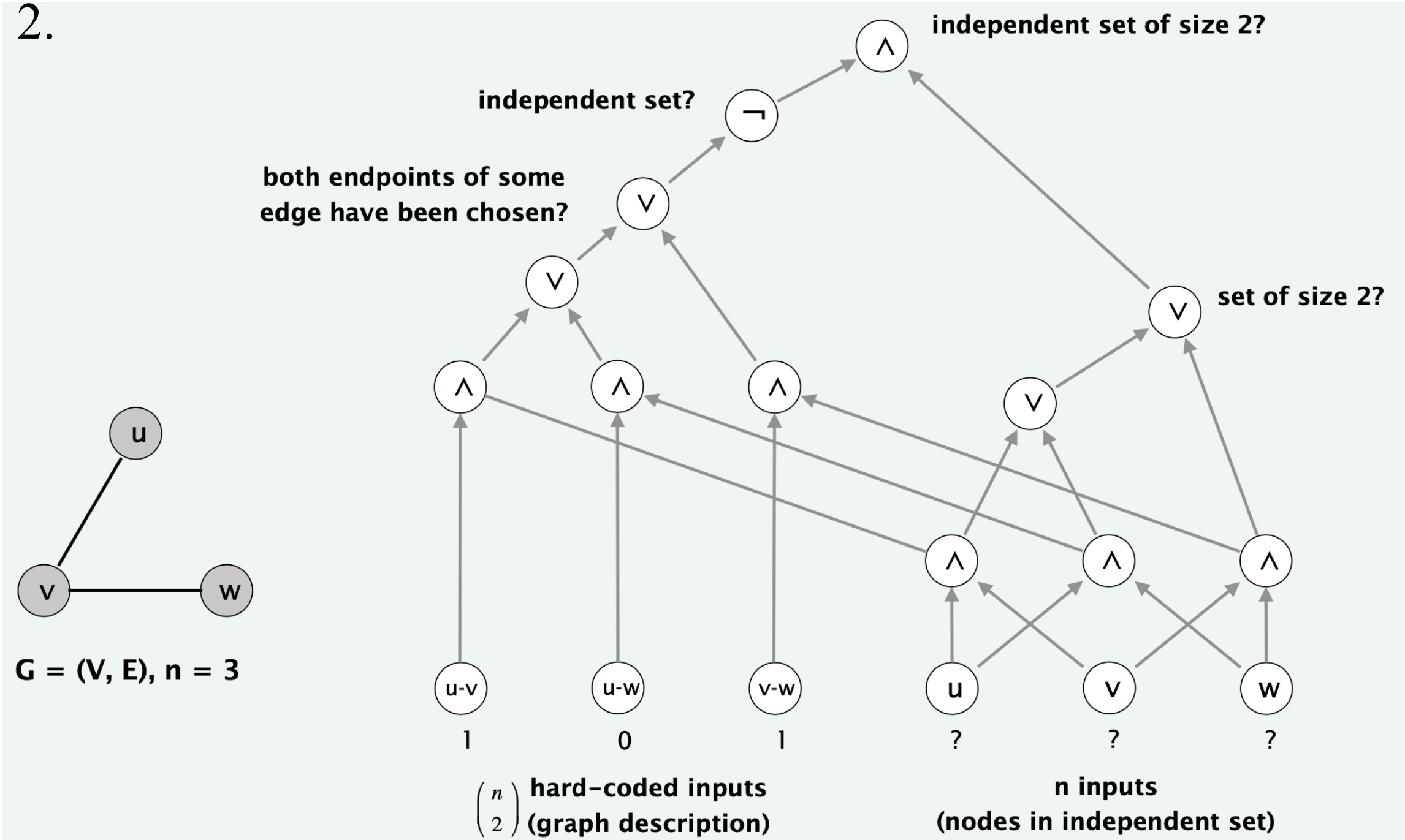
Задача 6. Рассматриваются матрицы из целых чисел от 1 до 100 и некоторое условие о том, какие числа в них могут соседствовать по вертикали и какие по горизонтали. Заданы числа на границе и требуется продолжить их на всю матрицу с соблюдением условия.

The "first" NP-complete problem

- **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 $C(s, t)$:
    $s \in X$ if and only if f 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

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



Proof of NP-Completeness

- To show that A is NP-complete

  (I) Prove that A is an NP problem.

  (II) Prove that $\exists$ B $\in$ NPC, B $\leq_p$ A.

  $\Rightarrow$ A $\in$ NPC

- Why ?

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 $X \leq_p Y$.

Establishing NP-completeness

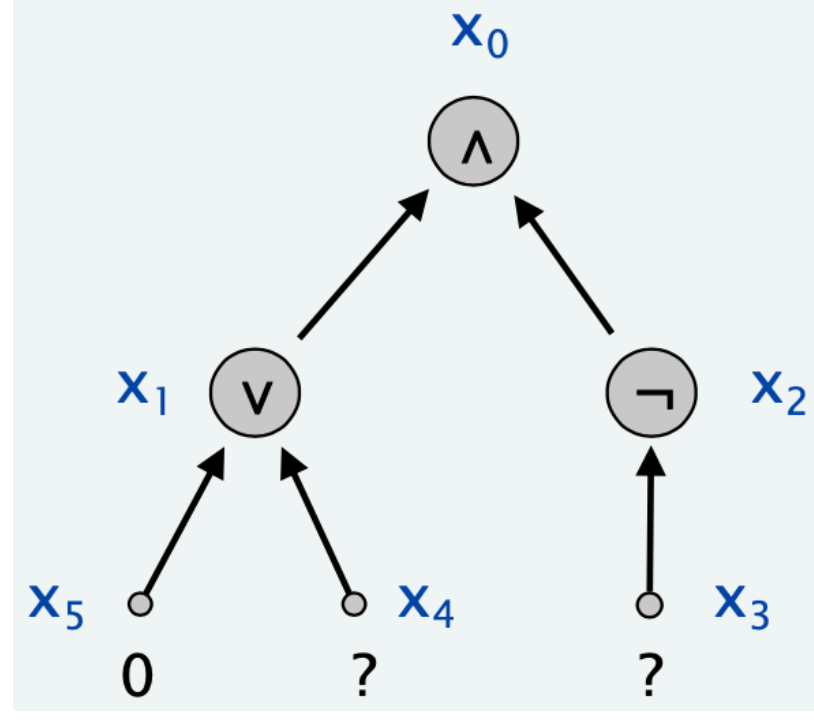
- Theorem.
  - If $X \in$ **NP**-complete, $Y \in$ **NP**, and $X \leq_p Y$, then $Y \in$ **NP**-complete.
- Pf.
  - Consider any problem $W \in$ **NP**.
  - Then, both $W \leq_p X$ and $X \leq_p Y$.
    - *Note*
      - $W \leq_p X$: by definition of
      - $X \leq_p Y$: by assumption NP-complete
  - 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 Φ of 3-SAT that is satisfiable if and only if 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 *xi* 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 clauses: $\overline{x_5}$
  - $x_0 = 1 \Rightarrow$ add 1 clauses: x_0

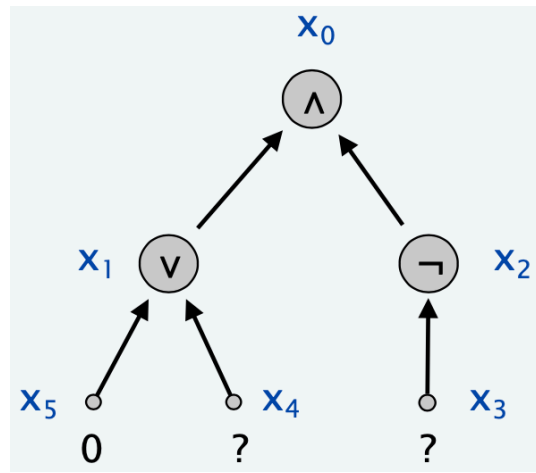


3-satisfiability is NP-complete

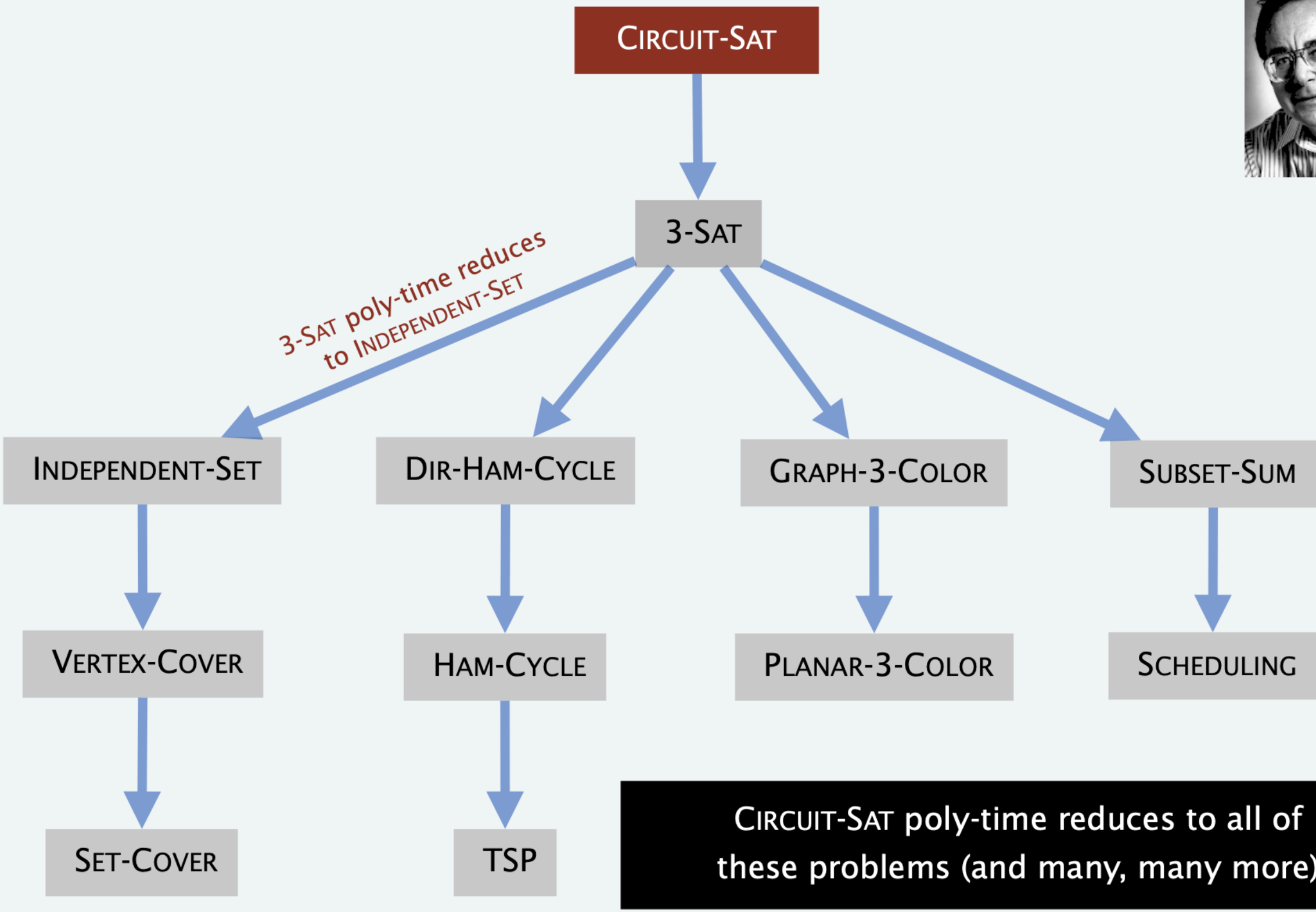
- Construction. [continued]
- Step 4. Turn clauses of length 1 or 2 into clauses of length 3.
- Create four new variables z_1, z_2, z_3, and z_4.
- Add 8 clauses to force $z_1 = z_2 =$ *false*:
  - $(\overline{z_1} \vee z_3 \vee z_4)$, $(\overline{z_1} \vee \overline{z_3} \vee z_4)$, $(\overline{z_1} \vee z_3 \vee \overline{z_4})$, $(\overline{z_1} \vee \overline{z_3} \vee \overline{z_4})$, $(\overline{z_2} \vee z_3 \vee z_4)$, $(\overline{z_2} \vee \overline{z_3} \vee z_4)$, $(\overline{z_2} \vee z_3 \vee \overline{z_4})$, $(\overline{z_2} \vee \overline{z_3} \vee \overline{z_4})$
- Replace any clause with a single term (t_i) with $(t_i \vee z_1 \vee z_2)$.
- Replace any clause with two terms $(t_i \vee t_j)$ with $(t_i \vee t_j \vee z_1)$.

3-satisfiability is NP-complete

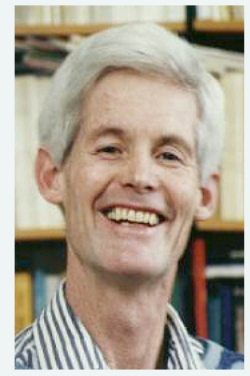
- **Lemma.** Φ is satisfiable if and only if the inputs of *K* can be set so that it outputs 1.
- **Pf.** ⇐ Suppose there are inputs of *K* that make it output 1.
  - Can propagate input values to create values at all nodes of *K*.
  - This set of values satisfies Φ.
- **Pf.** ⇒ Suppose Φ is satisfiable.
  - We claim that the set of values corresponding to the circuit inputs constitutes a way to make circuit *K* output 1.
  - The 3-SAT clauses were designed to ensure that the values assigned to all node in *K* exactly match what the circuit would compute for these nodes.



Implications of Karp



Implications of Cook-Levin



CIRCUIT-SAT

3-SAT poly-time reduces to INDEPENDENT-SET

3-SAT

INDEPENDENT-SET

DIR-HAM-CYCLE

GRAPH-3-COLOR

SUBSET-SUM

VERTEX-COVER

HAM-CYCLE

PLANAR-3-COLOR

SCHEDULING

SET-COVER

TSP

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

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: 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.

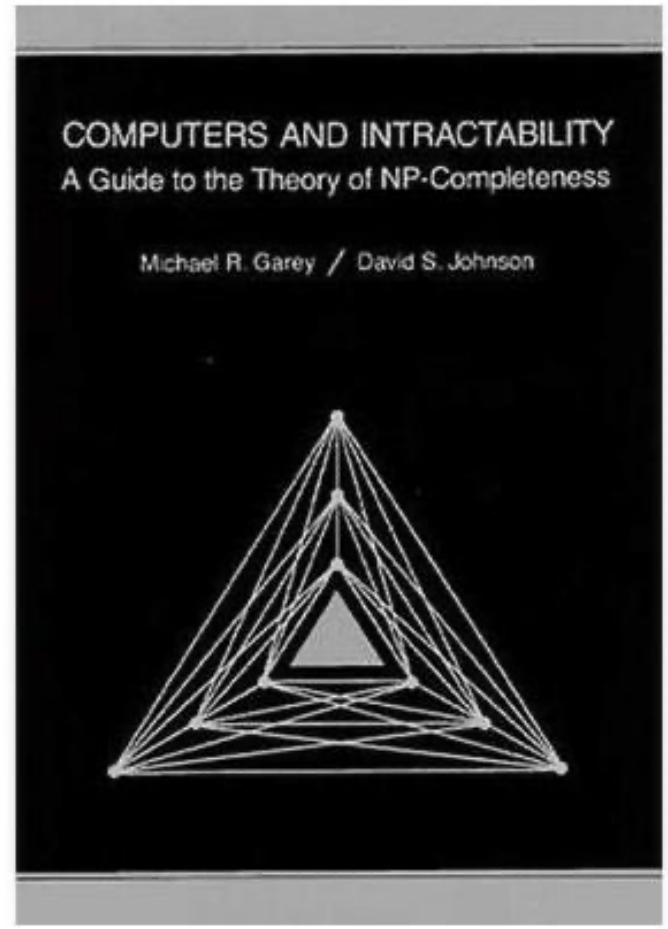
More hard computational problems

- 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 March 19, 2015. 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 | 2014 | 2015

1. M R Garey, D S Johnson
   Computers and Intractability: A Guide to the Theory of NPCompleteness" W.H. Feeman and 1979
   11468
2. J Sambrook, E F Fritsch, T Maniatis
   Molecular Cloning: A Laboratory Manual, Vol. 1, 2nd edn Nucleic Acids Research, 1989
   10362
3. V Vapnik
   Statistical Learning Theory. 1998
   9898
4. T M Cover, J A Thomas
   Elements of Information Theory Series in Telecommunications, 1991
   9198
5. U K Laemmli
   Cleavage of structural proteins during the assembly of the head of bacteriophage T4. Nature 227:680–685 1970
   9092
6. T H Cormen, C E Leiserson, R L Rivest, C Stein
   Introduction to Algorithms. 1990
   9039

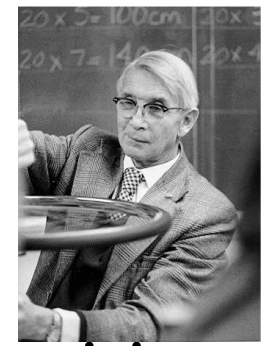


More hard computational problems

- 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, ..., *an*, 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

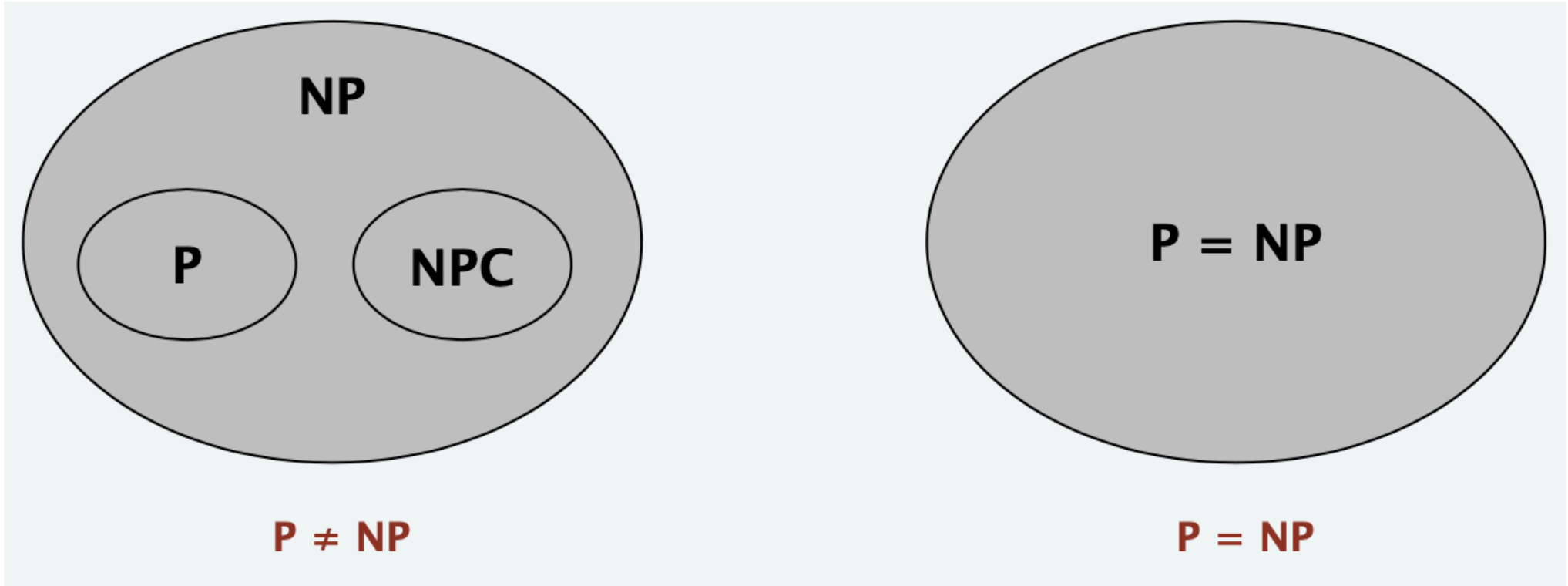
- Extent of NP-completeness. [Papadimitriou 1995]
  - Prime intellectual export of CS to other disciplines.
  - 6,000 citations per year (more than "compiler", "OS", "database").
  - Broad applicability and classification power.




- 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.
    - 3D-ISING: a holy grail of statistical mechanics
  - 2000: Istrail proves 3D-ISING $\in$ **NP**-complete.
    - search for closed formula appears doomed


P vs. NP revisited

- Overwhelming consensus (still). **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*

- *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?

Asymmetry of NP

- 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.
- **Definition.**
  - Given a decision problem X, its complement $\bar{X}$ is the same problem with the *yes* and *no* answers reverse.
- Ex.
  - X = { 0, 1, 4, 6, 8, 9, 10, 12, 14, 15, ... }
  - $\bar{X}$ = { 2, 3, 5, 7, 11, 13, 17, 23, 29, ... }
- **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 if and only if *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
- 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. P ⊆ NP ∩ co-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 **P** = **NP** ∩ **co-NP**?
  - Mixed opinions.
  - Many examples where problem found to have a nontrivial good characterization, but only years later discovered to be in **P**.

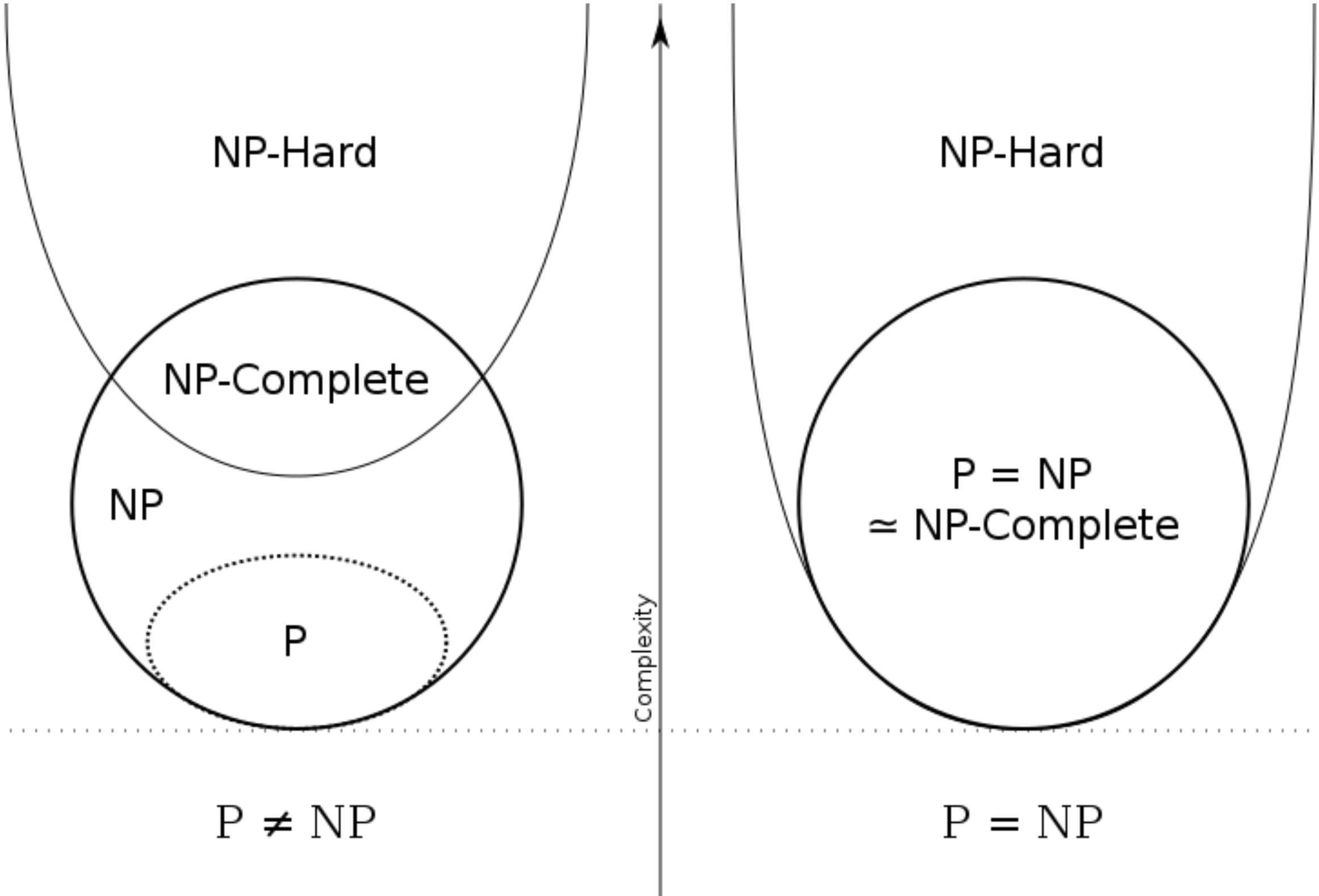
- *P vs. NP*
- *NP-complete*
- *co-NP*
- ***NP-hard***

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.
    - This problem can be **P**

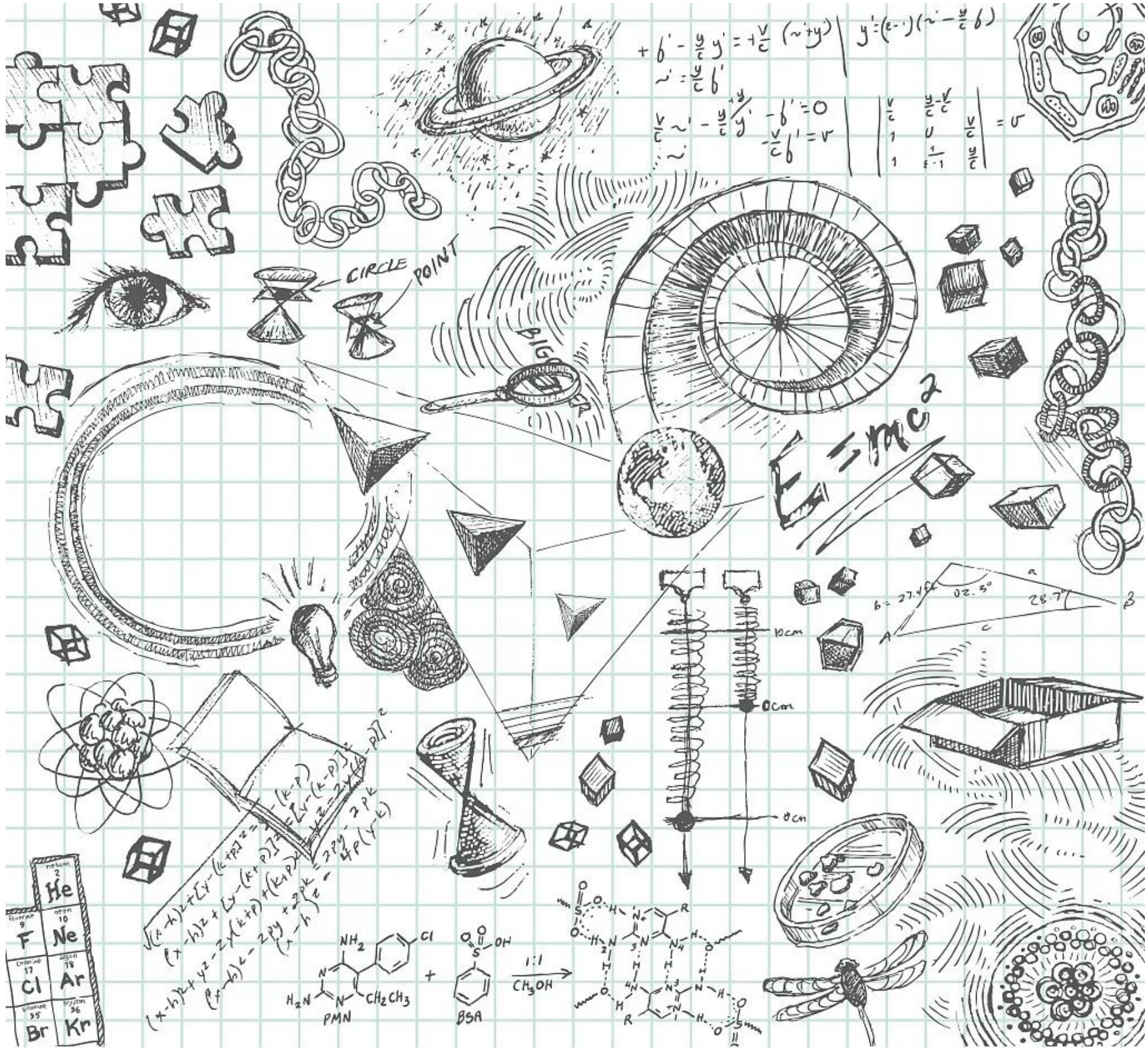
> One final criticism (which applies to all the terms suggested) was stated nicely by Vaughan Pratt: "If the Martians know that P = NP for Turing Machines and they kidnap me, I would lose face calling these problems 'formidable'." Yes; if P = NP , there's no need for any term at all. But I'm willing 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 .

P, NP, NP-complete, NP-hard



P ≠ NP

P = NP


CIRCLE
POINT
BIG
E=mc2
10cm
0cm
8cm
PMN
BSA
1:1
CH3OH