https://algs4.cs.princeton.edu

Intractability

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

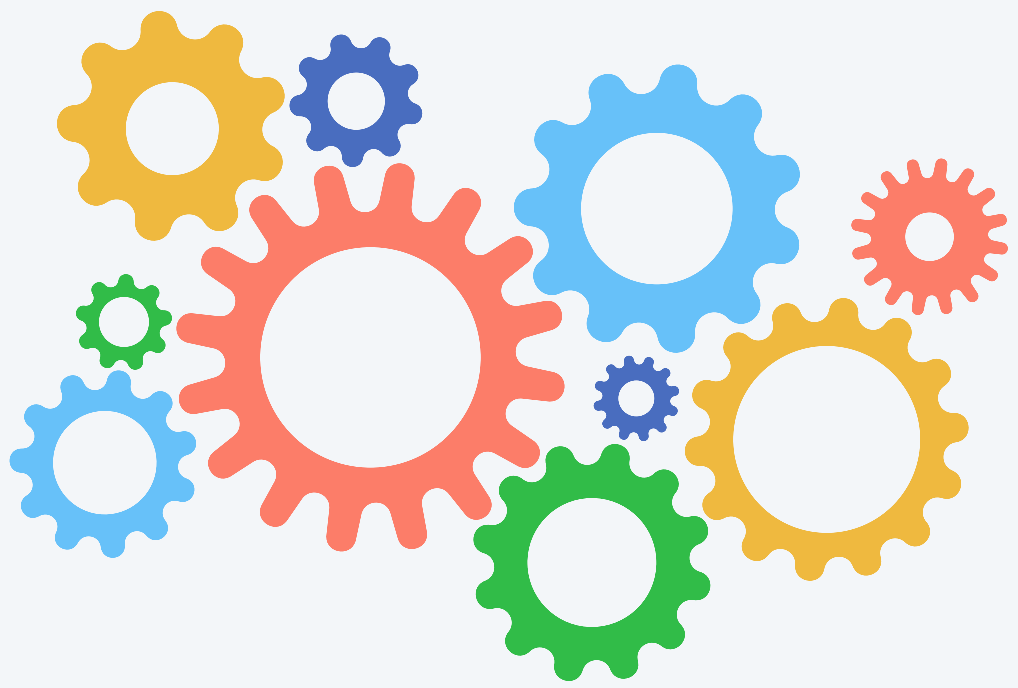
Overview: introduction to advanced topics

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/classes.
- 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?

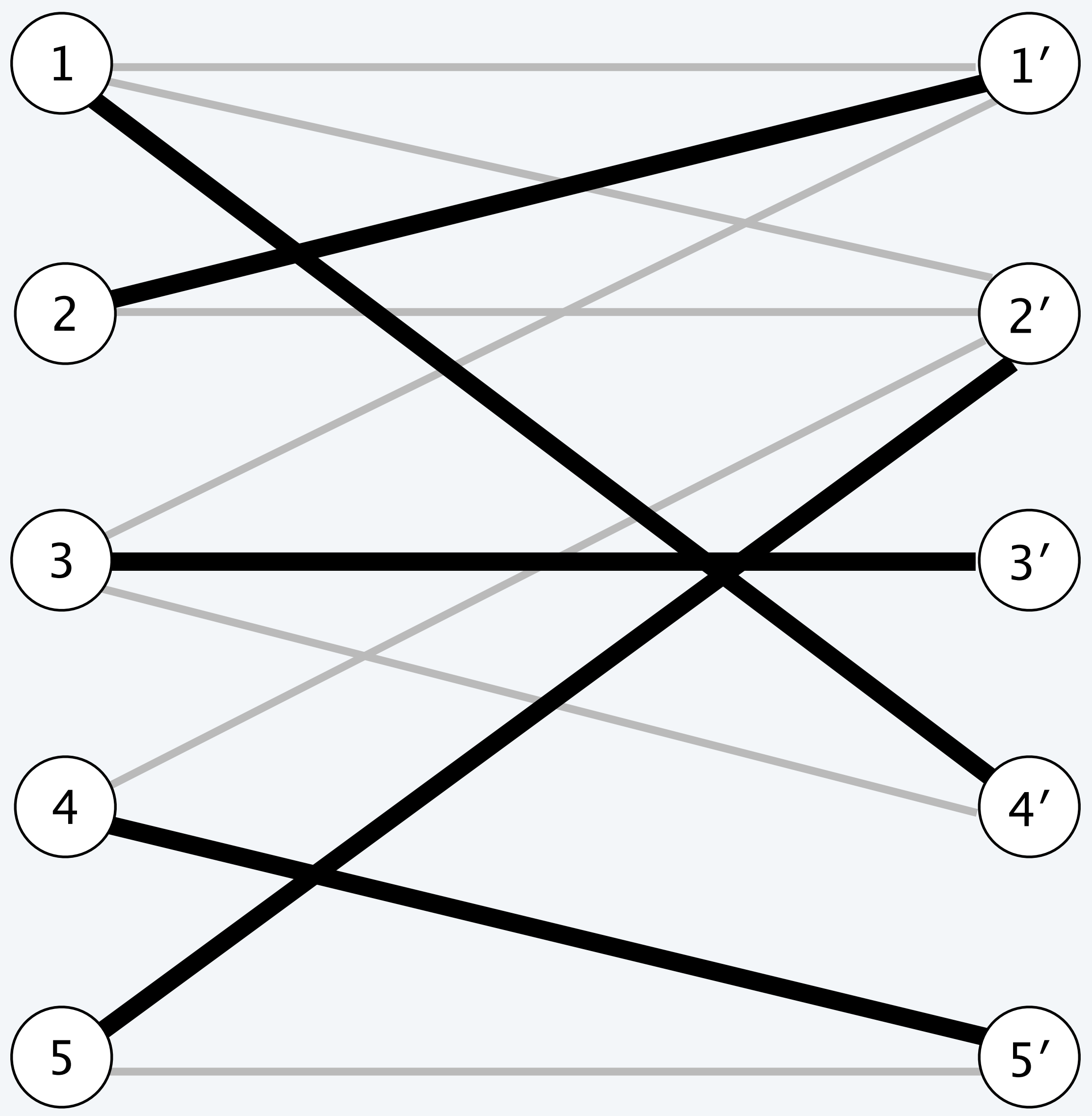
Q3. Which problems can be solved efficiently and which are intractable?

Q4. How can we prove that a problem is intractable?

A computationally easy problem: perfect matching

Perfect matching (search). Given a bipartite graph, find a perfect matching ← *or report that no such matching exists*
(set of edges such that every vertex is an endpoint of exactly one edge in the set).

bipartite graph



perfect matching

1–4′
2–1′
3–3′
4–5′
5–2′

A difficult problem: integer factorization

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

neither 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 $\sqrt{x}$

Q. Can we do anything substantially more clever?

Another difficult problem: boolean satisfiability

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

CNF, conjunctive normal form (AND of ORs)

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 (search). Given a system of boolean equations, find a satisfying truth assignment.

Ex.

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

a SAT instance

needle in a haystack

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: $2^{1000} > 10^{300} \gg 10^{79} \cdot 10^{13} \cdot 10^{17} = 10^{109}$.

Lesson. Exponential growth dwarfs technological change.

Polynomial time

Q2. What is an **efficient algorithm**?

n = # of bits in input

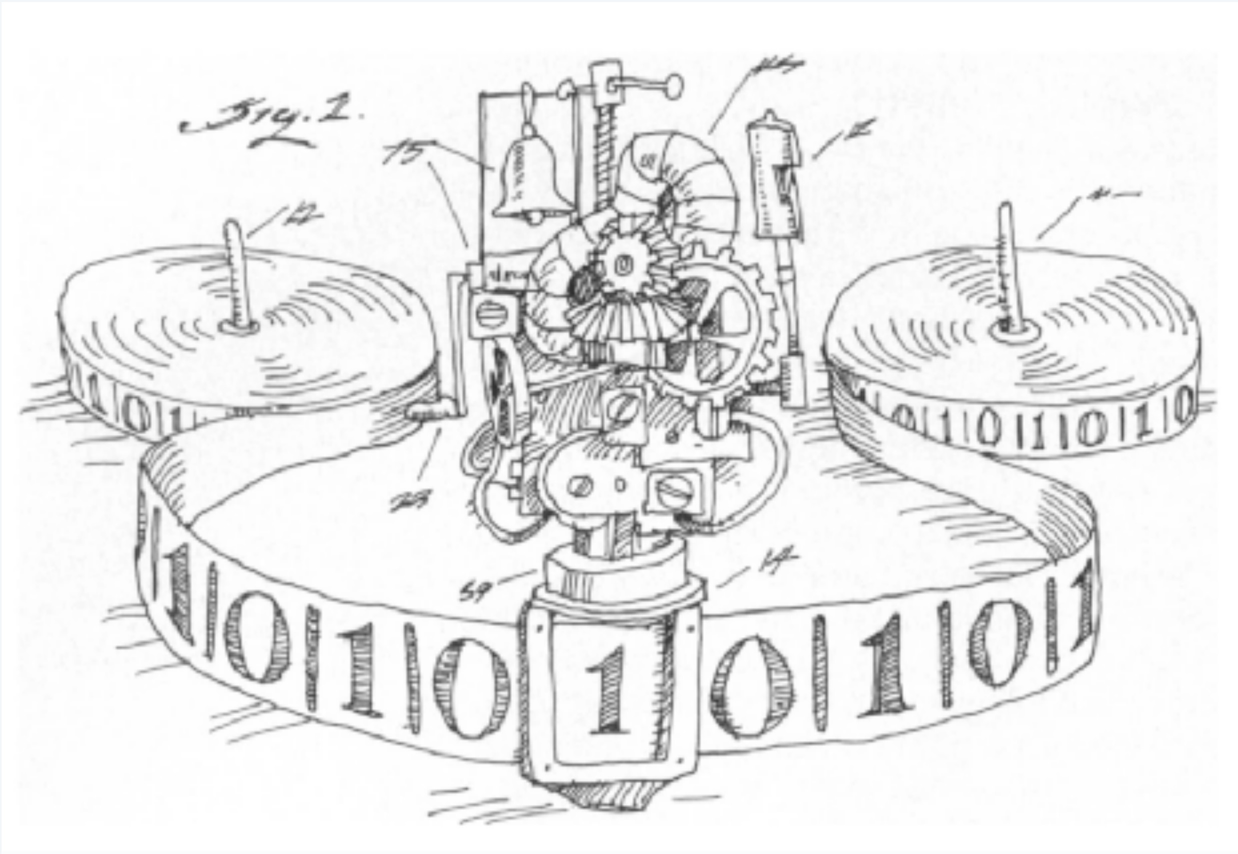
A2. Algorithm whose running time is at most polynomial in input size n.

Polynomial time. Number of *elementary operations* is at most an^b for some constant a and b. ← *must hold for all inputs of size n*

Q1. What is an **algorithm**?

A1. A **Turing Machine**! Equivalently, a program in Java/Python/C++/…

the extended Church-Turing thesis



A Turing machine

order	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 boolean satisfiability.

B. Brute-force search for integer factorization.

C. Both A and B.

D. Neither A nor B.

Intractable problems

Q3. Which problems can be solved efficiently?

A3. Those for which poly-time algorithms exist.

Why do we define poly-time as efficient?

Def. A problem is intractable if no poly-time algorithm solves it.

Q4. How can we prove that a problem is intractable?

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

Often times, efficient algorithms require deep math insights.

tractable	intractable?
primality	*integer factorization*
shortest path	*longest path*
min cut	*max cut*
Euler cycle	*Hamiltonian cycle*
2-SAT	3-SAT
⋮	⋮

require math insights

3 boolean variables per equation

Intractable problems

Algorithms

Robert Sedgewick | Kevin Wayne

https://algs4.cs.princeton.edu

Intractability

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

The P complexity class

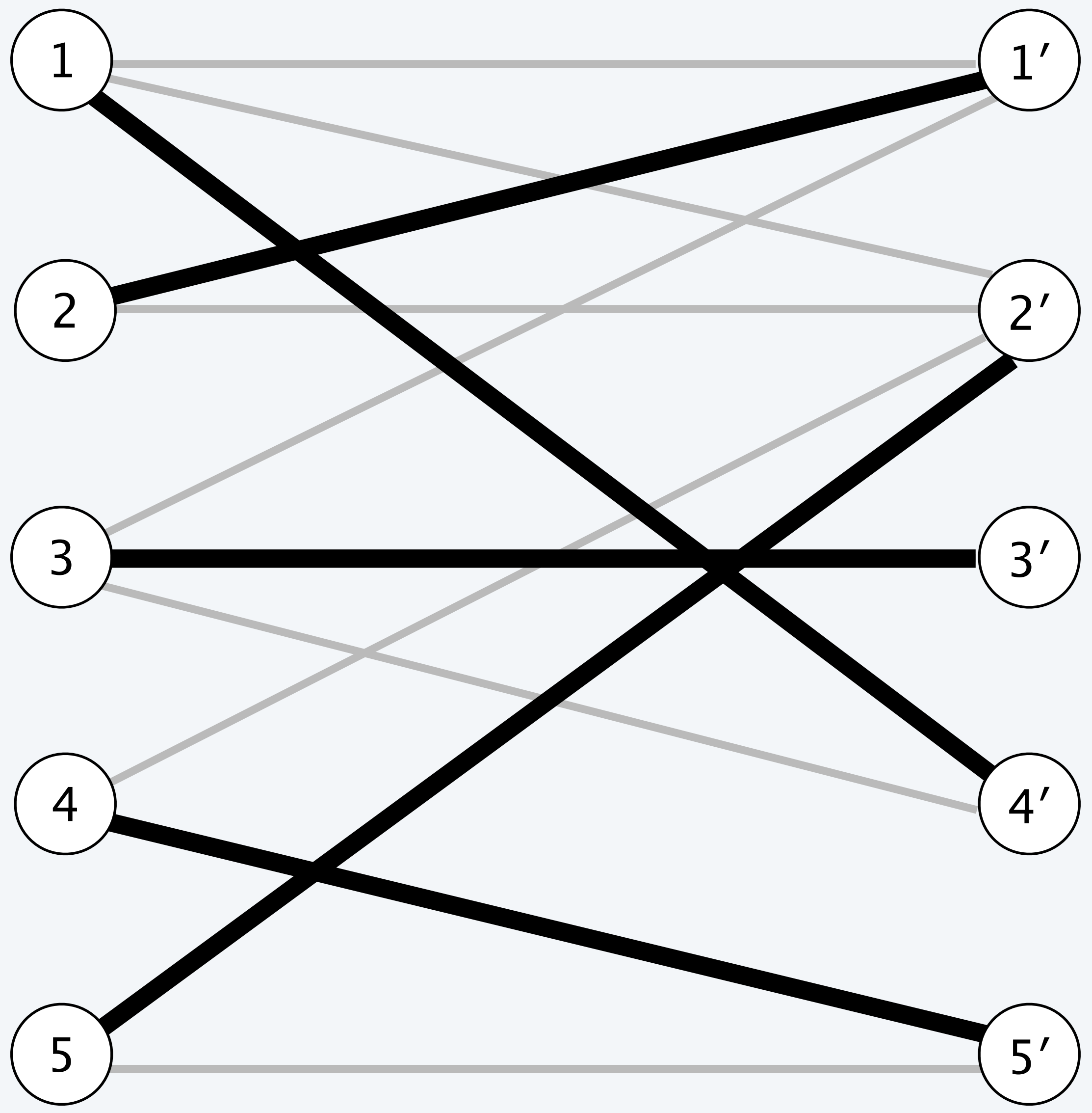
A decision problem is Boolean function (given an input answer YES/NO).

Def. **P** is the set of all decision problems that can be solved in poly-time.

Ex 1 – Perfect matching (decision): Given a bipartite graph, *is there* a perfect matching?

trick – max flow!

bipartite graph



perfect matching

1–4′
2–1′
3–3′
4–5′
5–2′

Are all “interesting” problems in **P**? Maybe there is always a clever trick for solving…

The NP complexity class

Def. **NP** is the set of all decision problems for which you can verify a YES answer in poly-time given a "witness" (a.k.a "proof", "certificate"). ← ***NP** = Nondeterministic Poly-time*

Ex 1 – boolean satisfiability (*decision*): Given a system of m boolean equations in n variables, *is there* an assignment that satisfies all equations?

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

A SAT instance

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

witness

Witness. A satisfying assignment.

Poly-time verification algorithm. Plug values of assignment into the equations and check. ← *verify in $O(mn)$ time*

The NP complexity class

Def. **NP** is the set of all decision problems for which you can verify a YES answer in poly-time given a "witness" (a.k.a "proof", "certificate").

Ex 2 – integer factorization (*decision*): Given two integers x and k, does x have a nontrivial factor greater than k ?

$x = 147573952589676412927$	$k = 100{,}000{,}000$	$193{,}707{,}721$
a FACTOR instance		witness

Witness. A nontrivial factor of x greater than k.

Poly-time verification algorithm. Check that the witness is greater than k and that it's a divisor of x. ← *$O(n^2)$ time via long division*

Note: For a problem to be in **NP**, it suffices to verify a *purported* witness for a *YES* answer.

- Doesn't need to *find* the witness (e.g., a candidate factor is given).
- Doesn't need verify a *NO* answer (e.g., no factor greater than k).

Which decision version of longest path is in NP?

A. Given a graph G and an integer k, is the longest simple path in G of length *at most* k edges.

B. Given a graph G and an integer k, is the longest simple path in G of length *at least* k edges.

C. Both A and B.

D. Neither A nor B.

P vs. NP

P = set of problems whose solution can be *computed* efficiently (in poly-time).

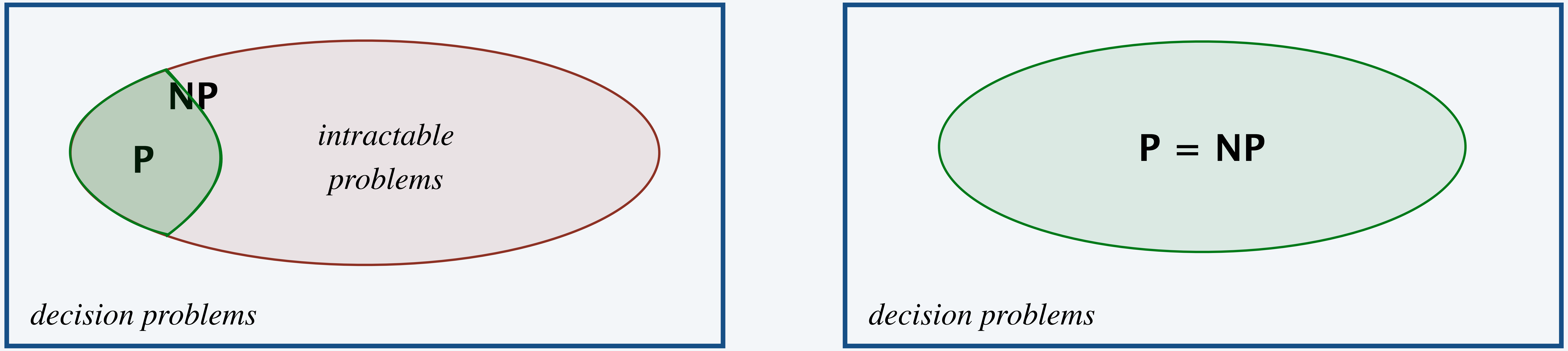
NP = set of problems whose solution can be *verified* efficiently (in poly-time).

any string serves as witness

Observation. **NP** contains **P** ← *e.g., perfect matching is in NP*

THE question. Does **P** = **NP** ? ← *$ 1M*

Two possible worlds.



P ≠ NP

brute-force search may be the best we can do

P = NP

poly-time algorithms for FACTOR, SAT, LONGEST-PATH, …

why?

Conjecture. **P ≠ NP**.

Why is P vs NP so central?

P vs **NP** is central in math, science, technology and beyond.

NP models many intellectual challenges humanity faces: *Why would you attempt to solve a problem if you cannot even tell if a solution is good?*

creative genius

domain	problem	witness/solution
mathematics	is a conjecture correct?	mathematical proof
engineering	given constraints (size, weight, energy), find a design (bridge, medicine, computer)	blueprint
science	given data on a phenomenon, find a theory explaining it	a scientific theory
the arts	write a beautiful poem / novel / pop song, draw a beautiful picture	a poem, novel, pop song, drawing

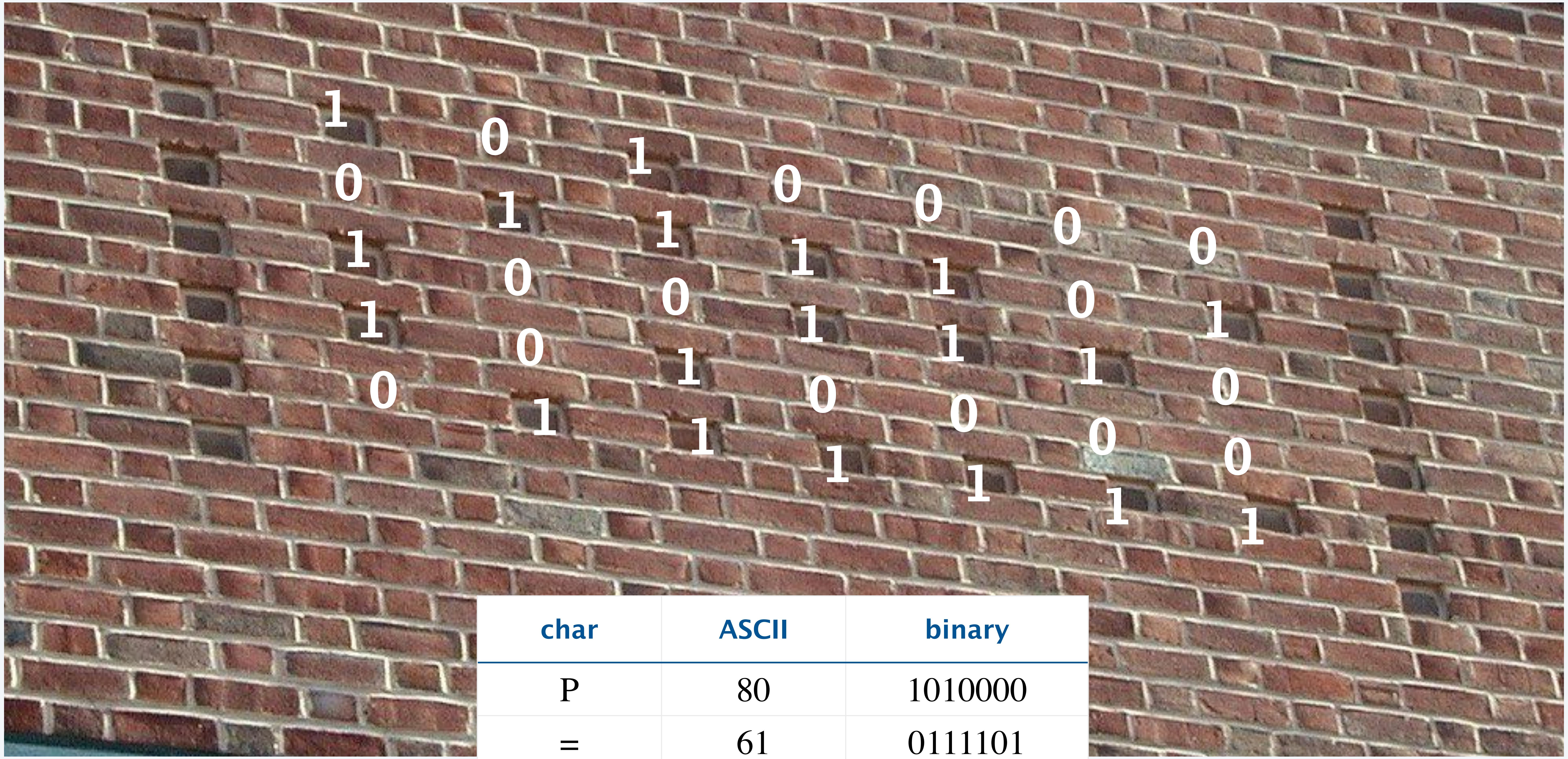
ordinary appreciation

Verifying a solution seems like it should be way easier than finding it! This suggests **P** ≠ **NP**.

Analogy for **P** vs **NP**. Creative genius vs. ordinary appreciation of creativity.

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

https://algs4.cs.princeton.edu

Intractability

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

Bird's-eye view

Goal. Classify problems according to computational requirements.

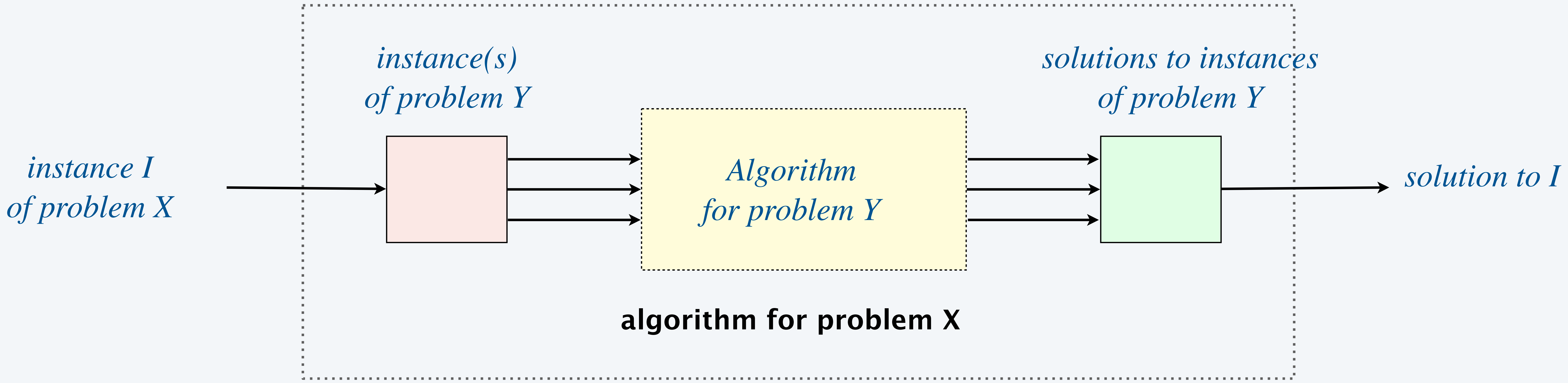
Goal′. Suppose we could (not) solve problem X efficiently.
What else could we (not) solve efficiently?

" 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: ⟵ $X \preceq Y$

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



Ex 1. MEDIAN poly-time reduces to SORT.

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

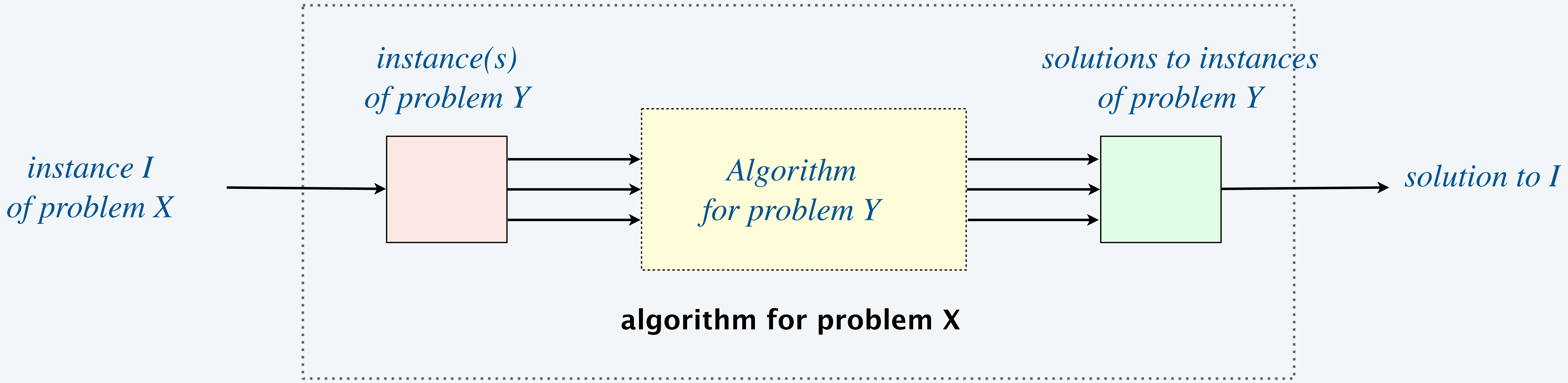
Design algorithms. If X poly-time reduces to Y and Y can be solved efficiently, then X can be solved efficiently.

Establish intractability. If SAT is intractable and SAT poly-time reduces to Y, then Y is intractable.

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.



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

X reduces to SAT: X is no harder than SAT (A solution to SAT implies a solution to X)

SAT reduces to X: X is no easier than SAT (A solution to X implies a solution to SAT)

"up to polynomials"

Integer linear programming

ILP. Given a system of linear inequalities, is there a solution where all variables take integer values?

$$
\begin{array}{rcrcrcrcrcl}
3x_1 & + & 5x_2 & + & 2x_3 & + & x_4 & + & 4x_5 & \geq & 10 \\
5x_1 & + & 2x_2 & & & + & 4x_4 & + & x_5 & \leq & 7 \\
x_1 & & & + & x_3 & + & 2x_4 & & & \leq & 2 \\
3x_1 & & & + & 4x_3 & + & 7x_4 & & & \leq & 7 \\
x_1 & & & & & + & x_4 & & & \leq & 1 \\
x_1 & & & + & x_3 & & & + & x_5 & \leq & 1
\end{array}
$$

linear inequalities

instance I

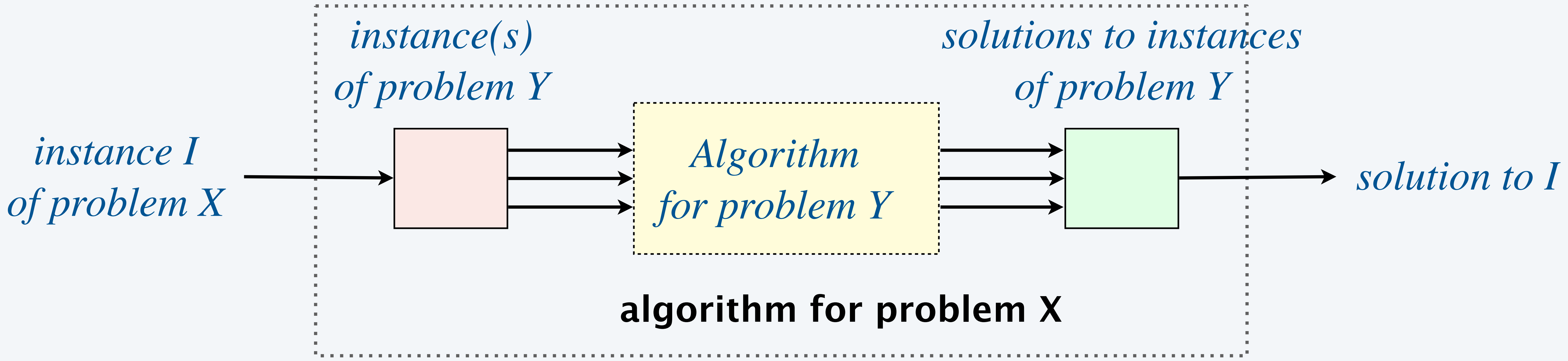
$$
\begin{array}{rcl}
x_1 & = & 0 \\
x_2 & = & 1 \\
x_3 & = & 0 \\
x_4 & = & 1 \\
x_5 & = & 1
\end{array}
$$

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

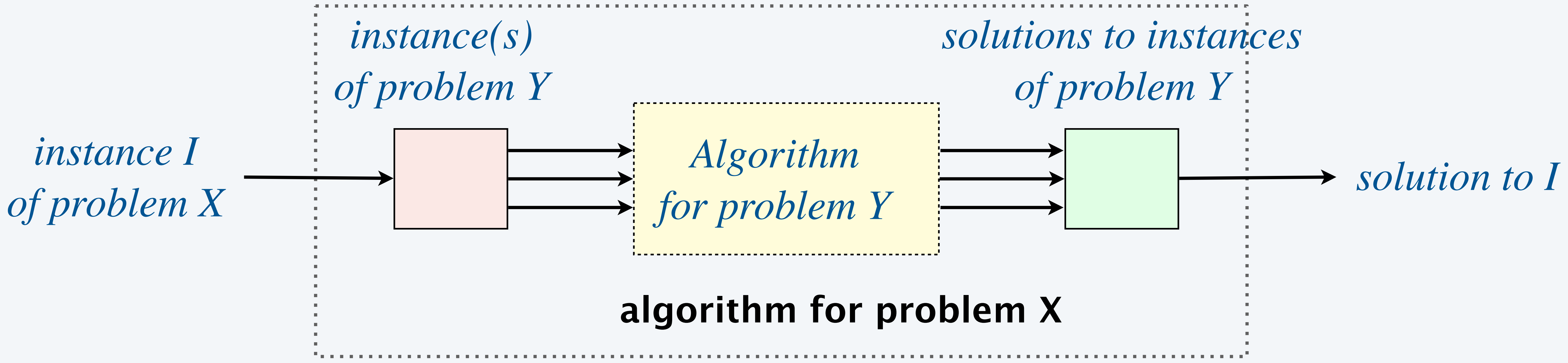


here X = SAT and Y = ILP

SAT poly-time reduces to ILP

SAT. Given a system of m boolean equations in n variables, is there an assignment that satisfies all equations?

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



here X = SAT and Y = ILP

ILP. Given a system of linear inequalities, is there an assignment where all variables take integer values?

$$
\begin{array}{ccccccccc}
(1 - y_1) & + & y_2 & + & y_3 & & & \geq & 1 \\
y_1 & + & (1 - y_2) & + & y_3 & & & \geq & 1 \\
(1 - y) & + & (1 - y_2) & + & (1 - y_3) & & y_4 & \geq & 1 \\
(1 - y_1) & + & (1 - y_2) & + & & + & y_4 & \geq & 1 \\
 & & (1 - y_2) & + & y_3 & + & y_4 & \geq & 1
\end{array}
$$

$y_i = 0 \;\Rightarrow\; x_i = false$

$y_i = 1 \;\Rightarrow\; x_i = true$

$$
\begin{array}{ccccc}
0 & \leq & y_1 & \leq & 1 \\
0 & \leq & y_2 & \leq & 1 \\
0 & \leq & y_3 & \leq & 1 \\
0 & \leq & y_4 & \leq & 1
\end{array}
$$

Intractability: quiz 3

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

Which of the following can we infer?

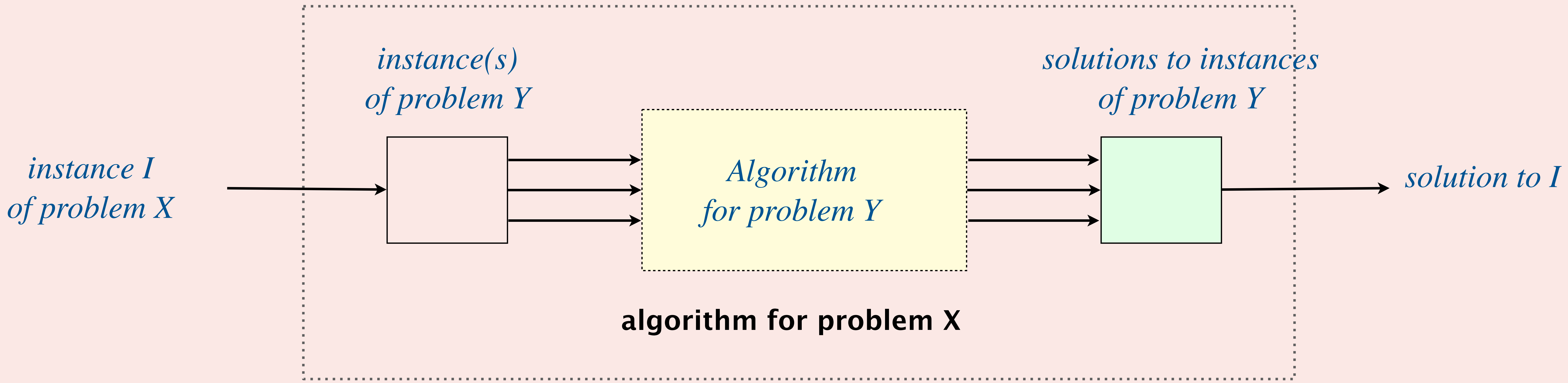
A. If X can be solved in poly-time, then so can Y.

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

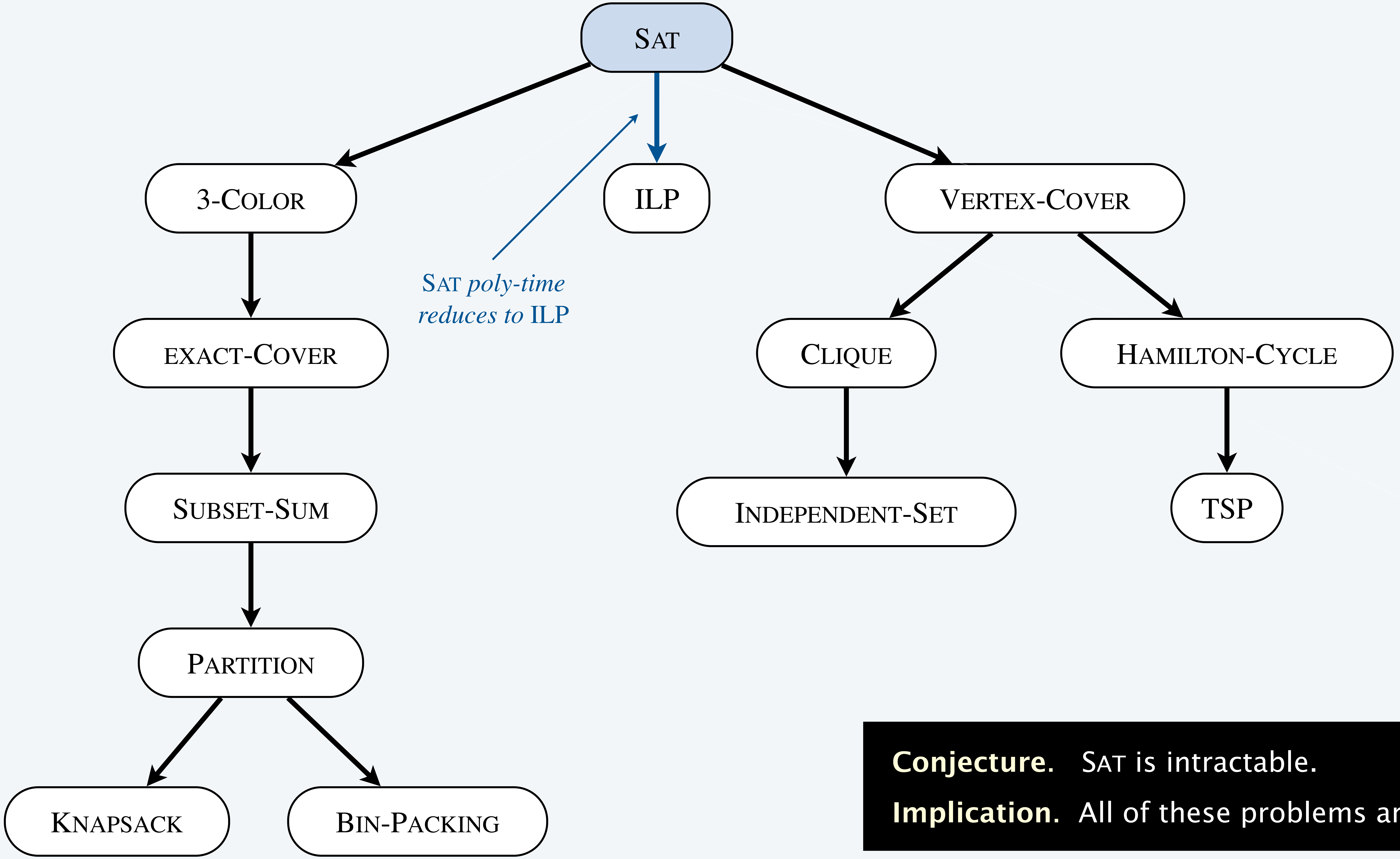
C. If Y can be solved in $\Theta(n^3)$ time, then X can be solved in poly-time.

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

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



More poly-time reductions from SAT



Richard Karp
(1972)

Conjecture. SAT is intractable.

Implication. All of these problems are intractable.

https://algs4.cs.princeton.edu

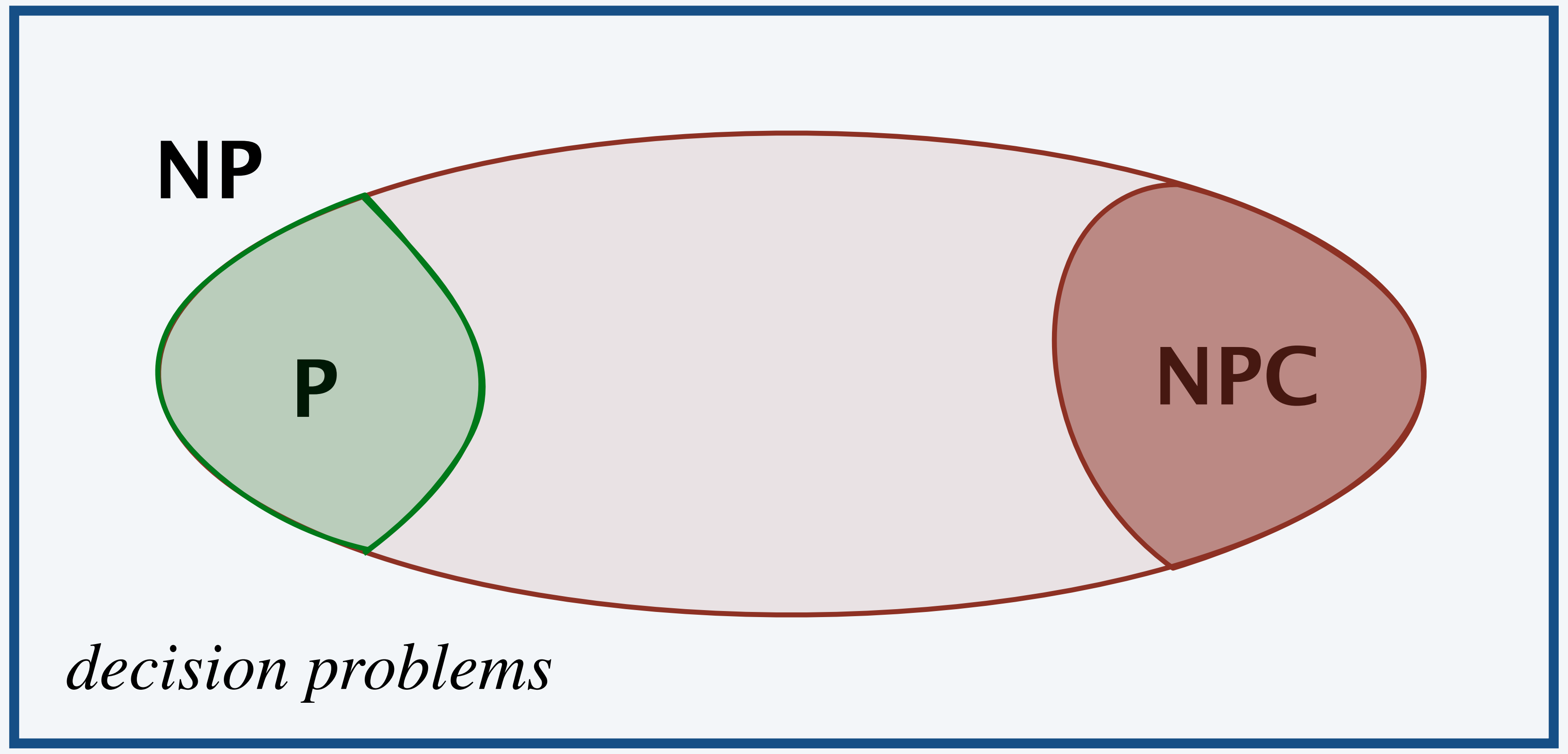
Intractability

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

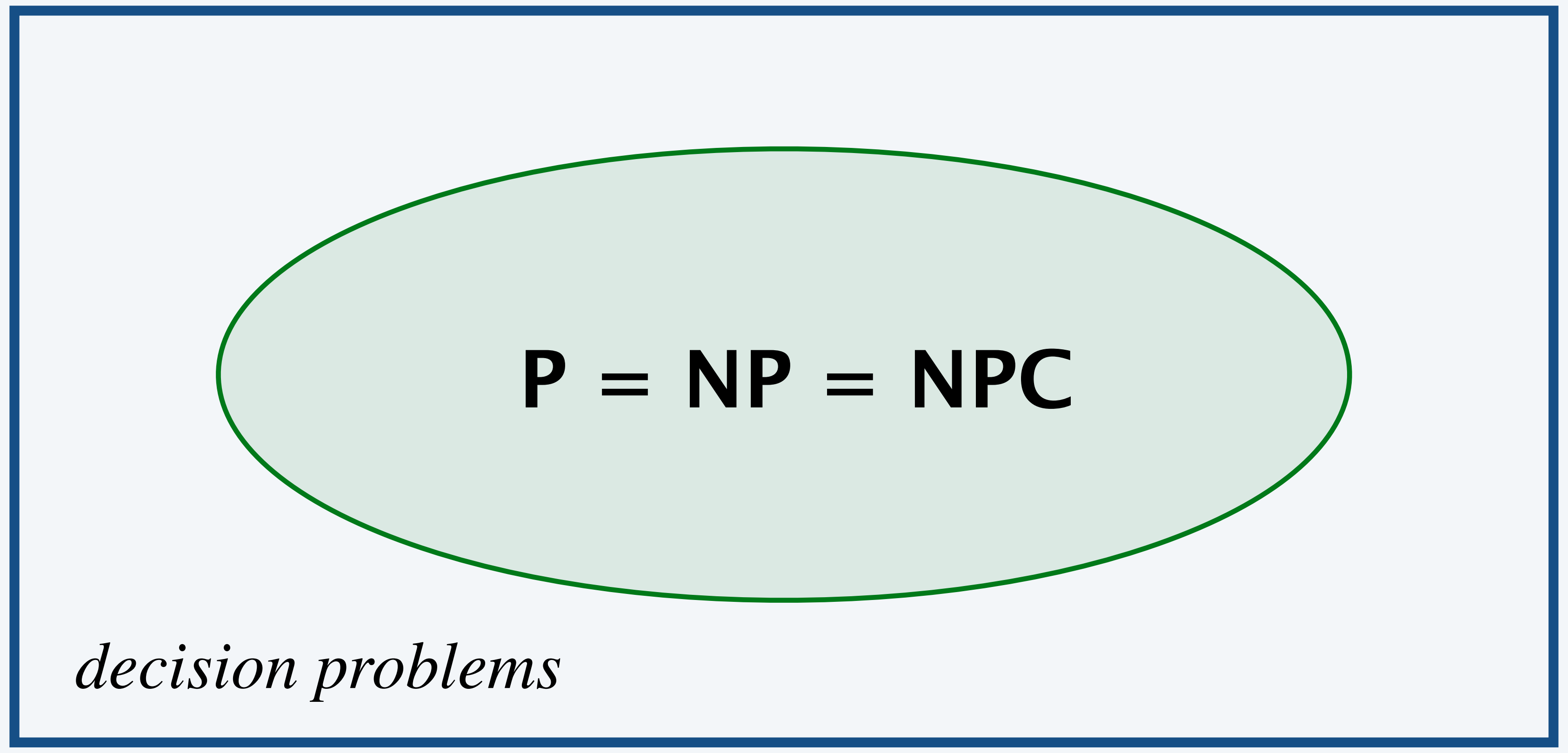
NP-completeness

Def. A decision 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

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**.

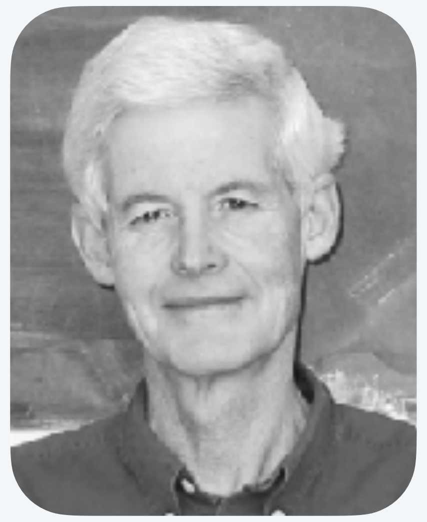
A. I only.

B. II only.

C. I and II only.

D. I, II, and III.

Cook–Levin theorem


Stephen Cook (1971)


Leonid Levin (1971)

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

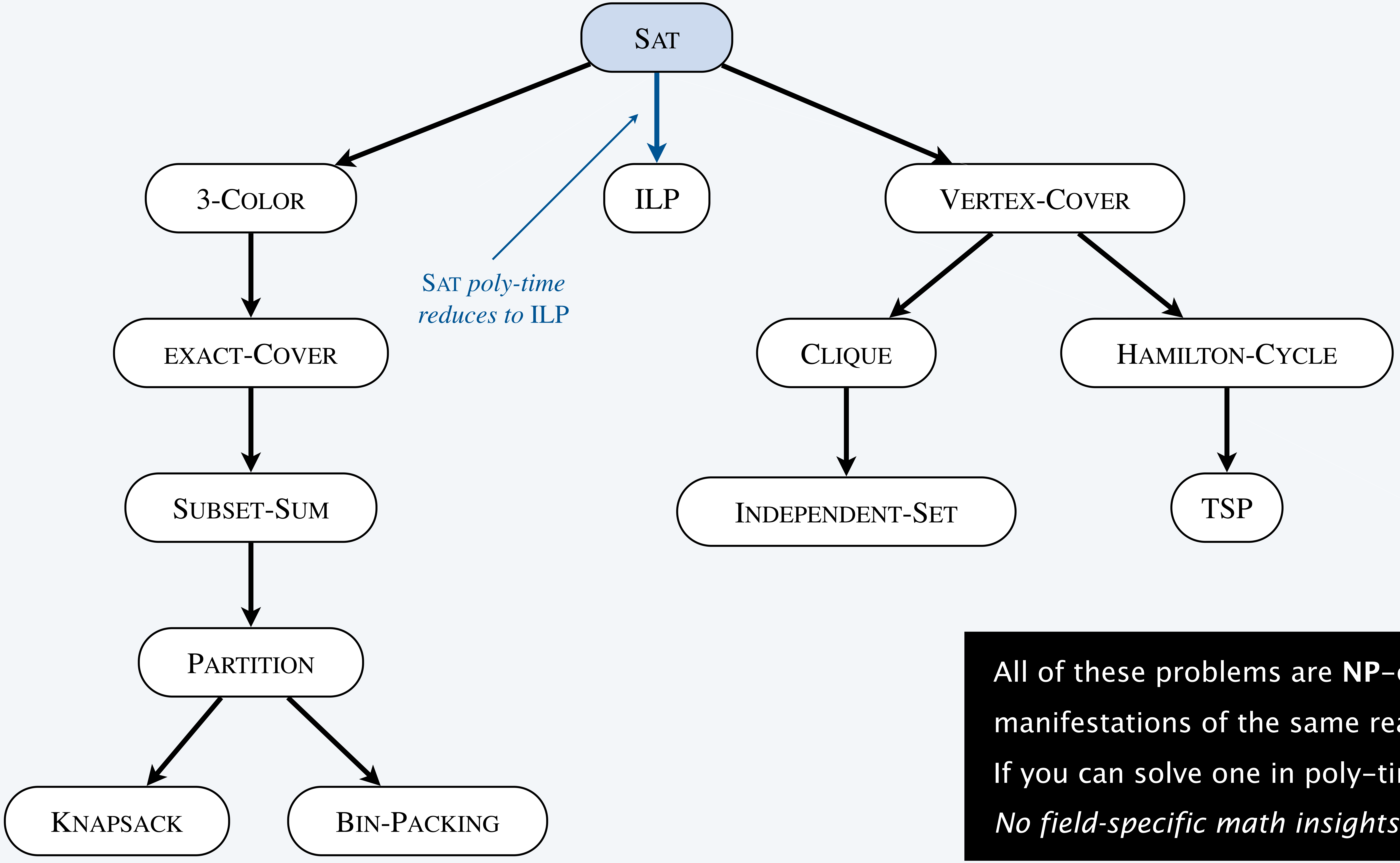
Pioneering result in computer science!

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

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

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

Implications of Karp + Cook–Levin



Richard Karp
(1972)

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

If you can solve one in poly–time, you can solve all!

No field-specific math insights are needed!

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.*

Algorithms

Robert Sedgewick | Kevin Wayne

https://algs4.cs.princeton.edu

Intractability

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

Dealing with intractability

Identifying intractable problems

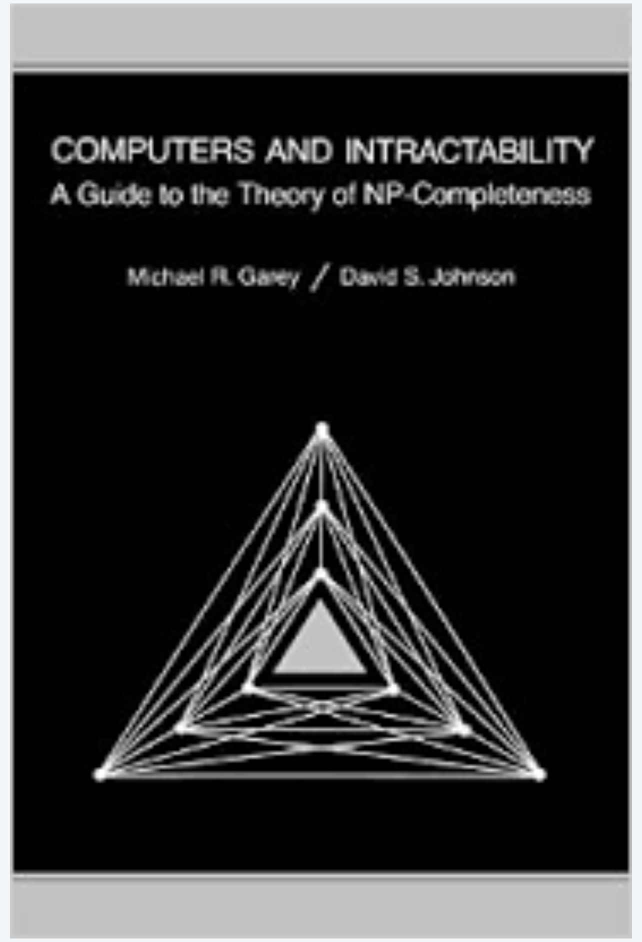
Establishing **NP**-completeness through poly-time reduction is an important tool in guiding algorithm design efforts.

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 any other **NP**-complete problem).

Caveat. Intricate reductions are common.



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/best* 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.

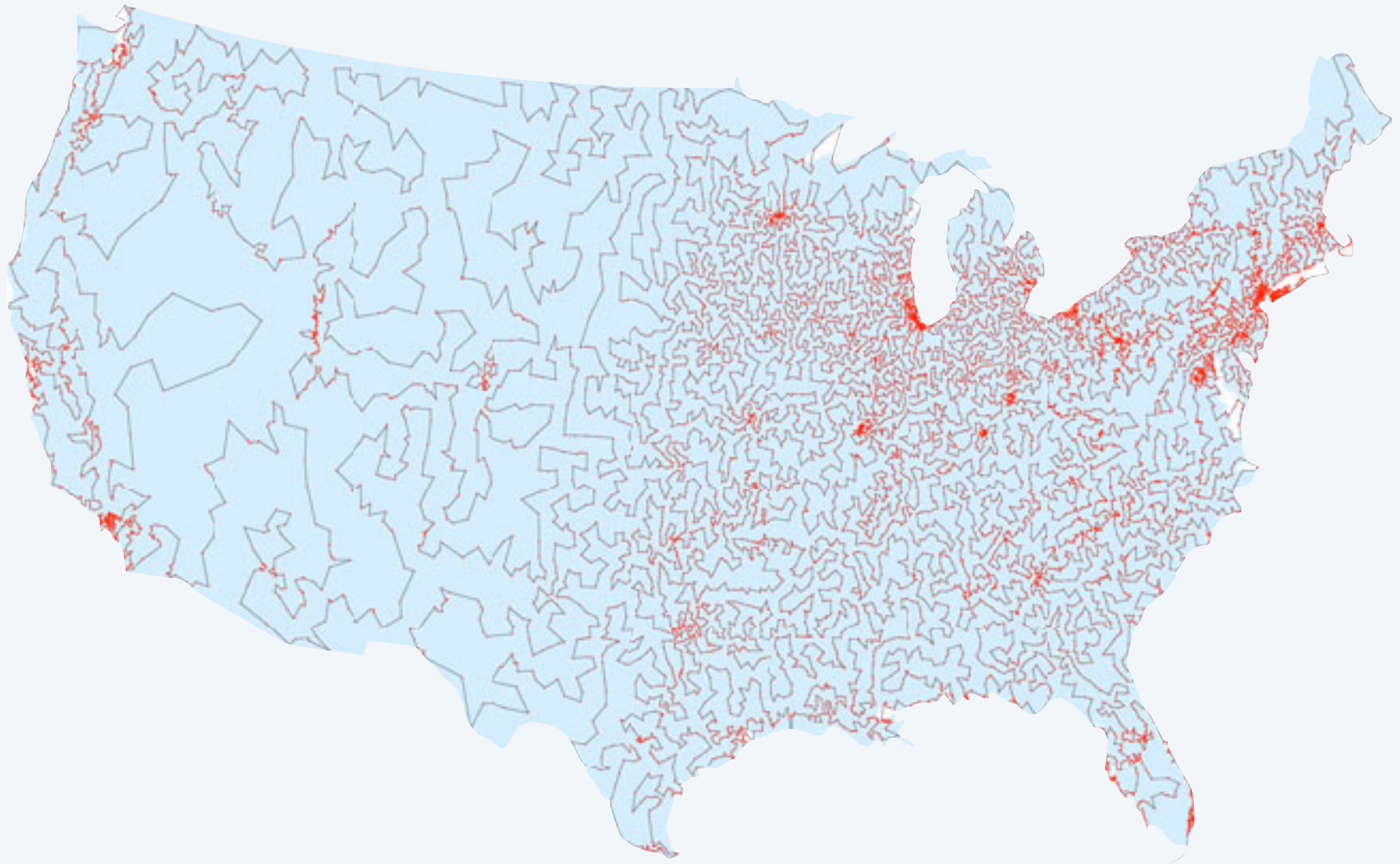
- Worst-case inputs may not occur for practical problems.
- Instances that do occur in practice may be easier to solve.
- Reasonable approach: relax the condition of guaranteed poly-time.

Boolean satisfiability.

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

Traveling salesperson problem.

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



TSP solution for 13,509 US cities

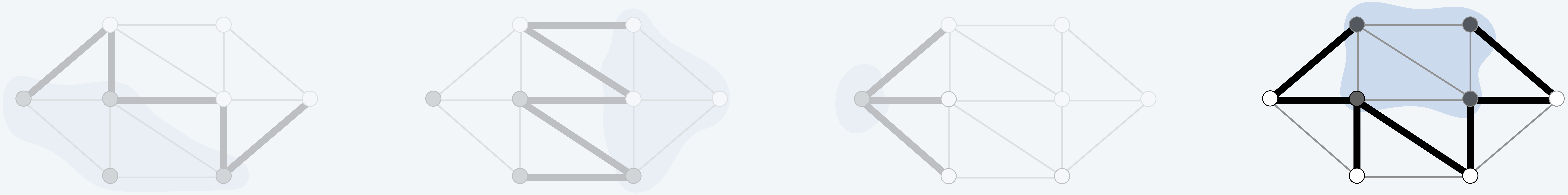
Integer linear programming.

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

Dealing with intractability: approximation algorithms

MAX-CUT (search): 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 (search): 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 $7/8$.

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.

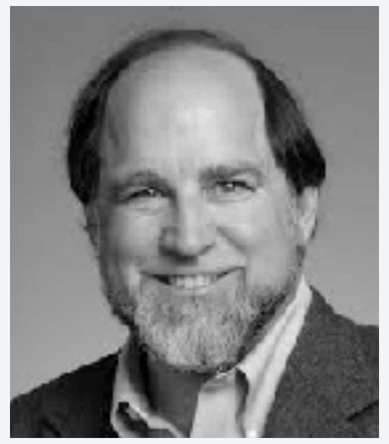
- Secure a secret communication.
- Append a digital signature.
- Credit card transactions.
- …

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

multiply
(easy)

$761838257287 \times 193707721$

147573952589676412927

factor
(difficult)

Leveraging intractability: guiding scientific inquiry

1926. Ising introduces a mathematical model for ferromagnetism.

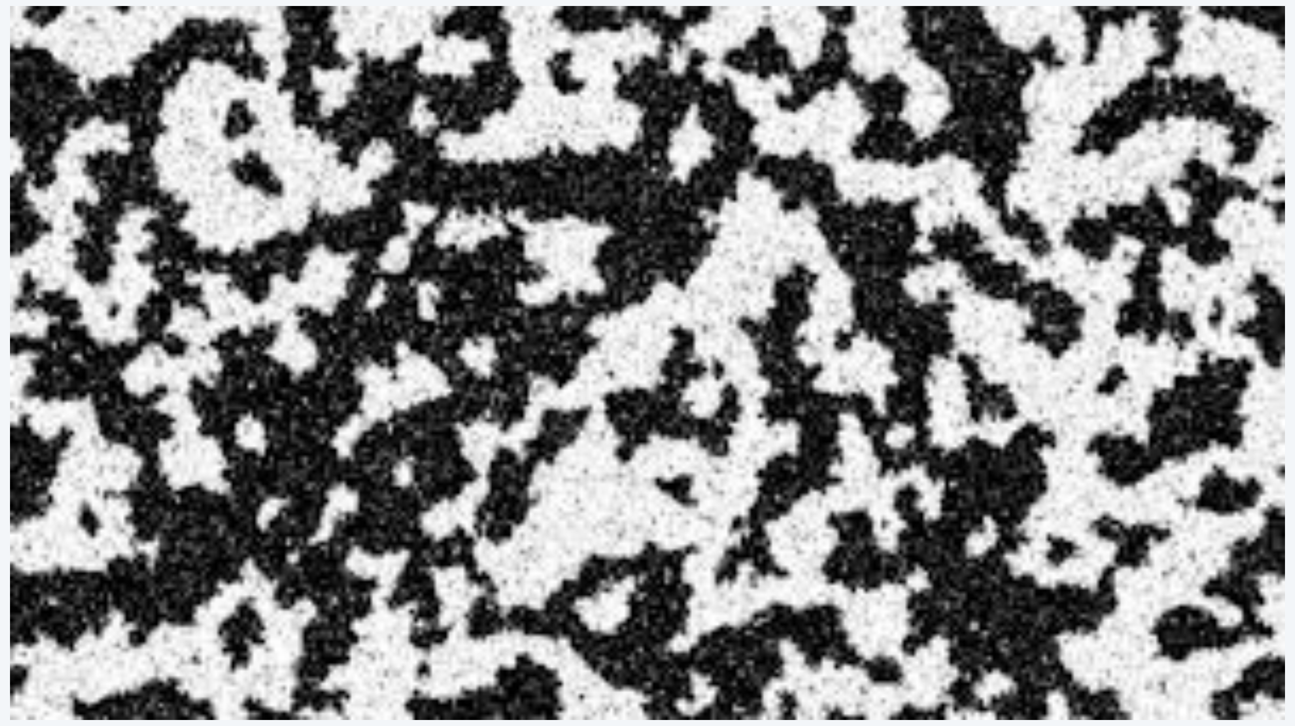
1930s. Closed form solution is a holy grail of statistical mechanics.

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 problems solvable in poly-time.

NP. Set of 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