https://algs4.cs.princeton.edu

INTRACTABILITY

- *introduction*
- *P vs. NP*
- *poly-time reductions*
- *NP-completeness*
- *dealing with intractability*

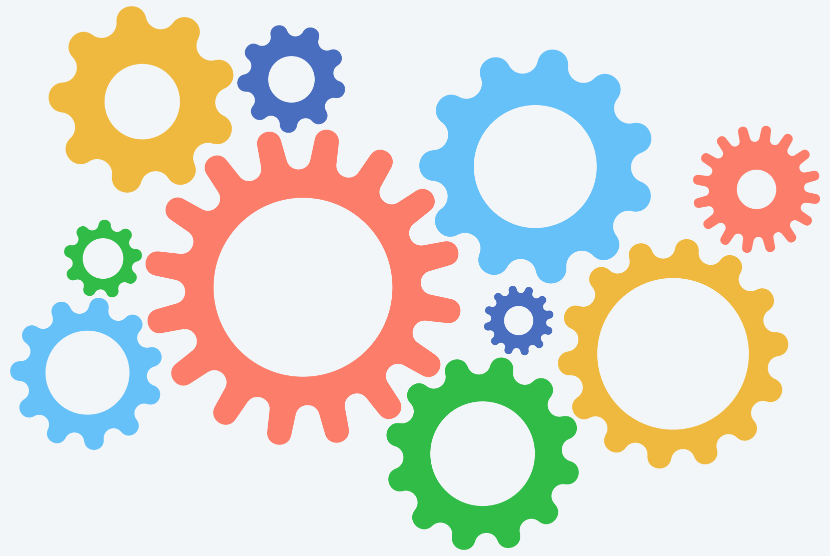
Last updated on 12/5/23 10:04AM

Overview: introduction to advanced topics

Main topics. [final two lectures]

- **Intractability:** barriers to designing efficient algorithms.
- **Algorithm design:** paradigms for solving problems.

Shifting gears.



- From individual problems to problem-solving models.
- From linear/quadratic to poly-time/exponential scale.
- From implementation details to conceptual frameworks.

Goals.

- Introduce you to essential ideas.
- Place algorithms and techniques we've studied in a larger context.


Algorithms

Robert Sedgewick | Kevin Wayne

https://algs4.cs.princeton.edu

INTRACTABILITY

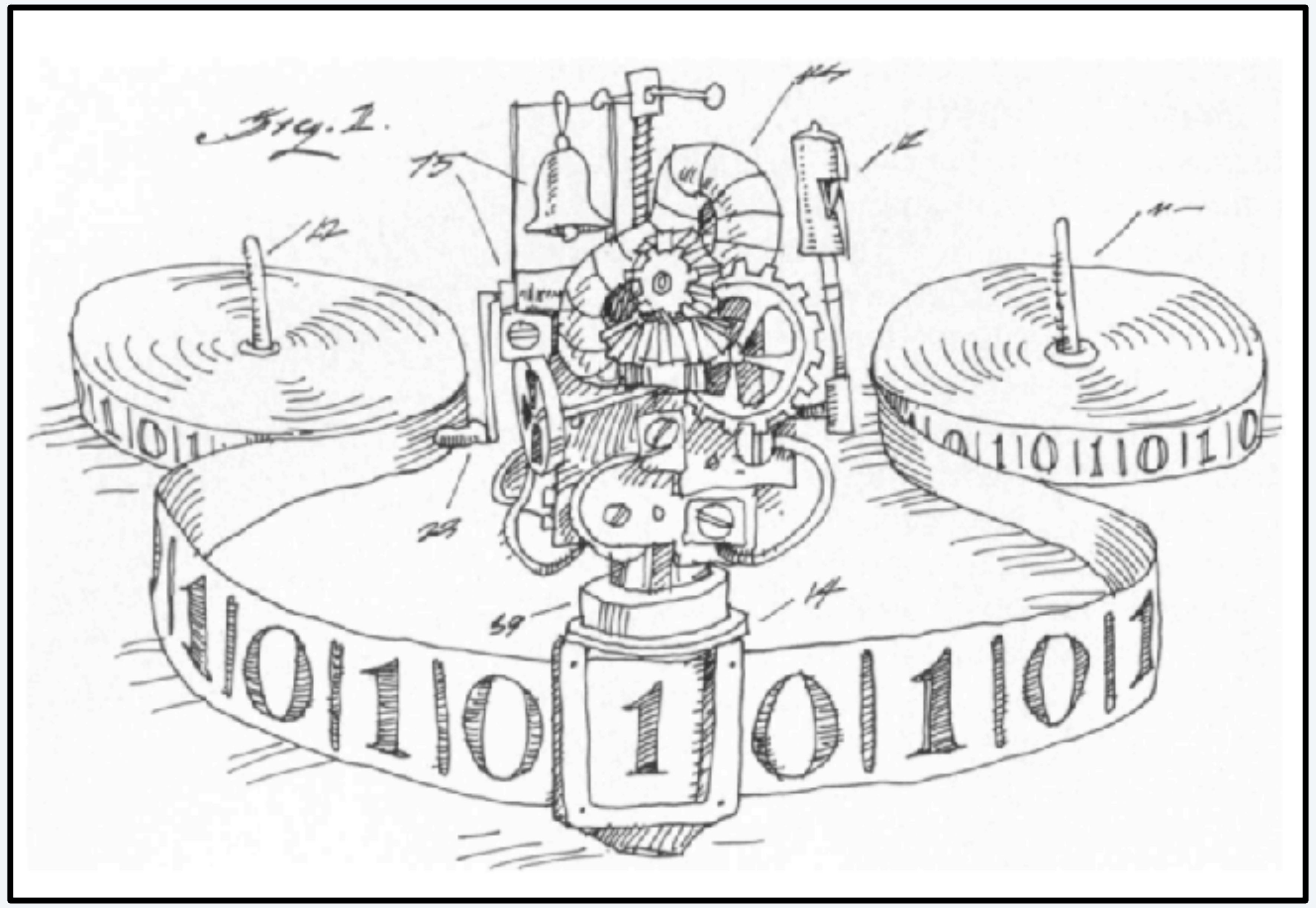
- ▸ *introduction*
- ▸ *P vs. NP*
- ▸ *poly-time reductions*
- ▸ *NP-completeness*
- ▸ *dealing with intractability*

Fundamental questions

Q1. What is an algorithm?

Q2. What is an efficient algorithm?

Q3. Which problems can be solved efficiently?



A Turing machine

A difficult problem: integer factorization

Integer factorization. Given an integer x, find a nontrivial factor. ← *or report that no such factor exists*

not 1 nor x

Ex.

147573952589676412927

a FACTOR instance

193707721

a factor

135066410865995223349603216278805969938881475605667027524485143851526510604859533833940287150571909441798207282164471551373680419703964191743046496589274256239341020864383202110372958725762358509643110564073501508187510676594629205563685529475213500852879416377328533906109750544334999811150056977236890927563

a very challenging FACTOR instance
(factor to earn an A+ in COS 226)

Core application area. Cryptography.

Brute-force search. Try all possible divisors between 2 and $\sqrt{x}$.

if there's a nontrivial factor larger than $\sqrt{x}$, there is one smaller than $\sqrt{x}$

Another difficult problem: boolean satisfiability

Boolean satisfiability. Given a system of boolean equations, find a satisfying truth assignment.

or report that no such assignment is possible

Ex.

$$\begin{array}{ccccccccc}
\neg x_1 & \textit{or} & x_2 & \textit{or} & x_3 & & & = & \textit{true} \\
x_1 & \textit{or} & \neg x_2 & \textit{or} & x_3 & & & = & \textit{true} \\
\neg x_1 & \textit{or} & \neg x_2 & \textit{or} & \neg x_3 & & & = & \textit{true} \\
\neg x_1 & \textit{or} & \neg x_2 & \textit{or} & & \textit{or} & x_4 & = & \textit{true} \\
 & & \neg x_2 & \textit{or} & x_3 & \textit{or} & x_4 & = & \textit{true}
\end{array}$$

a SAT instance

$$\begin{array}{ccc}
x_1 & = & \textit{false} \\
x_2 & = & \textit{false} \\
x_3 & = & \textit{true} \\
x_4 & = & \textit{true}
\end{array}$$

a satisfying truth assignment

Applications.

- Automatic verification systems for software.
- Mean field diluted spin glass model in physics.
- Electronic design automation (EDA) for hardware.
- …

Another difficult problem: boolean satisfiability

Boolean satisfiability. Given a system of boolean equations, find a satisfying truth assignment.

Ex.

$$\begin{array}{cccccccccc}
\neg x_1 & or & x_2 & or & x_3 & & & = & true \\
x_1 & or & \neg x_2 & or & x_3 & & & = & true \\
\neg x_1 & or & \neg x_2 & or & \neg x_3 & & & = & true \\
\neg x_1 & or & \neg x_2 & or & & or & x_4 & = & true \\
 & & \neg x_2 & or & x_3 & or & x_4 & = & true
\end{array}$$

a SAT instance

Brute-force search. Try all 2^n possible truth assignments, where n = # variables.

Q. Can we do anything substantially more clever?

A. Probably no. [stay tuned]

How difficult can it be?

Imagine a galactic computer…

- With as many processors as electrons in the universe.
- Each processor having the power of today's supercomputers.
- Each processor working for the lifetime of the universe.

quantity	**estimate**
electrons in universe	10^{79}
instructions per second	10^{13}
age of universe in seconds	10^{17}

Q. Could galactic computer solve satisfiability instance with $1{,}000$ variables using brute-force search?

A. Not even close: $2^{1000} > 10^{300} >> 10^{79} \cdot 10^{13} \cdot 10^{17} = 10^{109}$.

Lesson. Exponential growth dwarfs technological change.

Polynomial time

Q2. What is an efficient algorithm?

A2. Algorithm whose running time in polynomial in input size n. ← *n = # of bits in input*

with respect to some reasonable model of computation (Turing machine, λ-calculus, word RAM, …)

Polynomial time. Number of elementary operations is at most n^k for some constant k.

must hold for all inputs of size n (worst-case running time)

We use to a poly-time algorithm as a surrogate for useful in practice. ← *more on practicality shortly!*

order of growth	emoji	name	today
$\Theta(1)$	😍	**constant**	🙂
$\Theta(\log n)$	😎	**logarithmic**	🙂
$\Theta(n)$	😁	**linear**	🙂
$\Theta(n \log n)$	😀	**linearithmic**	🙂
$\Theta(n^2)$	😕	**quadratic**	🙂
$\Theta(n^3)$	🙁	**cubic**	🙂
$\Theta(n^{\log n})$	😨	**quasipolynomial**	👿
$\Theta(1.1^n)$	😭	**exponential**	👿
$\Theta(2^n)$	😈	**exponential**	👿
$\Theta(n!)$	👿	**factorial**	👿

Intractability: quiz 1

Which of the following are poly-time algorithms?

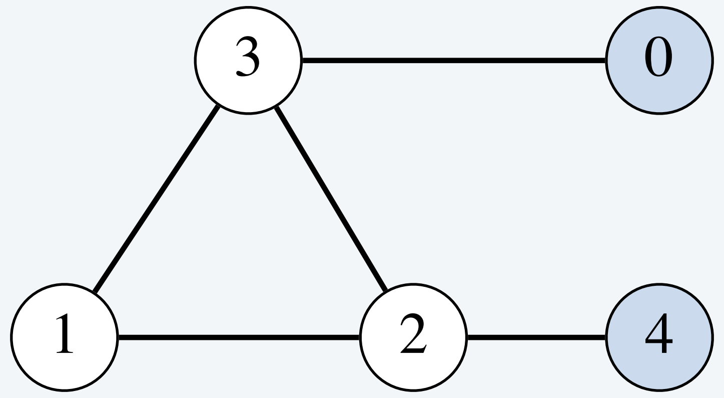
A. Brute-force search for satisfiability.

B. Brute-force search for factoring.

C. Both A and B.

D. Neither A nor B.

Some computational problems

problem	**description**	**example instance**	**a solution**	**poly-time algorithm**
FACTOR *(integer factorization)*	given an integer, find a nontrivial factor	147573952589676412927	193707721	?
SAT *(boolean satisfiability)*	given a system of boolean equations, find a satisfying assignment	$\neg x_2 \;\textit{or}\; x_3 = \textit{true}$ $\neg x_1 \;\textit{or}\; \neg x_2 \;\textit{or}\; \neg x_3 = \textit{true}$ $x_2 \;\textit{or}\; \neg x_3 = \textit{true}$	$x_1 = \textit{false}$ $x_2 = \textit{true}$ $x_3 = \textit{true}$	?
SORT *(sorting)*	given an array of integers, find a permutation that puts the elements in ascending order	[45, 32, 21, 67, 226]	[2, 1, 0, 3, 4]	*insertion sort*
ST-CONN *(graph connectivity)*	given a graph and two vertices, find a path that connects them		0–3–2–4	*depth-first search*
⋮	⋮	⋮	⋮	⋮

Intractable problems

Q3. Which problems can be solved efficiently?

A3. Those for which poly-time algorithms exist.

Def. A problem is intractable if no poly-time algorithm exists to solve it.

Q4. How to prove that a problem is intractable?

A4. Generally no easy way. Focus of today's lecture.

tractable	**intractable?**
primality	*integer factorization*
shortest path	*longest path*
min cut	*max cut*
2-SAT	3-SAT ← *3 boolean variables per equation*
⋮	⋮

Intractable problems

https://algs4.cs.princeton.edu

INTRACTABILITY

- ▸ *introduction*
- ▸ ***P vs. NP***
- ▸ *poly-time reductions*
- ▸ *NP-completeness*
- ▸ *dealing with intractability*

Search problems

Search problem. Computational problem for which you can check a solution in poly-time.

Ex 1. [integer factorization] Given an n-bit integer x, find a nontrivial factor.

147573952589676412927

instance I

193707721

solution S

Poly-time checking algorithm. Check whether solution is a divisor of x. ← *$O(n^2)$ time via long division*

Remark. Suffices to verify a purported solution.

- Doesn't need to find the solution from scratch.
- Doesn't need to address case when no solution exists (e.g., if x is prime).

Search problems

Search problem. Computational problem for which you can check a solution in poly-time.

Ex 2. [boolean satisfiability] Given a system of m boolean equations in n variables, find a satisfying assignment.

$$\begin{array}{ccccccccc}
\neg x_1 & or & x_2 & or & x_3 & & & = & true \\
x_1 & or & \neg x_2 & or & x_3 & & & = & true \\
\neg x_1 & or & \neg x_2 & or & \neg x_3 & & & = & true \\
\neg x_1 & or & \neg x_2 & or & & or & x_4 & = & true \\
 & & \neg x_2 & or & x_3 & or & x_4 & = & true
\end{array}$$

instance I

$x_1 = false$

$x_2 = false$

$x_3 = true$

$x_4 = true$

solution S

Poly-time checking algorithm. Plug values of solution into system of equations and check.

$O(mn)$ time

NP

Definition. **NP** is the class of all search problems. ←

More accurately, **FNP**.
NP = *nondeterministic poly-time*

problem	description	poly-time checking algorithm
FACTOR *(integer factorization)*	given an integer, find a nontrivial factor	*long division*
SAT *(boolean satisfiability)*	given a boolean formula, find a satisfying assignment	*plug in boolean values and evaluate boolean equations*
SORT *(sorting)*	given an array of integers, find a permutation that puts the elements in ascending order	*compare all adjacent integers in permutation*
ST-CONN *(graph connectivity)*	given a digraph and two vertices, find a path that connects them	*check for existence of edges between consecutive vertices in path*
BLOCK-CHAIN (hash verification)	given strings `s` and `h`, find a string `t` whose concatenation with `s` hashes to `h`	*compute the hash of the concatenation and compare with* h
⋮	⋮	⋮

Significance. Problems that scientists, engineers, and programmers aspire to solve in practice.

Intractability: quiz 2

Which of these problems are in NP?

A. Given a graph G, find a simple path with the most edges.

B. Given a graph G and an integer k, find a simple path with $\geq k$ edges.

C. Both A and B.

D. Neither A nor B.

P

Definition. **P** is the class of all search problems that can be solved in poly-time. ← *more accurately,* **FP**

problem	description	poly-time algorithm
SORT *(sorting)*	given an array of integers, find a permutation that puts the elements in ascending order	*insertion sort*
ST-CONN *(graph connectivity)*	given a digraph and two vertices, find a path that connects them	*depth-first search*
JAVA *(legal Java program)*	given a string, is it a legal Java program?	*javac*
L-SOLVE *(system of linear equations)*	given a system of linear equations, find a solution	*Gaussian elimination*†
⋮	⋮	⋮

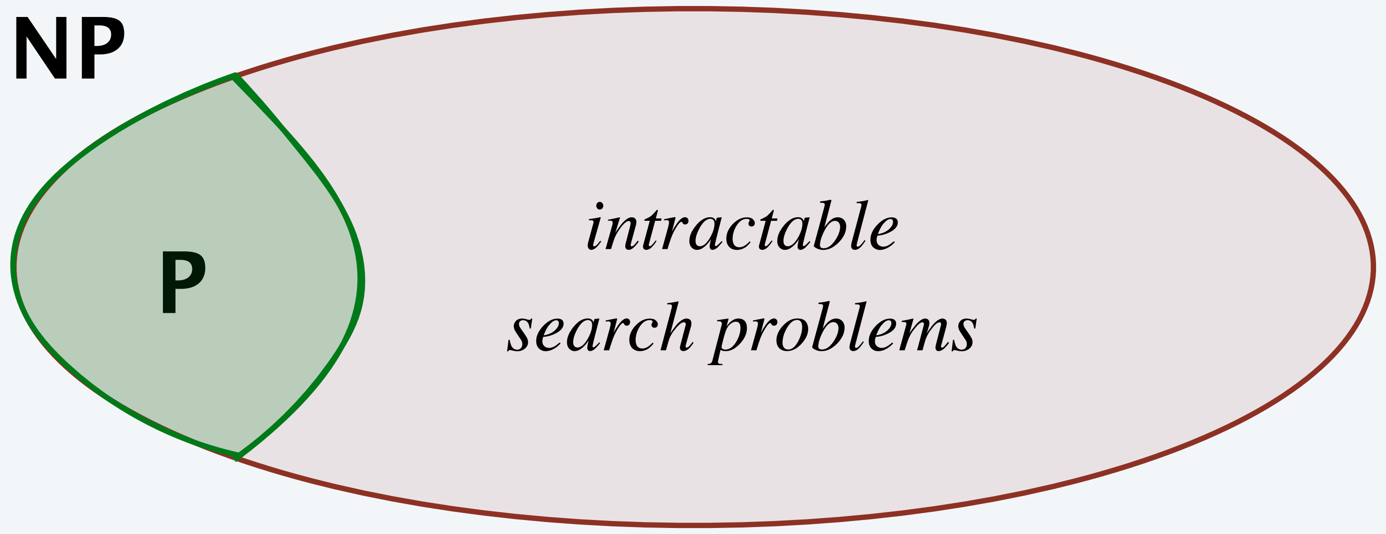
Significance. Problems that scientists, engineers, and programmers do solve in practice.

Note. All problems in **P** are also in **NP**. ← *any string serves as certificate*

P vs. NP

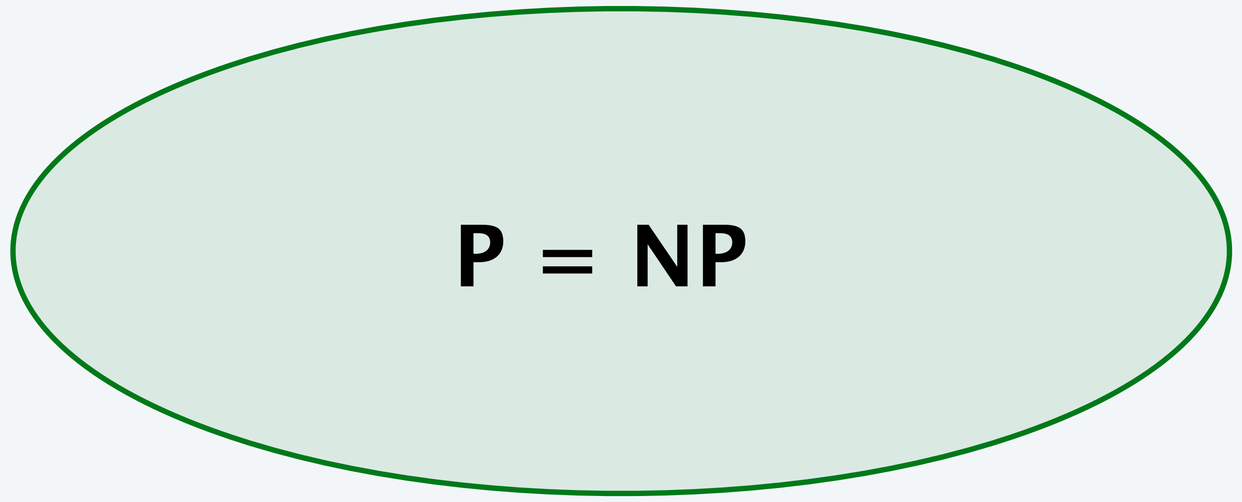
The central question. Does **P** = **NP** ?

- **P** = set of search problems that are solvable in poly-time.
- **NP** = set of search problems (checkable in poly-time).



P ≠ NP

brute-force search may be
the best we can do



P = NP

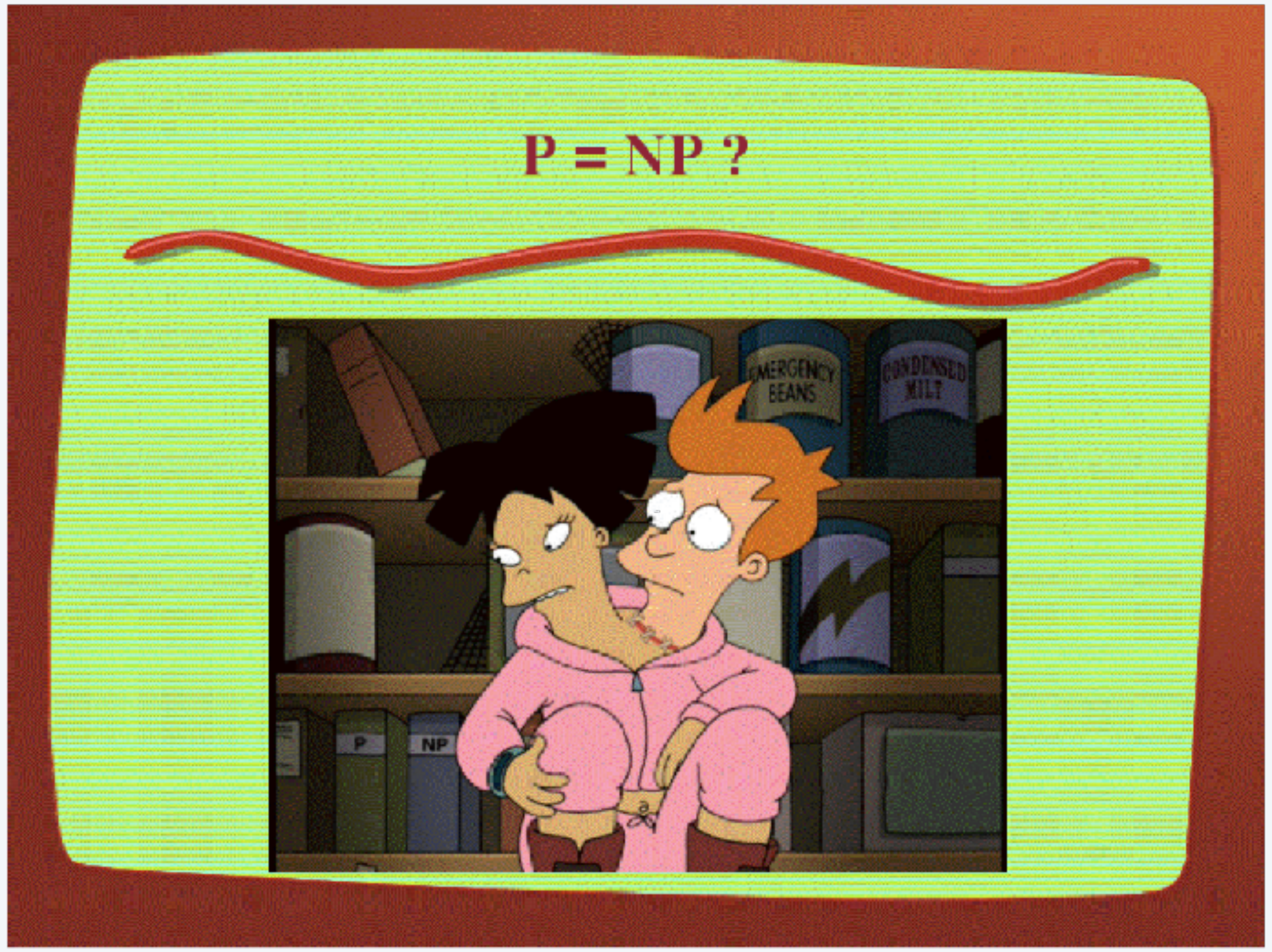
poly-time algorithms for
Factor, Sat, Longest-Path, …

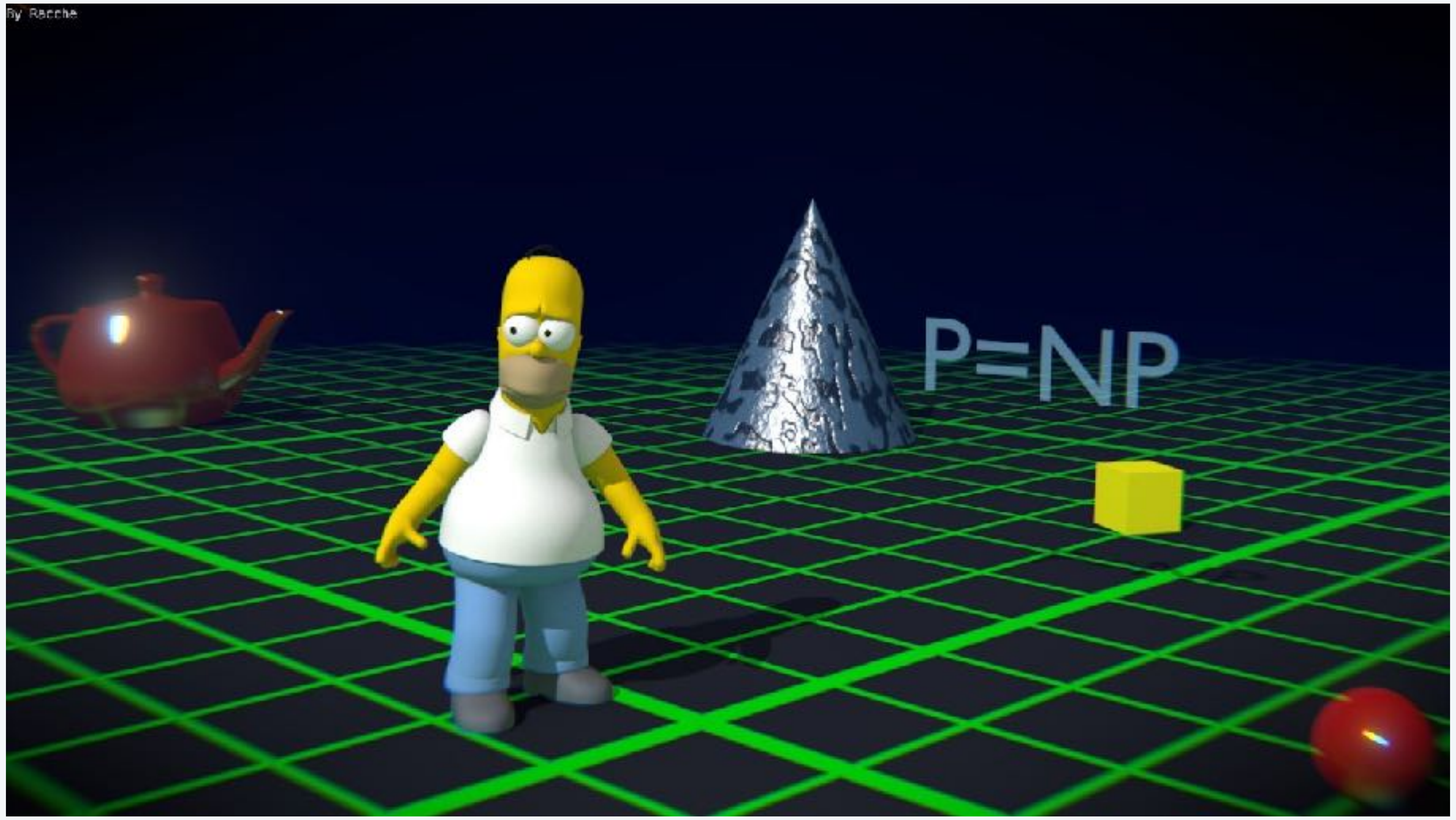
Consensus opinion. **P ≠ NP.** ← *but nobody has been able to prove or disprove* (!!!)

P vs. NP

The central question. Does **P** = **NP** ?

- **P** = set of search problems that are solvable in poly-time.
- **NP** = set of search problems (checkable in poly-time).





Consensus opinion. **P ≠ NP.** ← *but nobody has been able to prove or disprove* (!!!)

Creativity: another way to view the situation

Analogy. Creative genius vs. ordinary appreciation of creativity.

domain	creative genius	ordinary appreciation
music	Taylor Swift writes a song	a Swiftie appreciates it
mathematics	Wiles proves a deep theorem	a colleague checks it
engineering	Boeing designs an efficient airfoil	a simulator verifies it
science	Einstein proposes a theory	an experimentalist validates it
programming	GitHub Copilot generates a program	a programmer verifies it

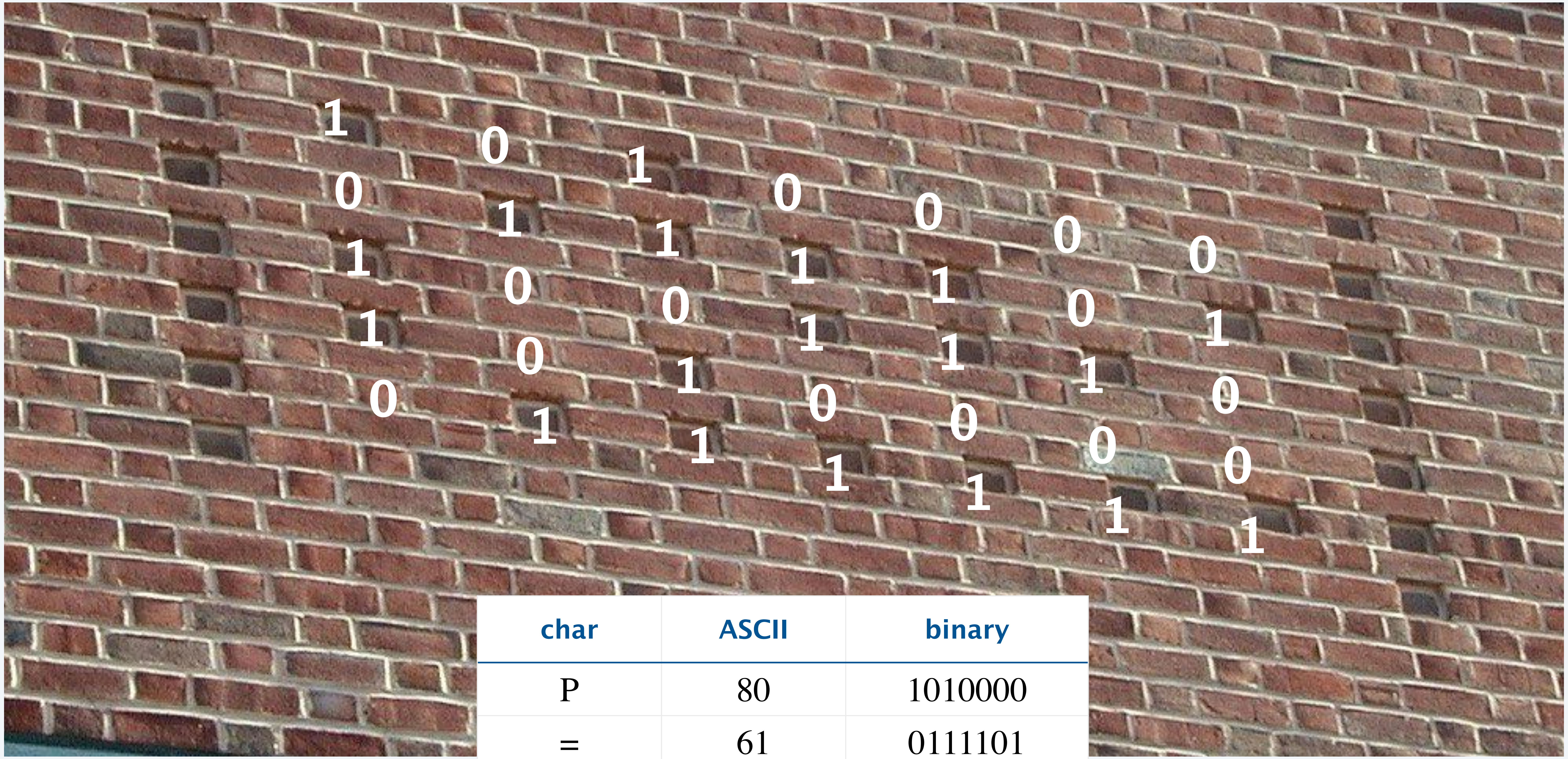
creative genius

ordinary appreciation

Intuition. Checking a solution seems like it should be way easier than finding it.

Princeton computer science building

Princeton computer science building (closeup)



char	ASCII	binary
P	80	1010000
=	61	0111101
N	78	1001110
P	80	1010000
?	63	0111111

Algorithms

Robert Sedgewick | Kevin Wayne

https://algs4.cs.princeton.edu

INTRACTABILITY

- *introduction*
- *P vs. NP*
- *poly-time reductions*
- *NP-completeness*
- *dealing with intractability*

Bird's-eye view

Desiderata. Classify problems according to computational requirements.

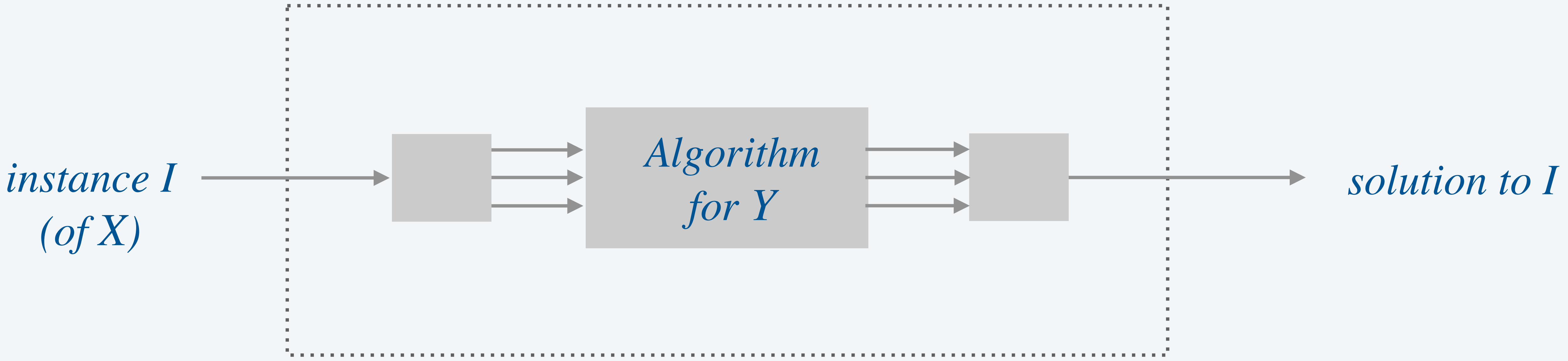
Desiderata'. Suppose we could (not) solve problem X efficiently.
What else could we (not) solve efficiently?

> " *Give me a lever long enough and a fulcrum on which to place it, and I shall move the world.* " — *Archimedes*

Poly-time reduction

Def. Problem X poly-time reduces to problem Y if X can be solved with:

- Polynomial number of elementary operations.
- Polynomial number of calls to Y. ← *Cook reduction*



Algorithm for X

Design algorithms. If X poly-time reduces to Y, and can solve Y efficiently, then can also solve X.

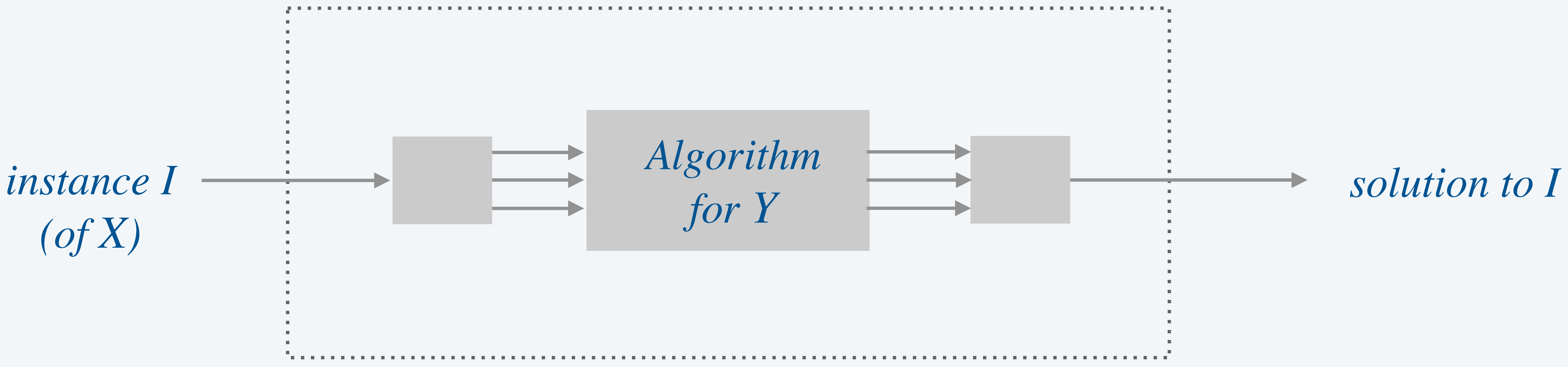
Ex 1. MEDIAN reduces to SORT.

Ex 2. BIPARTITE-MATCHING reduces to MAX-FLOW.

Poly-time reduction

Def. Problem X poly-time reduces to problem Y if X can be solved with:

- Polynomial number of elementary operations.
- Polynomial number of calls to Y.



Algorithm for X

Establish intractability. If SAT poly-time reduces to Y, then Y is intractable. ← *assuming* SAT *is intractable*

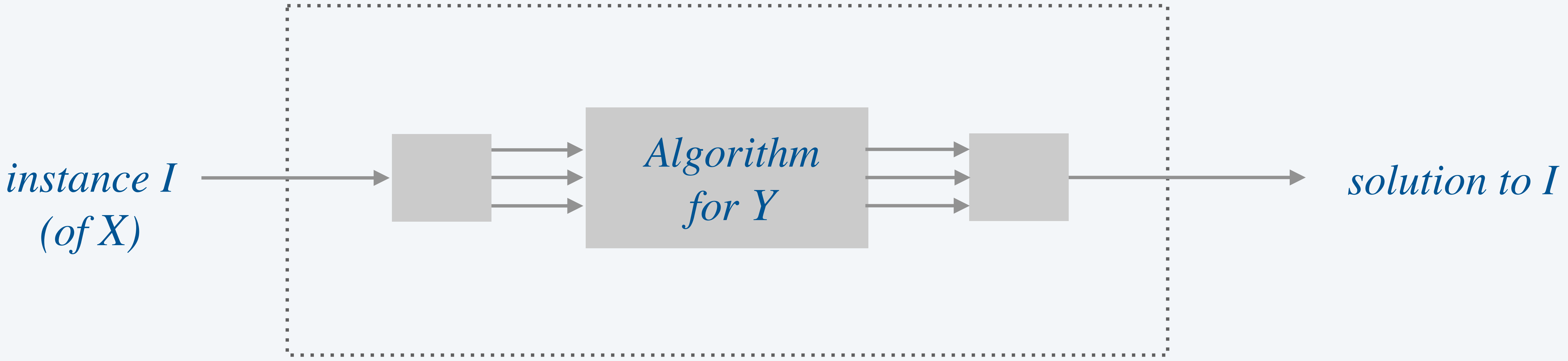
Mentality (to establish intractability).

- If I could solve Y in poly-time, then I could also solve SAT in poly-time.
- SAT is believed to be intractable.
- Therefore, so is Y.

Poly-time reduction

Def. Problem X poly-time reduces to problem Y if X can be solved with:

- Polynomial number of elementary operations.
- Polynomial number of calls to Y.



Algorithm for X

Common mistake. Confusing *X poly-time reduces to Y* with *Y poly-time reduces to X*.

X reduces to SAT: X is no harder than SAT. (If I can solve SAT, then I can solve X.)

SAT reduces to X: X is no easier than SAT. (If I can solve X, then I can solve SAT.)

Integer linear programming

ILP. Given a system of linear inequalities, find an integer-valued solution.

$$
\begin{array}{ccccccccccc}
3x_1 & + & 5x_2 & + & 2x_3 & + & x_4 & + & 4x_5 & \geq & 10 \\
5x_1 & + & 2x_2 & & & + & 4x_4 & + & x_5 & \leq & 7 \\
x_1 & & & + & x_3 & + & 2x_4 & & & \leq & 2 \\
3x_1 & & & + & 4x_3 & + & 7x_4 & & & \leq & 7 \\
x_1 & & & & & + & x_4 & & & \leq & 1 \\
x_1 & & & + & x_3 & & & + & x_5 & \leq & 1 \\
x_1 & , & x_2 & , & x_3 & , & x_4 & , & x_5 & \in & \mathbb{Z}
\end{array}
$$

linear inequalities

integer variables

instance I

$$
\begin{array}{ccc}
x_1 & = & 0 \\
x_2 & = & 1 \\
x_3 & = & 0 \\
x_4 & = & 1 \\
x_5 & = & 1
\end{array}
$$

solution S

Context. Cornerstone problem in operations research.

Remark. Finding a real-valued solution can be solved in poly-time (linear programming).

SAT poly-time reduces to ILP

SAT. Given a system of boolean equations in CNF, find a solution.

conjunctive normal form (AND of ORs)

$$\begin{array}{cccccccccc}
\neg x_1 & or & x_2 & or & x_3 & & & = & true \\
x_1 & or & \neg x_2 & or & x_3 & & & = & true \\
\neg x_1 & or & \neg x_2 & or & \neg x_3 & & & = & true \\
\neg x_1 & or & \neg x_2 & or & & or & x_4 & = & true \\
 & & \neg x_2 & or & x_3 & or & x_4 & = & true
\end{array}$$

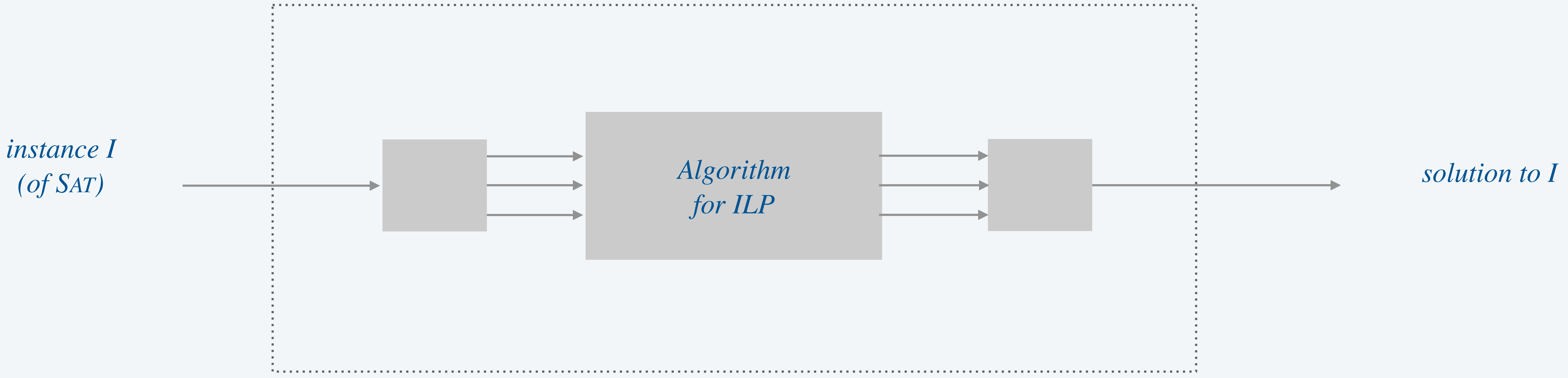
ILP. Given a system of linear inequalities, find an integer-valued solution.

$$\begin{array}{l}
y_i = 0 \;\Rightarrow\; x_i = false \\
y_i = 1 \;\Rightarrow\; x_i = true
\end{array}
\longrightarrow
\begin{array}{ccccc}
0 & \le & y_1 & \le & 1 \\
0 & \le & y_2 & \le & 1 \\
0 & \le & y_3 & \le & 1 \\
0 & \le & y_4 & \le & 1
\end{array}$$

$$\begin{array}{cccccccccc}
(1-y_1) & + & y_2 & + & y_3 & & & \ge & 1 \\
y_1 & + & (1-y_2) & + & y_3 & & & \ge & 1 \\
(1-y_1) & + & (1-y_2) & + & (1-y_3) & & y_4 & \ge & 1 \\
(1-y_1) & + & (1-y_2) & + & & + & y_4 & \ge & 1 \\
 & & (1-y_2) & + & y_3 & + & y_4 & \ge & 1
\end{array}$$

Solution to ILP instance provides solution to SAT instance.

SAT poly-time reduces to ILP



Algorithm for SAT

Preprocessing: boolean equations to linear inequalities

Post-processing: $y_i = 0 \implies x_i =$ false
$y_i = 1 \implies x_i =$ true

Intractability: quiz 3

Suppose that Problem X poly-time reduces to Problem Y.

Which of the following can we infer?

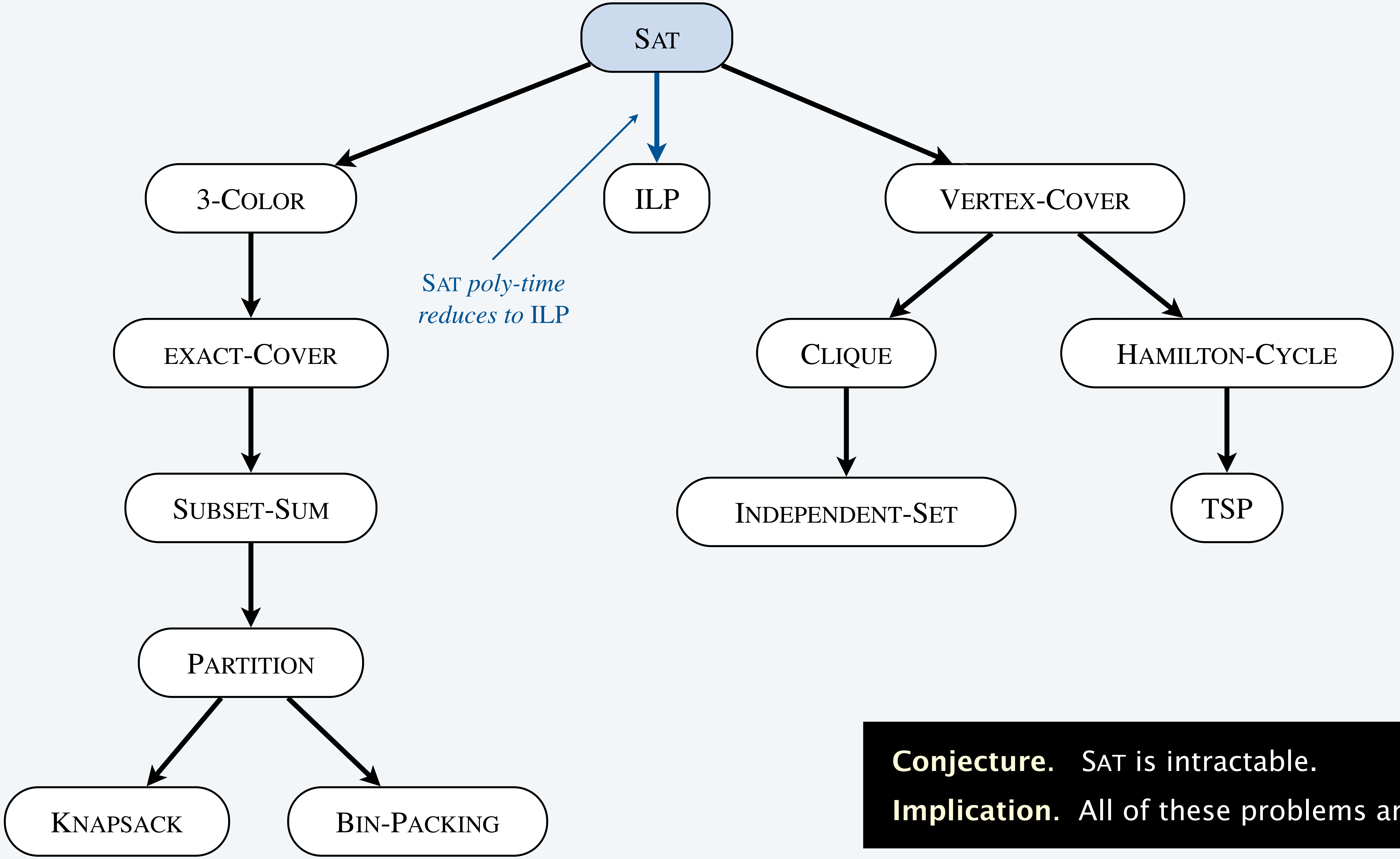
A. If X can be solved in poly-time, then so can Y.

B. If X cannot be solved in $\Theta(n^3)$ time, Y cannot be solved in poly-time.

C. If Y can be solved in $\Theta(n^3)$ time, then X can be solved in poly-time.

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

More poly-time reductions from SAT



SAT

3-COLOR

ILP

VERTEX-COVER

EXACT-COVER

CLIQUE

HAMILTON-CYCLE

SUBSET-SUM

INDEPENDENT-SET

TSP

PARTITION

KNAPSACK

BIN-PACKING

SAT poly-time reduces to ILP

Richard Karp
(1972)

Conjecture. SAT is intractable.

Implication. All of these problems are intractable.


Algorithms

Robert Sedgewick | Kevin Wayne

https://algs4.cs.princeton.edu

INTRACTABILITY

- ▸ *introduction*
- ▸ *P vs. NP*
- ▸ *poly-time reductions*
- ▸ *NP-completeness*
- ▸ *dealing with intractability*

NP-completeness

Def. A problem is **NP**-complete if

- It is in **NP**.
- All problems in **NP** poly-time to reduce to it. ← *intuitively, the "hardest problems" in* **NP**

Two worlds.

P ≠ NP

P = NP

Intractability: quiz 4

Suppose that X is NP-complete. What can you infer?

- **I.** X is in **NP**.
- **II.** If X can be solved in poly-time, then **P** = **NP**.
- **III.** If X cannot be solved in poly-time, then **P** ≠ **NP**.

- **A.** I only.
- **B.** II only.
- **C.** I and II only.
- **D.** I, II, and III.

Key property. An **NP**-complete problem can be solved in poly-time if and only if **P** = **NP**.

Cook–Levin theorem

Cook–Levin theorem. SAT is **NP**-complete.

Pf. Pioneering result in computer science.

Stephen Cook
(1971)

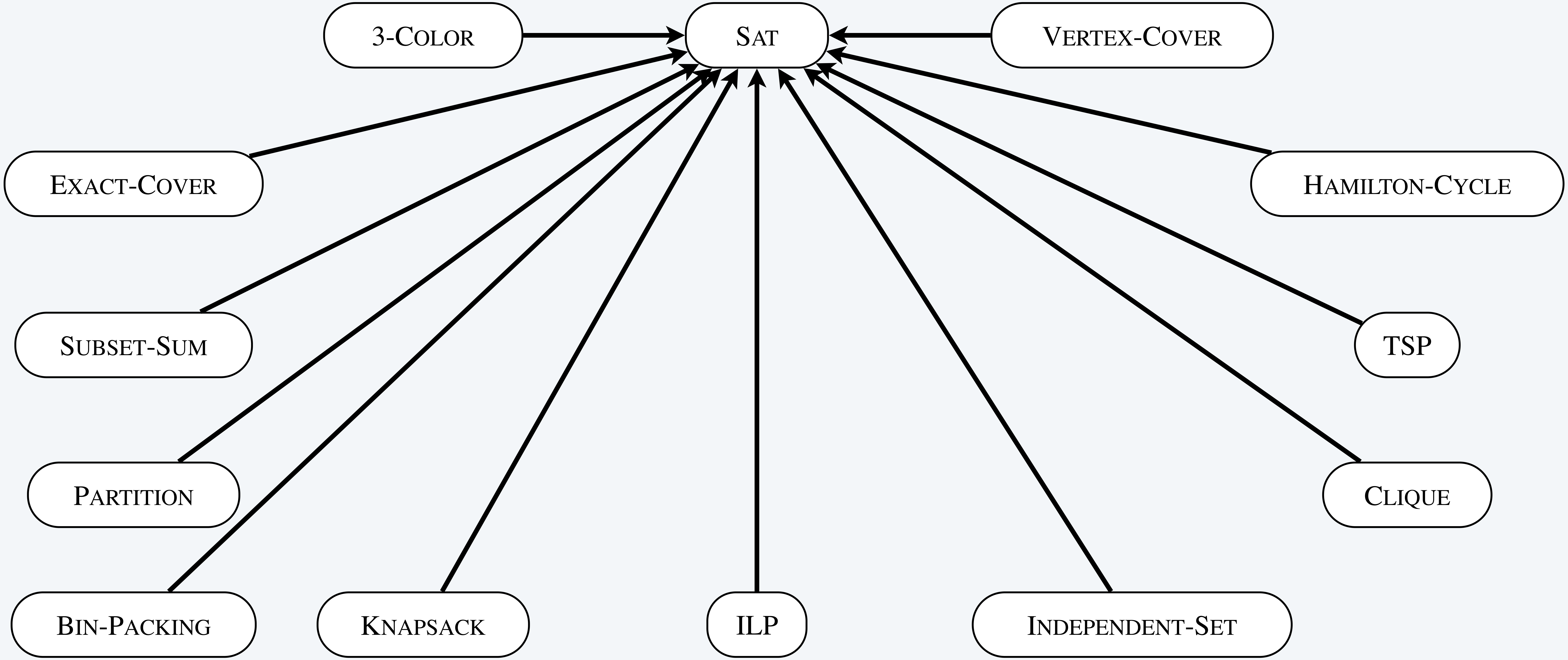
Leonid Levin
(1971)

Corollary. SAT can be solved in poly-time if and only if **P** = **NP**.

Impact. To provide that a new problem Y is **NP**-complete, suffices to show that:

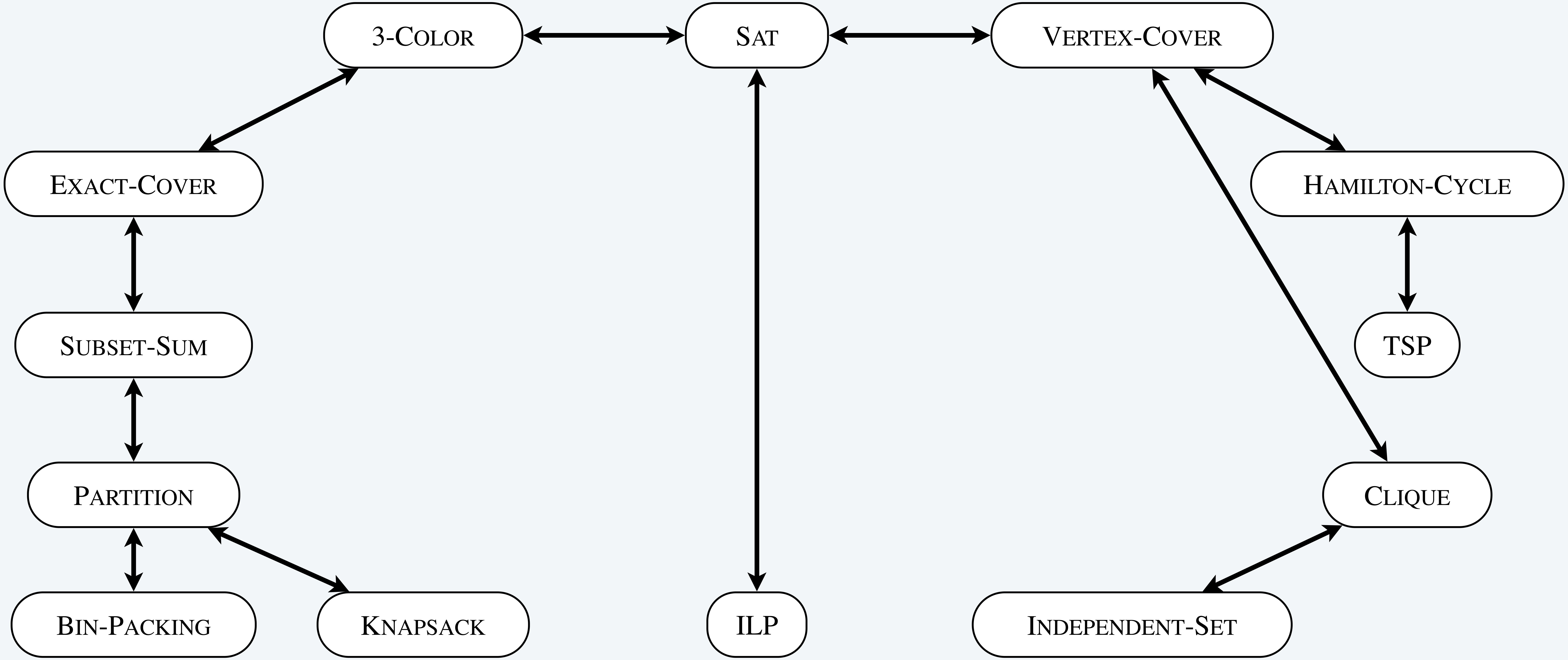
- Y is in **NP**.
- SAT poly-time reduces to Y.

Implications of Cook–Levin theorem



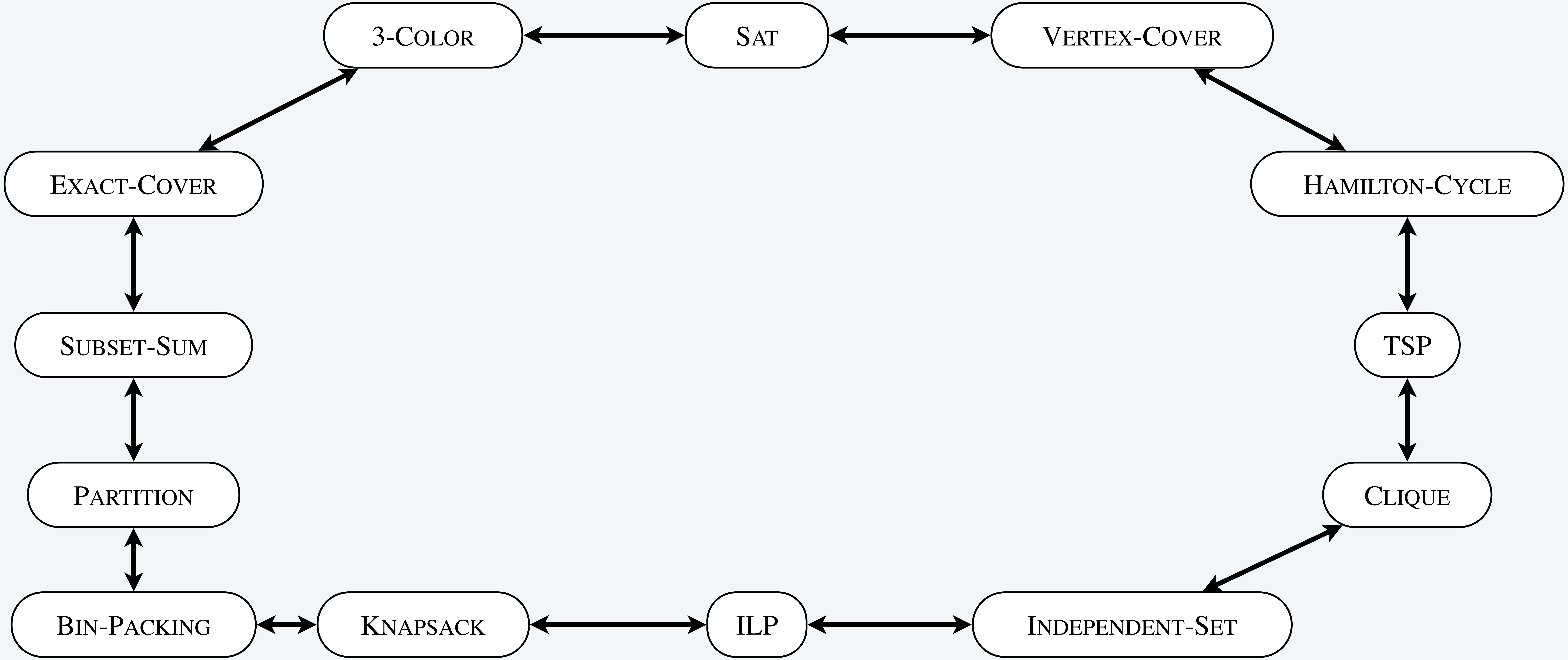
All of these problems (and many, many more) poly-time reduce to SAT.

Implications of Karp + Cook–Levin



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

Implications of Karp + Cook–Levin



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

More NP-complete problems

field of study	NP-complete problem
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*
Mechanical engineering	*structure of turbulence in sheared flows*
Medicine	*reconstructing 3d shape from biplane angiocardiogram*
Operations research	*traveling salesperson problem, integer programming*
Physics	*partition function of 3d Ising model*
Politics	*Shapley–Shubik voting power*
Pop culture	*versions of Sudoku, Checkers, Minesweeper, Tetris*
Statistics	*optimal experimental design*

6,000+ *scientific papers per year.*

https://algs4.cs.princeton.edu

INTRACTABILITY

- ‣ introduction
- ‣ P vs. NP
- ‣ poly-time reductions
- ‣ NP-completeness
- ‣ *dealing with intractability*

Dealing with intractability

Intractability: quiz 5

A program with which of these running times is most likely to be useful in practice?

- **A.** $10^{226}\, n$
- **B.** n^{226}

← *poly-time (but probably not useful in practice)*

- **C.** 1.000000001^n ← *exponential time (but probably useful in practice)*
- **D.** $(n!)\,!$ ← *good luck if $n \geq 5$*

Key point. Poly-time is not always a surrogate for useful in practice, though it tends to be true for the algorithms we encounter in the wild.

← *some poly-time algorithms are slow; some exponential-time algorithms are fast!*

Identifying intractable problems

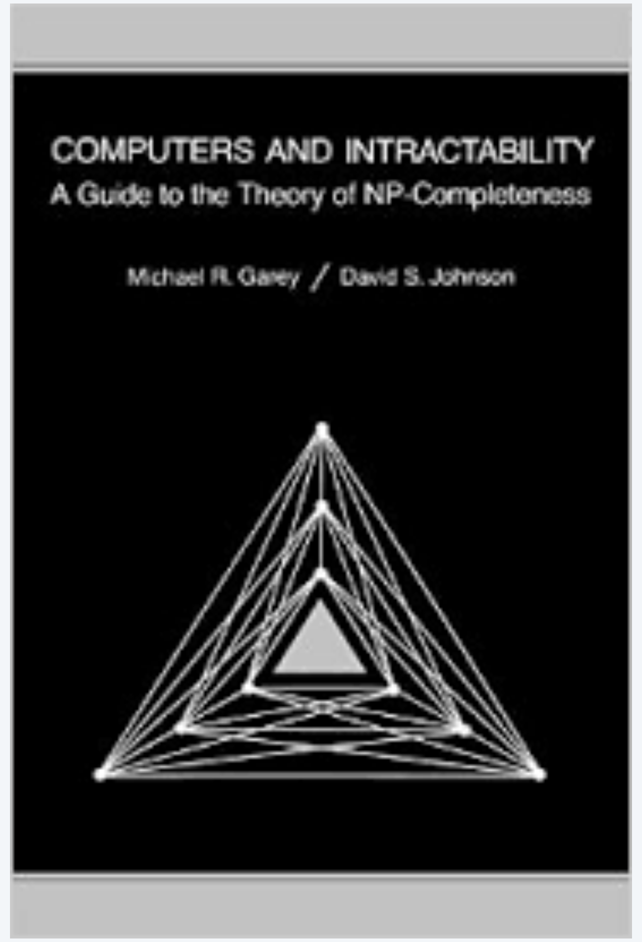
Establishing **NP**-completeness through poly-time reduction is an important tool in guiding algorithm design efforts.

Q4′. How to convince yourself that a problem is (probably) intractable?

A. [hard way] Long futile search for a poly-time algorithm (as for SAT).

A. [easy way] Poly-time reduction from SAT. ← *or from any other* **NP***-complete problem*

Caveat. Intricate reductions are common.

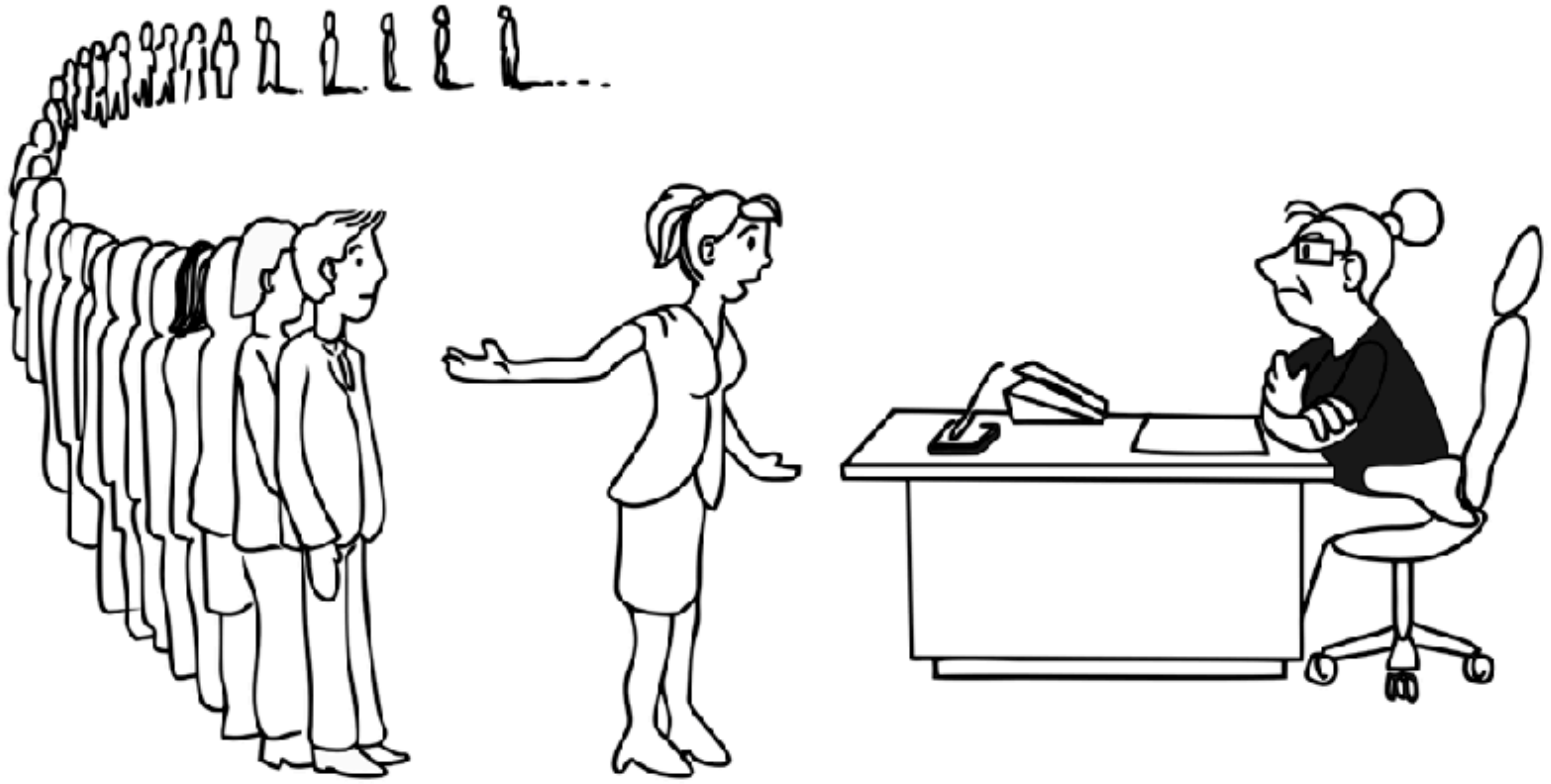

COMPUTERS AND INTRACTABILITY
A Guide to the Theory of NP-Completeness
Michael R. Garey / David S. Johnson

Identifying intractable problems

Step 1. Learn to identify **NP**-complete problems.

"I can't find a poly-time algorithm."

does not know about NP-completeness



"I can't find a poly-time algorithm, but neither can all these famous people."

knows about NP-completeness

Approaches to dealing with intractability

Q. What to do when you identify an **NP**-complete problem?

A. Safe to assume it is intractable: no worst-case poly-time algorithm for all problem instances.

Q1. Must your algorithm *always* run fast?

Solve real-world instances. Backtracking, TSP, SAT.

Q2. Do you need the *right* solution or a *good* solution?

Approximation algorithms. Look for suboptimal solutions.

Q3. Can you use the problem's hardness in your favor?

Leverage intractability. Cryptography.

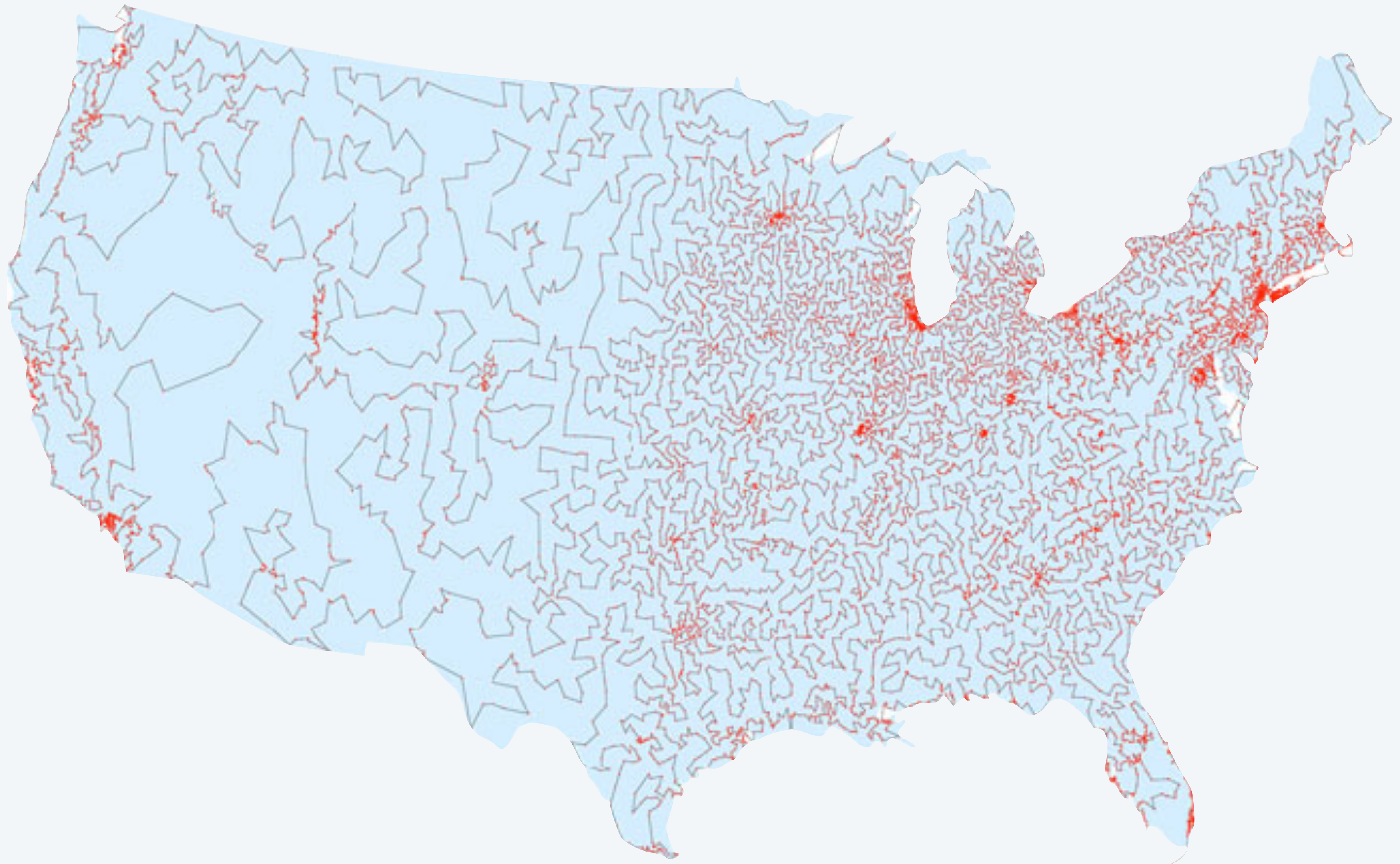
Dealing with intractability: find solutions to real-world problem instances

Observations.

- Worst-case inputs may not occur for practical problems.
- Instances that do occur in practice may be easier to solve.
- Reasonable approach: relax the condition of guaranteed poly-time.

Boolean satisfiability.

- Chaff solves real-world instances with 10,000+ variables.
- Princeton senior independent work (!) in 2000.



TSP solution for 13,509 US cities

Traveling salesperson problem.

- Concorde routinely solves large real-world instances.
- 85,900-city instance solved in 2006.

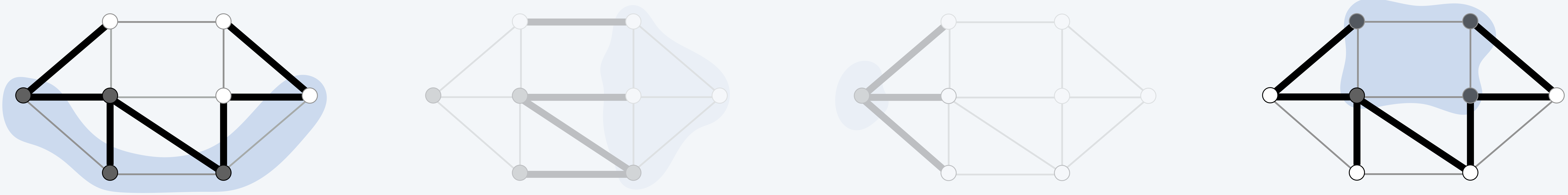
Integer linear programming.

- CPLEX routinely solves large real-world instances.
- Routinely used in scientific and commercial applications.

Dealing with intractability: approximation algorithms

MAX-CUT: given a graph G, find the cut with maximum number M of crossing edges.

Approximate version: find a large cut.



Algorithm: take a uniformly random cut.

Expected size is $E/2$; random assignment size is $\geq E/2 \geq M/2$ with at least 50% probability.

can improve to .878M

Dealing with intractability: approximation algorithms

3-SAT: given 3-variable equations on n boolean variables, find satisfying truth assignment.

Approximate version: find assignment that satisfies many equations.

Algorithm: take a uniformly random assignment.

Expected fraction of satisfied equations is $7/8$; random assignment does with at least 50% probability.

can't be improved (unless **P = NP***)*

Remark. Some problems have approximation algorithms with arbitrary precision.

For others, finding better approximations is also **NP**-complete!

Leveraging intractability: RSA cryptosystem

Modern cryptography applications.

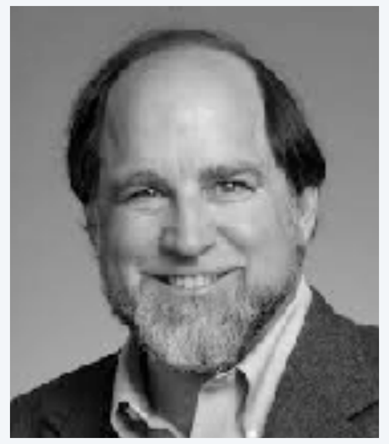
- Secure a secret communication.
- Append a digital signature.
- Credit card transactions.
- …

Verified by VISA

MasterCard® SecureCode™

RSA cryptosystem exploits intractability.

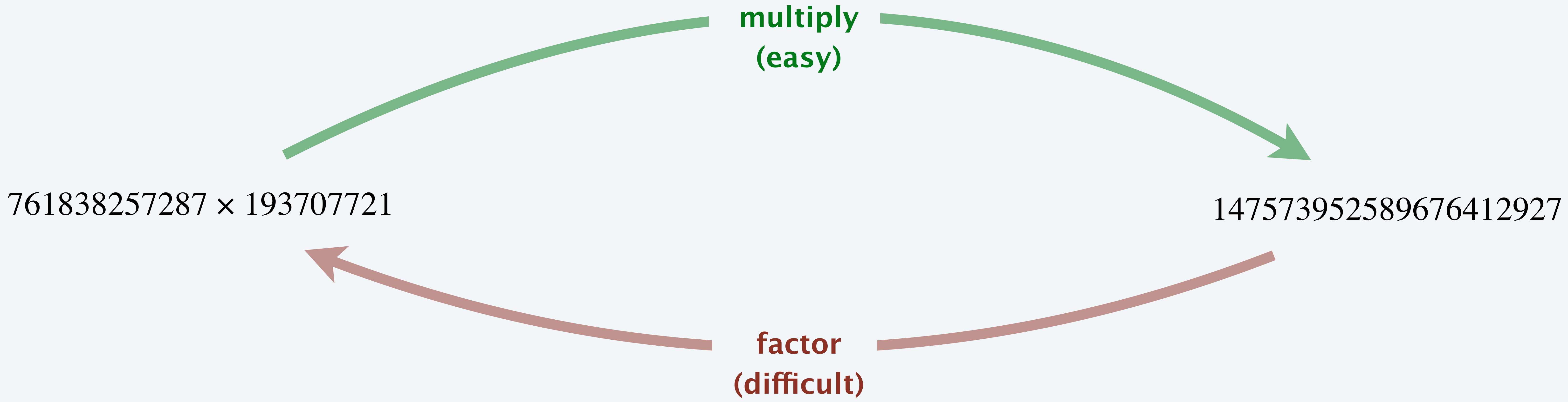
- To use: multiply/divide two n-digit integers (easy).
- To break: factor a $2n$-digit integer (intractable?).



Ron Rivest

Adi Shamir

Len Adelman



Leveraging intractability: guiding scientific inquiry

1926. Ising introduces a mathematical model for ferromagnetism.

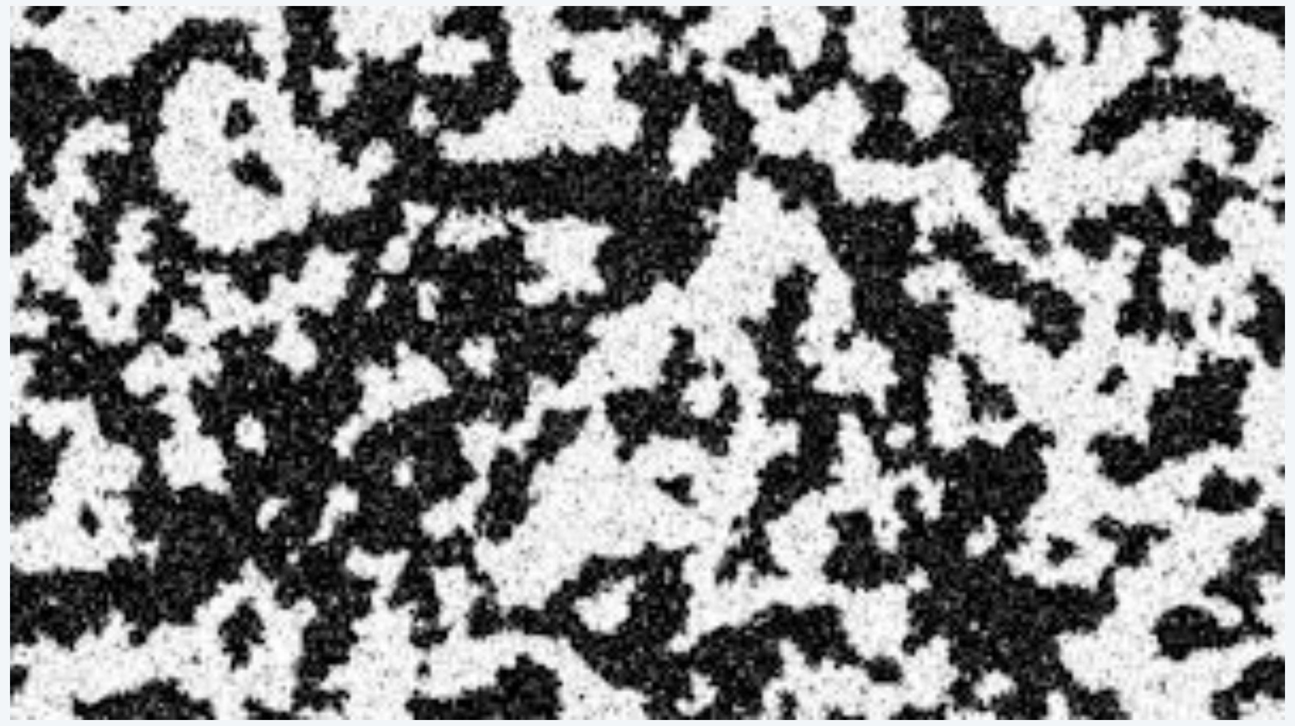
1930s. Closed form solution is a holy grail of statistical mechanics.

1944. Onsager finds closed form solution to 2D version in tour de force.

1950s. Feynman (and others) seek closed form solution to 3D version.

2000. Istrail shows that ISING-3D is **NP**-complete.

Bottom line. Search for a closed formula seems futile.



Summary

P. Set of search problems solvable in poly-time.

NP. Set of search problems (checkable in poly-time).

NP-complete. "Hardest" problems in **NP**. ← SAT, LONGEST-PATH, ILP, TSP, …

Use theory as a guide

- You will confront **NP**-complete problems in your career.
- An poly-time algorithm for an **NP**-complete problem would be a stunning scientific breakthrough (a proof that **P** = **NP**).
- It is safe to assume that **P** ≠ **NP** and that such problems are intractable.
- Identify these situations and proceed accordingly.

Credits

image	source	license
Gears	Adobe Stock	Education License
Finding a Needle in a Haystack	Basic Vision	
Galactic Computer	Adobe Stock	Education License
Taylor Swift Caricature	Cory Jensen	CC BY-NC-ND
Fans in a Stadium	Adobe Stock	Education License
P and NP cookbooks	Futurama S2E10	
Homer Simpson and P = NP	Simpsons	
Archimedes, Lever, and Fulcrum	unknown	
COS Building, Western Wall	Kevin Wayne	
Richard Karp	Berkeley EECS	
Stephen Cook	U. Toronto	
Leonid Levin	Wikimedia	CC BY-SA 3.0
Garey–Johnson Cartoon Updated	Stefan Szeider	CC BY 4.0
Cartoon of Turing Machine	Tom Dunne	
Warning sign	Adobe Stock	Education License
Glass with water	Adobe Stock	Education License
John Nash	Wikimedia	CC BY-SA 3.0

A final thought

“ Now my general conjecture is as follows: for almost all sufficiently complex types of enciphering, […] the mean key computation length increases exponentially with the length of the key […].

The nature of this conjecture is such that I cannot prove it […].
Nor do I expect it to be proven. ”

— John Nash