https://algs4.cs.princeton.edu

Intractability

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

https://algs4.cs.princeton.edu

Intractability

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

Fundamental questions

What is an algorithm?

What is an efficient algorithm?

Which problems can be solved efficiently and which are intractable?

How can we prove that a problem is intractable?

How can we cope with intractability?

How can we benefit from intractability?

Multiplication

$$37 \cdot 79 = \;?$$

Factorization

$$? \cdot ? = 2881$$

Slightly bigger multiplication

33478071698956898
78604416984821269
08177047949837137
68568912431388982
88379387800228761
47116525317430877
37814467999489

•

36746043666799590
28244633799627952
63227915816434308
76426760322838157
39666511279233373
41714339681027009
2798736308917

= ?

Computed in a split second by a standard laptop!

Slightly bigger factorization

$$? \cdot ? =$$

12301866845301177
55130494958384962
72077285356959533
47921973224521517
26400507263657518
74520219978646938
99564749427740638
45925192557326303
45373154826850791
70261221429134616
70429214311602221
24047927473779408
06653514195974598
56902143413

$50,000
RSA factoring challenge
2 years, team of mathematicians

RSA-768, 232 digits

Multiplication (computationally easy)

Multiplication. Given integers x, y, return xy.

Algorithm. Grade-school multiplication runs in time $\Theta(n^2)$, where n is the number of digits in x, y.

Integer factorization (computationally hard?)

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

neither 1 nor x

Applications. Cryptography. [stay tuned]

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 $\sqrt{x}$

Can we do anything substantially more clever?

boolean satisfiability with 2 vars (computationally easy)

2-SAT (search). Given m boolean equations over the variables $x_1 \ldots x_n$ in the form "y_i *or* y_j = *true*", where y_i is either x_i or $\neg x_i$, return a truth assignment that satisfies all equations.

CNF, conjunctive normal form

or report that no such assignment is possible

Example.

$\neg x_1$	*or*	x_2	=	*true*
x_1	*or*	x_3	=	*true*
$\neg x_2$	*or*	$\neg x_3$	=	*true*
$\neg x_2$	*or*	x_4	=	*true*
x_3	*or*	$\neg x_4$	=	*true*

2-SAT instance

x_1	=	*false*
x_2	=	*false*
x_3	=	*true*
x_4	=	*true*

satisfying assignment

SAT applications.

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

boolean satisfiability with 3 vars (computationally hard?)

3-SAT (search). Same as 2-SAT, but every equation has 3 variables instead of 2.

Example.

$\neg x_1$	*or*	x_2	*or*	x_3	=	*true*
x_1	*or*	$\neg x_3$	*or*	x_4	=	*true*
x_2	*or*	$\neg x_3$	*or*	$\neg x_1$	=	*true*
$\neg x_2$	*or*	x_4	*or*	x_3	=	*true*
$\neg x_3$	*or*	$\neg x_4$	*or*	$\neg x_2$	=	*true*

3-SAT instance

x_1	=	*false*
x_2	=	*false*
x_3	=	*true*
x_4	=	*true*

satisfying assignment

Brute-force search. Try all 2^n possible assignments (n = # variables).

Can we do anything substantially more clever?

Probably not. [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}

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

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

Lesson. Exponential growth dwarfs technological change!

Efficient algorithms

What is an efficient algorithm?

# of bits in the input's representation

Algorithm whose running time is at most polynomial *in the size of the input.*

A problem is efficient/tractable if there exists an efficient (poly-time) algorithm that solves it. Otherwise, it is intractable.

What is an algorithm?

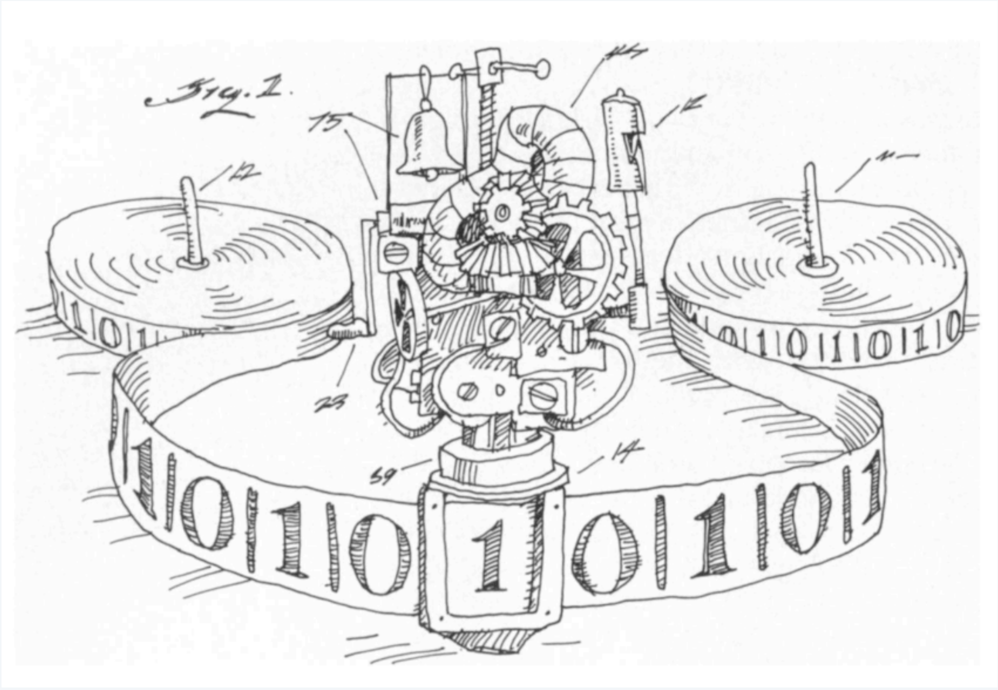
A Turing Machine! Equivalently, a program in Java/Python/C++/…

falsifiable thesis. believed to be false — quantum computers

Extended Church-Turing thesis. Any problem the can be efficiently solved by a physical system can also be efficiently solved by a Turing machine.

is $n^{billion}$ better than $2^{n/billion}$?

Why is polynomial time considered efficient?
robust across models, closed under composition,
most poly-time algos have small exponents.



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*	👿

Which of the following are poly-time algorithms?

- **A.** Brute-force search for 3-SAT.
- **B.** The Ford-Fulkerson algorithm on a weighted graph.
- **C.** Try all factors search for factorization.
- **D.** All of the above.
- **E.** None of the above.

Intractable problems

https://algs4.cs.princeton.edu

Intractability

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

The P complexity class

A decision problem is a Boolean function that, given an input, answers YES/NO.

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

Examples.

2-SAT (decision): Given a system of equations, is there an assignment that satisfies all equations?

Mincut (decision): Given a graph G and integer k, is there a cut in G with $\leq k$ crossing edges?

Multiplication (decision): Given integers x, y, k, is $xy \geq k$?

Primality (decision): Given an integer x, is x prime? ⟵ *first poly-time algorithm in 2002!*

Are all "interesting" problems in **P**? Perhaps there is always a clever algorithm…

The NP complexity class

Def. **NP** is the set of all decision problems for which a YES answer can be verified in polynomial time provided a "witness" (a.k.a "proof", "certificate").

$x = 2881$ $k = 50$

factorization instance

Examples.

43

witness

Factorization (decision): Given integers x, k, does x have a nontrivial factor $\leq k$?

- Witness. A nontrivial factor $f \leq k$ of x.
- Verification. Output YES if $1 < f \leq k$ and f divides x. ← *quadratic time using long division*

3-SAT (decision): Given a system of equations, is there an assignment that satisfies all equations?

- Witness. A satisfying assignment.
- Verification. Output YES if the assignment satisfies all equations.

$x_1 = \textit{false}$
$x_2 = \textit{false}$
$x_3 = \textit{true}$
$x_4 = \textit{true}$

satisfying assignment

Note. A problem is in **NP** if a *purported* witness for a *YES* answer can be verified in poly time:

- It does not require *finding* the witness (e.g., the candidate factor is provided).
- It does not require verifying a *NO* answer (e.g., no factor $\leq k$).

P vs. NP

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

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

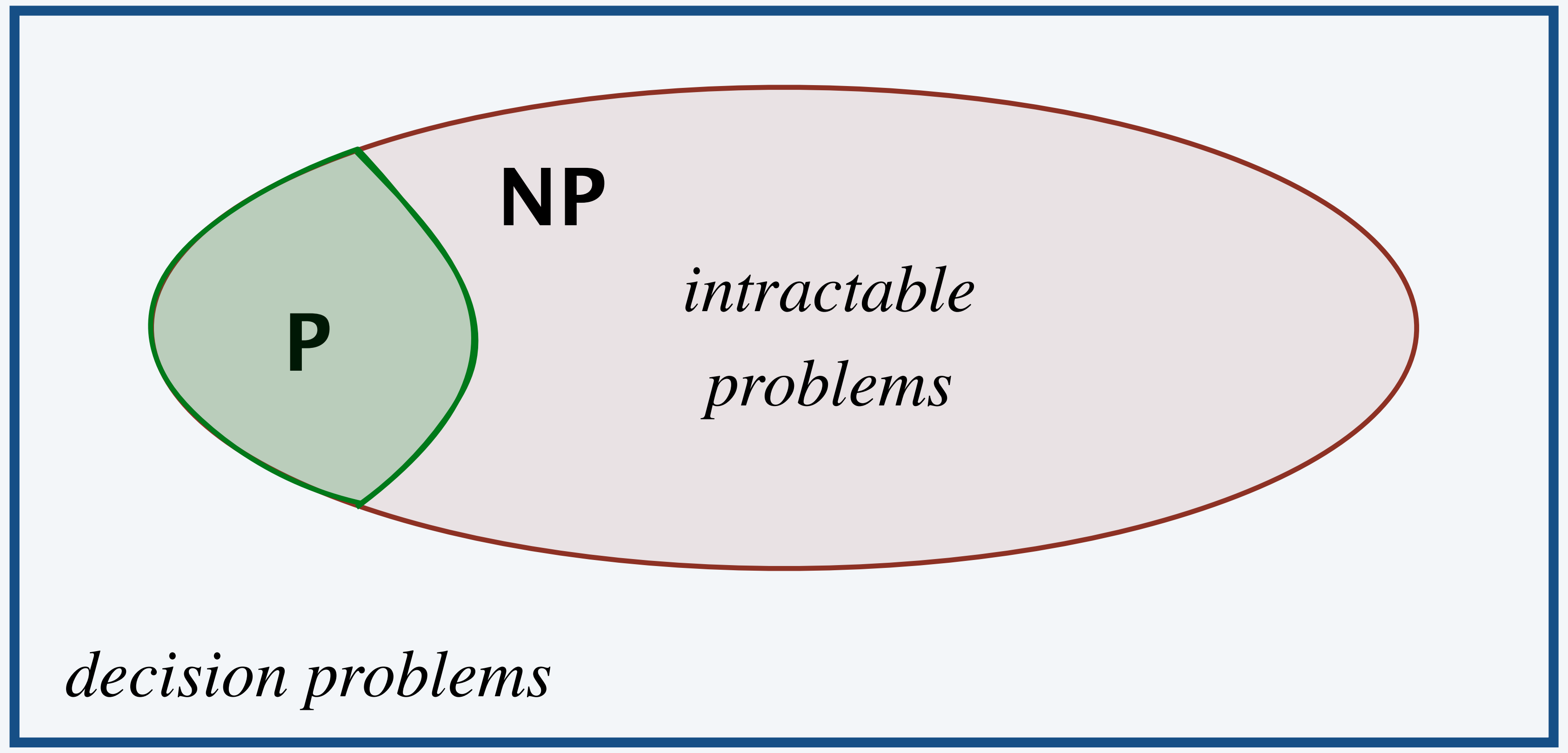
Observation. **NP** contains **P** ⟵ *any string serves as witness*

$ 1M

THE question. **P** = **NP** ?

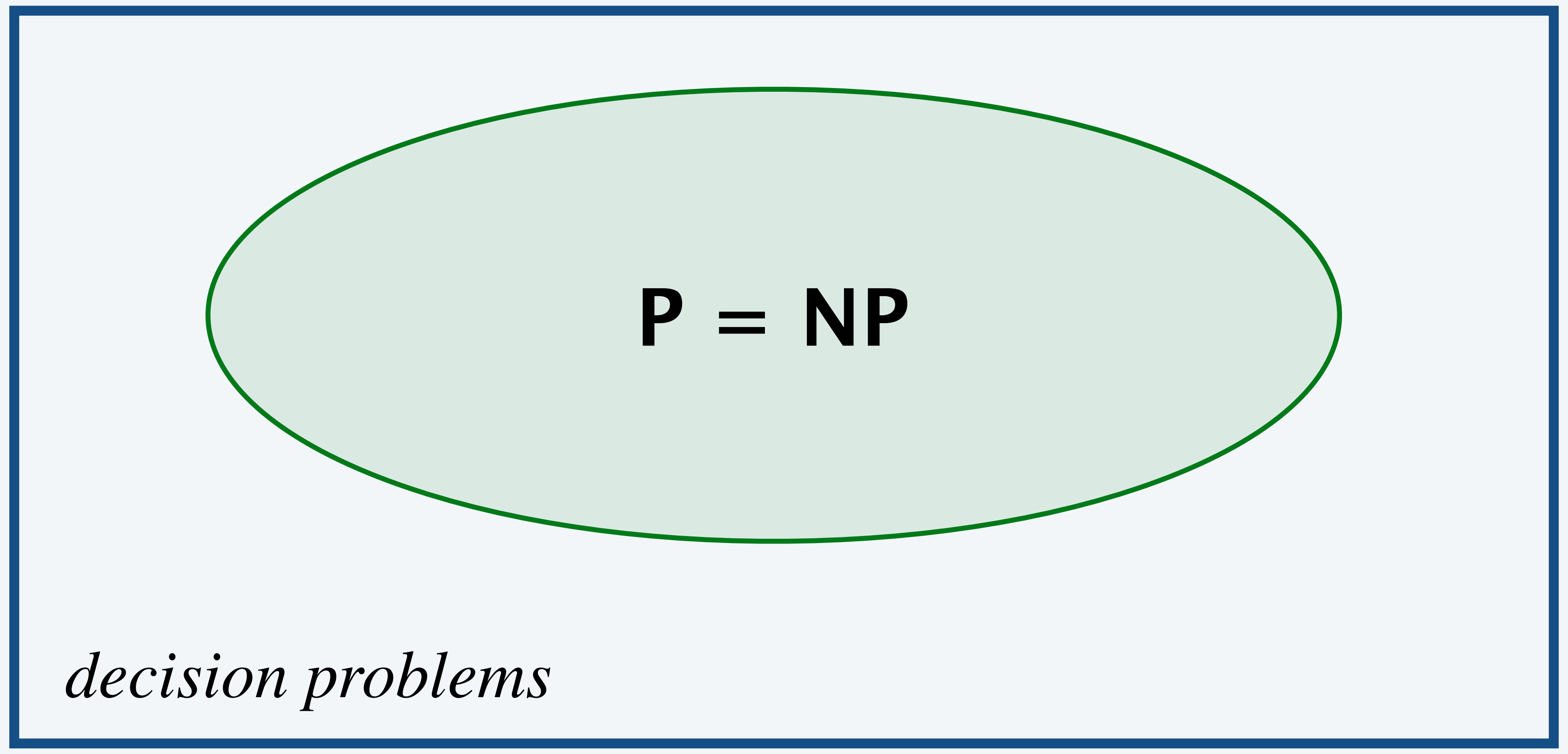
Is *solving* harder than *verifying*?

Two possible worlds.



P ≠ NP

brute-force search may be the best we can do



P = NP

poly-time algorithms for factorization, 3-SAT, maxcut, …

long futile search for poly-time algorithms

↓

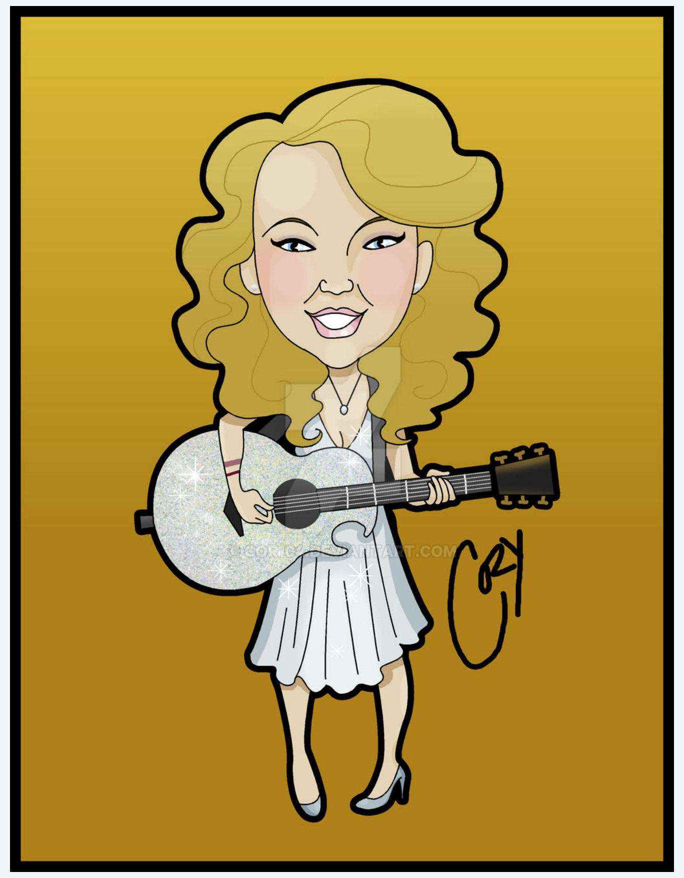
Conjecture. **P ≠ NP**.

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 try to solve a problem if you cannot even determine whether a solution is good?*

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



creative genius

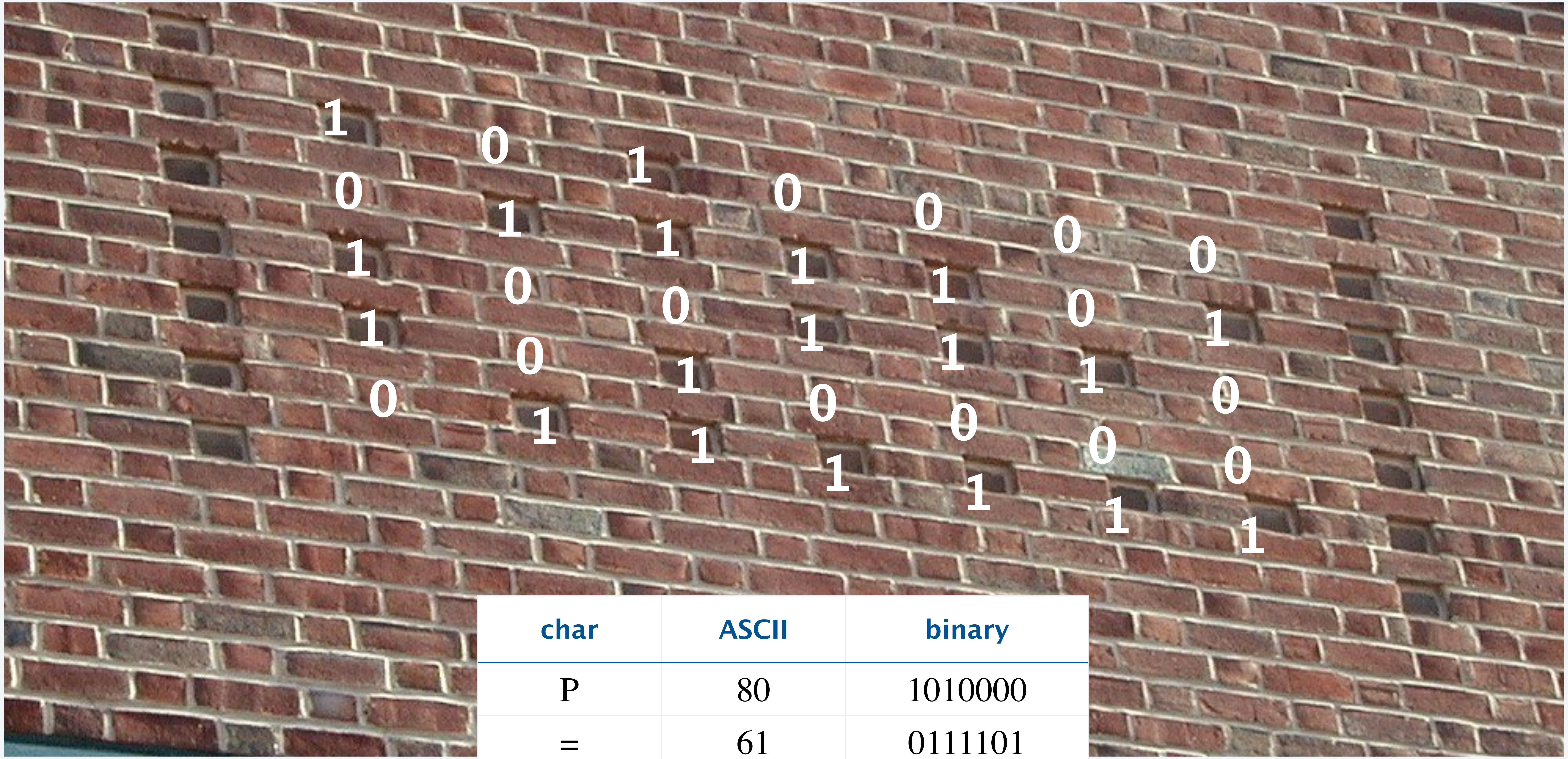
ordinary appreciation

Intuitively, verifying a solution should be way easier than finding it, supporting **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
- Leveraging intractability

Poly-time reduction

Goals.

- Classify problems according to computational requirements.
- If we can (or cannot) solve problem X efficiently, what other problems can (or cannot) be solved efficiently?

"solution to Y implies solution to X"
"Y is harder than X" (up to polys)
denoted $X \preceq Y$

Def. Problem X poly-time reduces to problem Y, if there exists a polynomial $p(n)$ such that any time-$T(n)$ algorithm for Y can be used to construct a time-$T(p(n))$ algorithm for X.

← *formal def in COS 240!*

$T(n) \geq n$

ex. $T(n) = n^2$, $p(n) = n^3$ *implies* $T(p(n)) = n^6$.

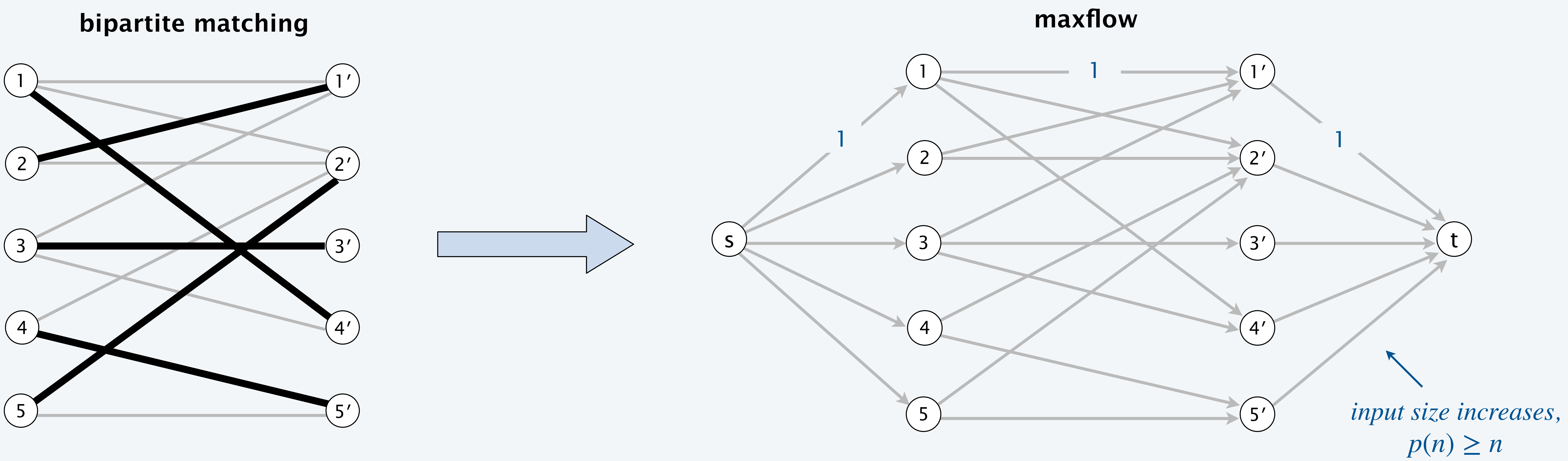
Algorithm design. If $X \preceq Y$ and Y can be solved efficiently, then X can also be solved efficiently.

Establishing intractability. If $X \preceq Y$ and X is intractable, then Y is also intractable.

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

Poly-time reduction example 1

Bipartite matching ≤ maxflow:



Algorithm design. Since maxflow can be solved efficiently, so can bipartite matching.

Poly-time reduction example 2

3-SAT ≤ Clique finding: Given a graph G and integer k, is there a subset with $\geq k$ vertices all pairwise adjacent?

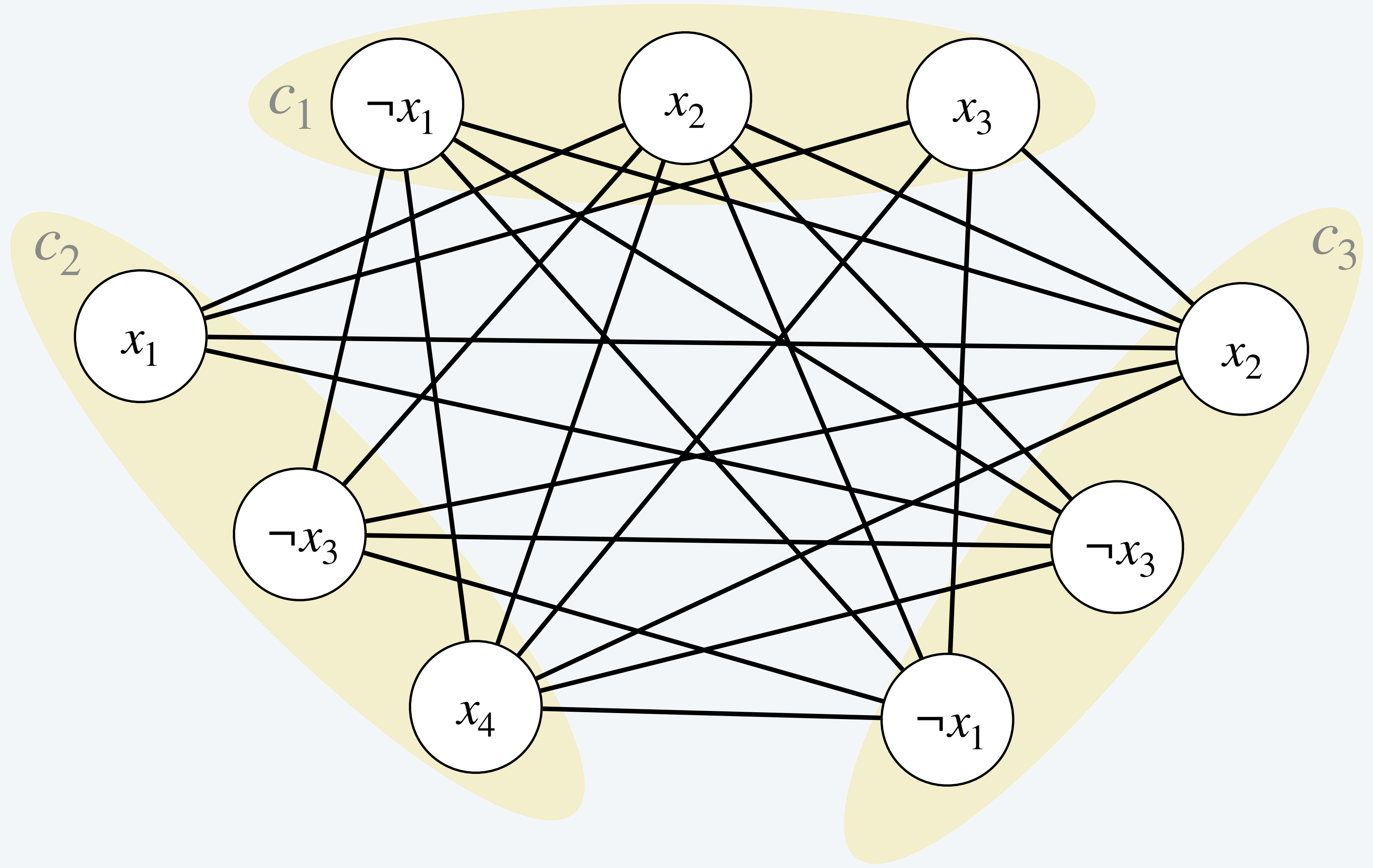
3-SAT instance

c_1	$\neg x_1$	*or*	x_2	*or*	x_3	=	*true*
c_2	x_1	*or*	$\neg x_3$	*or*	x_4	=	*true*
c_3	x_2	*or*	$\neg x_3$	*or*	$\neg x_1$	=	*true*

one vertex per variable per equation

one edge between variables in different equations as long as they aren't $x, \neg x$

Clique instance



m = # of equations

Why? 3-SAT instance is satisfiable, if and only if exists clique of size m ← *(optional) short proof is based on analyzing YES instances*

Establishing intractability. If 3-SAT is intractable (as conjectured), then clique finding is also intractable.

$p(n) = n^2$

If $T(n)$ algorithm for clique finding, then there is a $T(n^2)$ algorithm for 3-SAT (since #edges is quadratic in #equations)

Intractability: quiz 2

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

Which of the following can we infer?

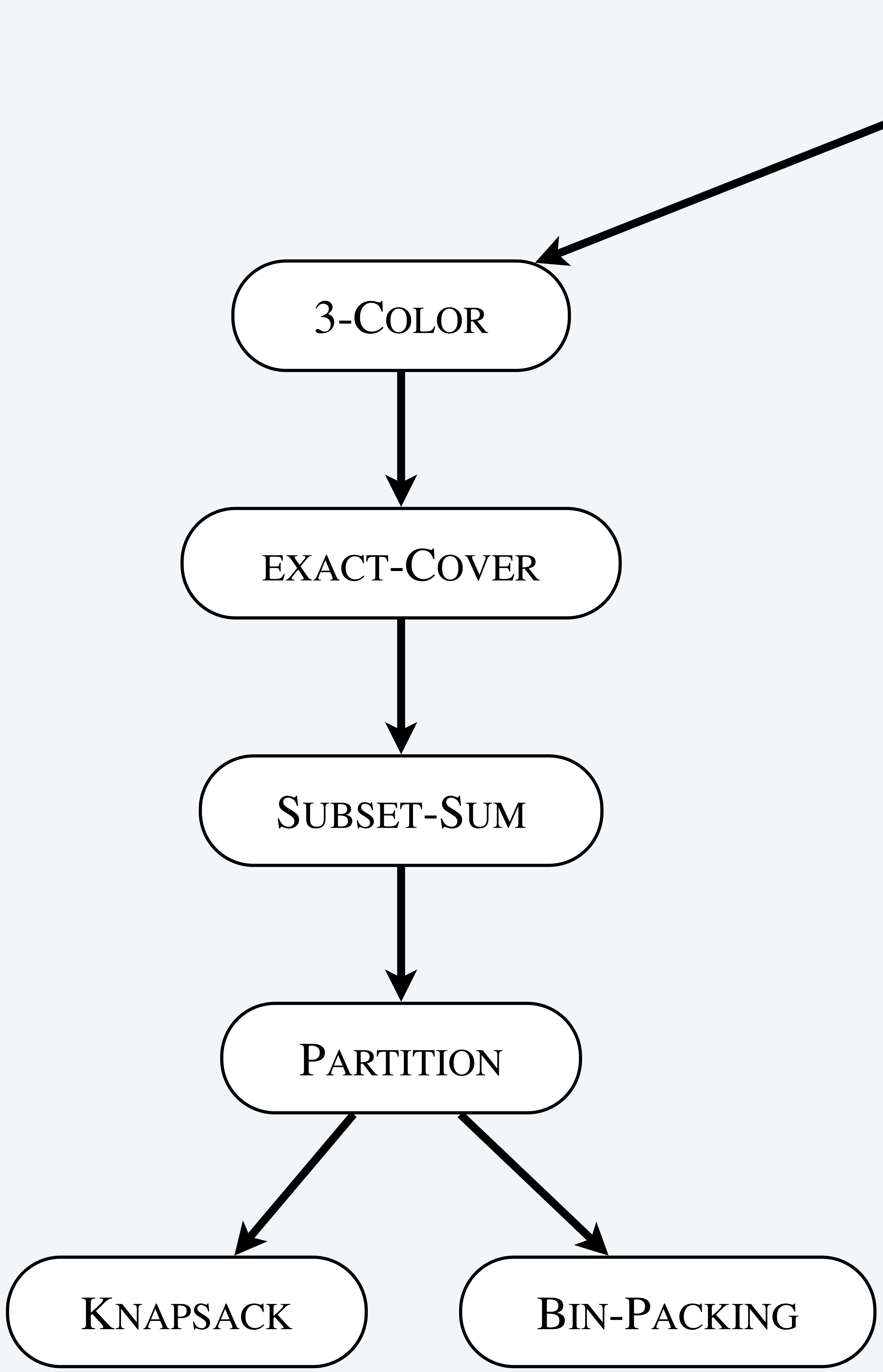
A. If Y can be solved in $\Theta(n^3)$ time, then X can be solved in $\Theta(n^3)$ time.

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

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

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

Some poly-time reductions from SAT



SAT poly-time reduces VERTEX-COVER.

how can an algorithm solving a graph problem be used to solve a system of equations?

intricate reductions are common!

Richard Karp
(1972)

Conjecture. 3-SAT is intractable.

If true, all these problems are also intractable!

https://algs4.cs.princeton.edu

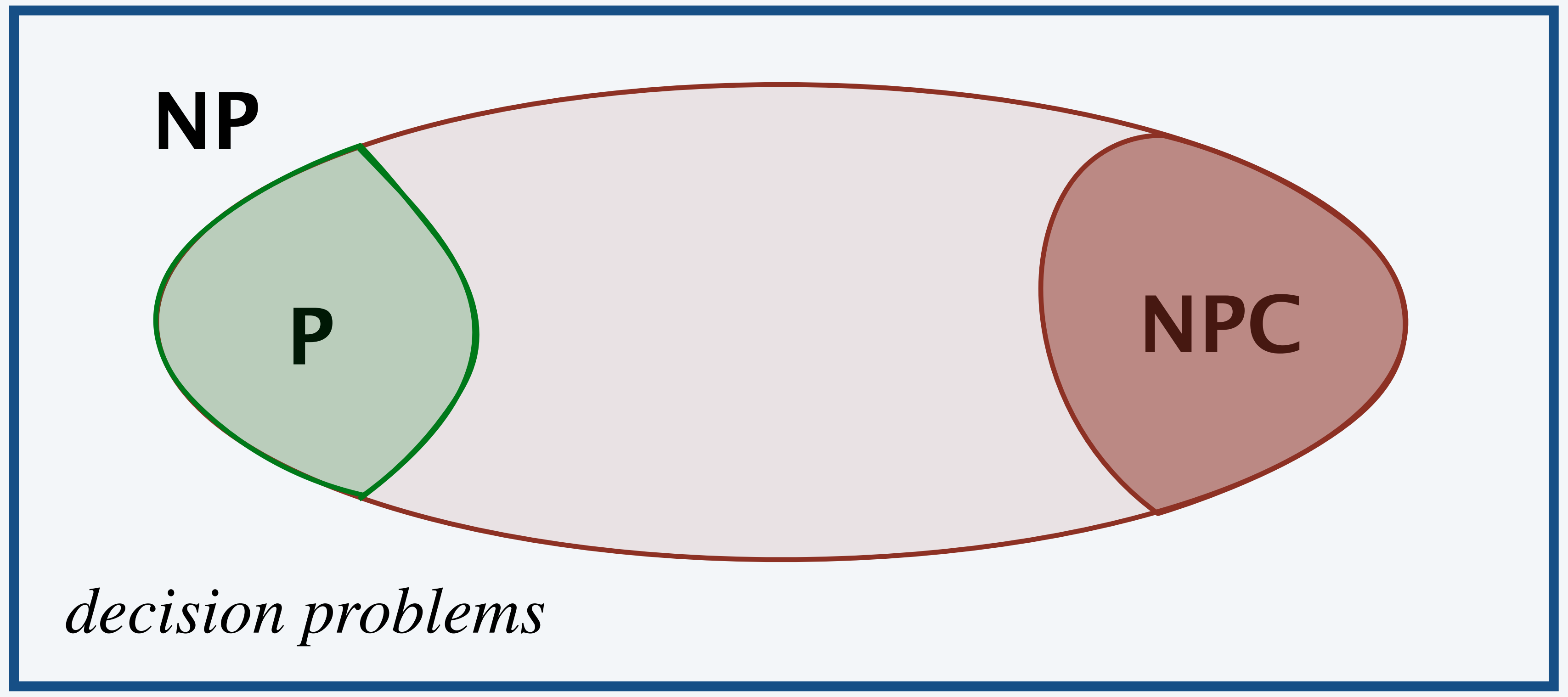
Intractability

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

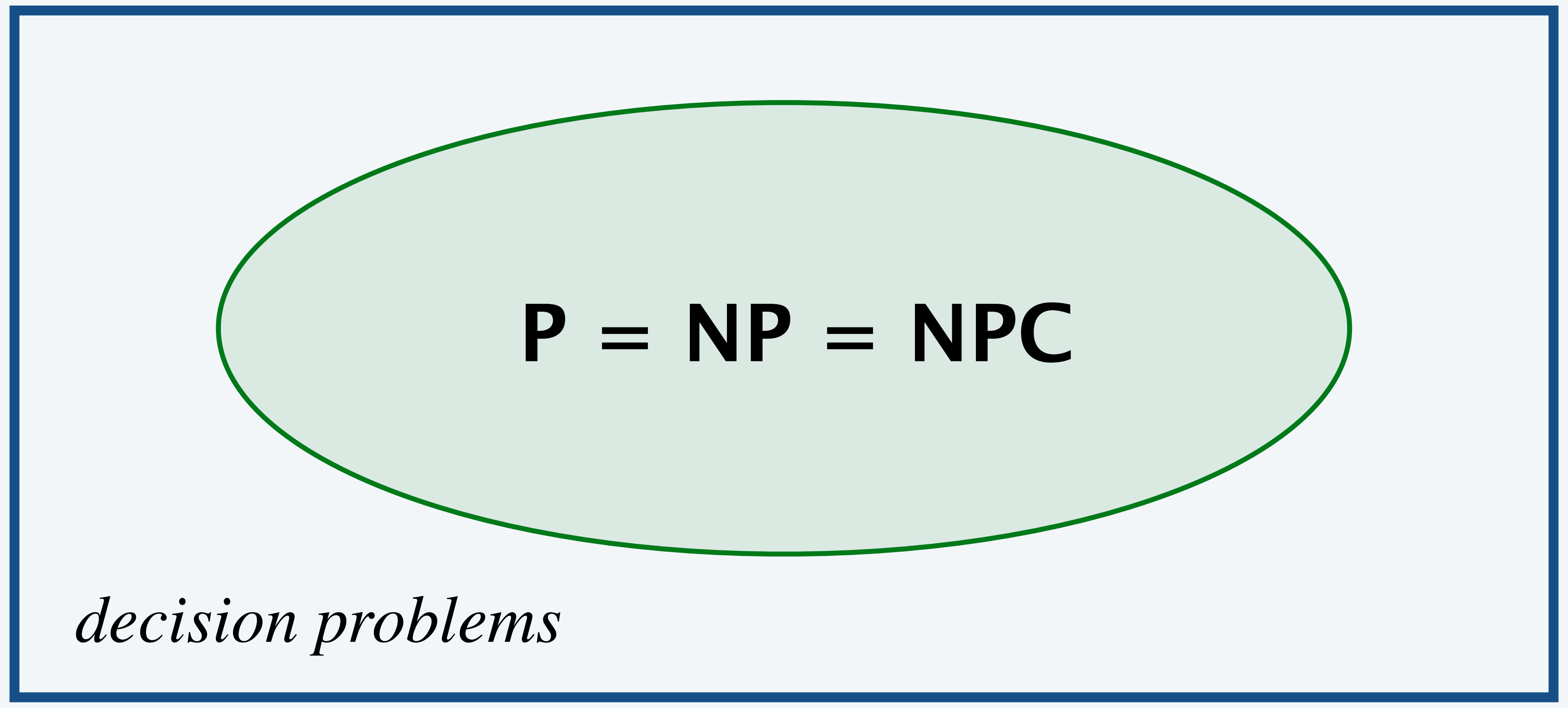
NP-completeness

Def. $Y \in$ **NP** is **NP-complete** if for all $X \in$ **NP**, $X \leq Y$. ⟵ *X is maximally hard in* **NP**

Two worlds.



P ≠ NP



P = NP

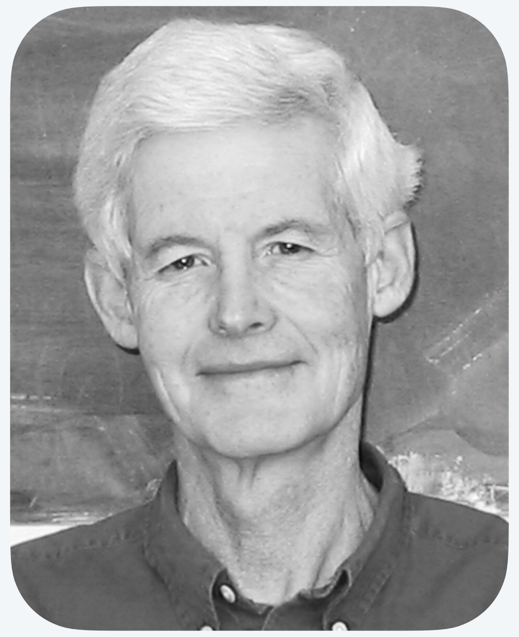
Cook-Levin theorem. 3-SAT is **NP**-complete. ⟵ *how can we prove $X \leq$ 3-SAT if we don't know X?*

Pioneering result in computer science!

Corollary 1. 3-SAT can be solved in poly-time if and only if **P** = **NP**.

Corollary 2. To show that $Y \in$ **NP** is **NP**-complete, it suffices to show 3-SAT $\leq Y$.

Thousands of problems have been proven to be **NP**-complete!



Stephen Cook
(1971)

Leonid Levin
(1971)

NP-complete problems

field of study	NP-complete problem
Computer sience / Math	*maxcut, longest path, vertex cover, 3-SAT,…*
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.*

NP-complete problems are different manifestations of the *same* fundamentally hard problem.

Solving any one of them in poly time solves all!

No field-specific math insights are required!

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

A. $X \in$ **NP**.

B. If X can be solved in poly-time, then **P** $=$ **NP**.

C. If X cannot be solved in poly-time, then **P** $\neq$ **NP**.

D. If $Y \in$ **NP** and $X \preceq Y$ then Y is **NP**-complete.

E. All of the above.

https://algs4.cs.princeton.edu

Intractability

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

Dealing with intractability

Approaches to coping with intractability

… so your problem is **NP**-complete 😭

Safe to assume it is intractable: no worst-case poly-time algorithm <u>solves</u> <u>all</u> problem instances.

Do you need to solve *all* instances?

Model real-world instances. Worst-case inputs might not arise in practical applications.

protein folding is **NP**-*complete* ↓

Do you need the exact *optimal* solution?

Approximation algorithms. Look for good (though potentially suboptimal) solutions.

Approximating 3-SAT

(Recall) 3-SAT. Given m boolean equations over the variables $x_1 \ldots x_n$ in the form "y_i *or* y_j *or* y_k = *true*", where y_i is either x_i or $\neg x_i$, return a truth assignment that satisfies all equations, or report that there is none.

$\neg x_1$	*or*	x_2	*or*	x_3	=	*true*
x_1	*or*	$\neg x_3$	*or*	x_4	=	*true*
x_2	*or*	$\neg x_3$	*or*	$\neg x_1$	=	*true*
$\neg x_2$	*or*	x_4	*or*	x_3	=	*true*
$\neg x_3$	*or*	$\neg x_4$	*or*	$\neg x_2$	=	*true*

3-SAT instance I

x_1	=	*false*
x_2	=	*false*
x_3	=	*true*
x_4	=	*true*

satisfying assignment

x_1	=	*false*
x_2	=	*false*
x_3	=	*true*
x_4	=	*false*

assignment that satisfies 4/5 fraction

Def. $OPT(I)$ is the maximum fraction of equations in I that can be satisfied.

often $k = 1$

3-SAT (decision): Given I, k, is $OPT(I) \geq k$? ⟵ **NP**-complete 😭

3-SAT (α-approx): Given I, return an assignment that satisfies $\geq \alpha \cdot OPT(I)$ fraction of the equations.

want $\alpha < 1$ as large as possible

Approximation algorithm

α-approximation algorithm:

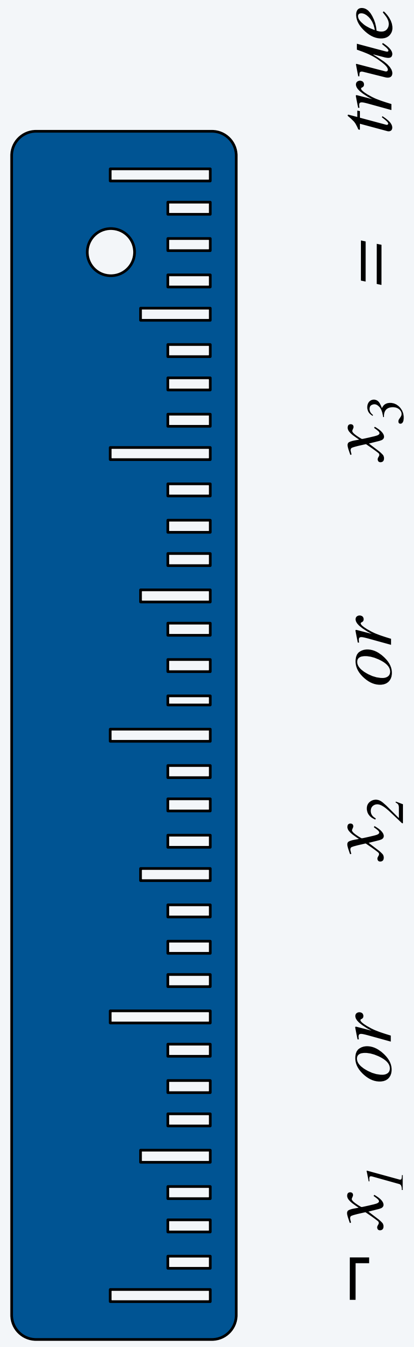
For a *minimization* problems: return a solution with value $\leq \alpha \cdot OPT$, $\alpha > 1$.

For a *maximization* problems: return a solution with value $\geq \alpha \cdot OPT$, $\alpha < 1$.

An **NP**-complete problem may admit a *polynomial-time* α-approximation algorithm:

- For no constant α. ⟵ *hard to solve with any precision*
- For some constant α (e.g., 2, 1/2 or 7/8). ⟵ *easy to solve with precision α, hard with better precision*
- For every $\alpha \neq 0, 1$ (PTAS/FPTAS). ⟵ *easy to solve with any precision, hard to solve exactly*

The field of hardness of approximation studies the optimal α achievable for different **NP**-complete problems.



3-SAT: randomized 7/8-approximation algorithm

Observation. A random assignment satisfies an equation with probability $7/8$.

E.g., "$\neg x_5$ *or* x_8 *or* $\neg x_9 = true$" is not satisfied only when $x_5 = T, x_8 = F, x_9 = T$, which happens with probability $(1/2)^3$.

m = # of equations

Property. For any I, the probability that a random assignment satisfies at least $\frac{7m}{8}$ equations is at least $\frac{1}{8m}$

This is intuitive given the observation and can be shown using some simple probability facts

(optional) linearity of expectation + Markov's inequality

Algorithm.

Generate $100m$ random assignments and return the one that satisfies the most equations. ← *randomized polynomial time*

optimal! (unless **P = NP***)*

Claim. For any I, with probability .99, the returned assignment satisfies $\geq 7/8$ fraction of the equations.

Proof. We can use the above property plus the error reduction property

recall: If $\mathbb{P}[A \text{ fails}] = p$ and want failure $\leq q$, repeat $k \geq \log_p q$ times.
Then, $\mathbb{P}[A \text{ fails } k \text{ times}] = p^k \leq q$.

https://algs4.cs.princeton.edu

Intractability

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

Leveraging intractability: guiding scientific inquiry

1926. Ising introduces a mathematical model for ferromagnetism.

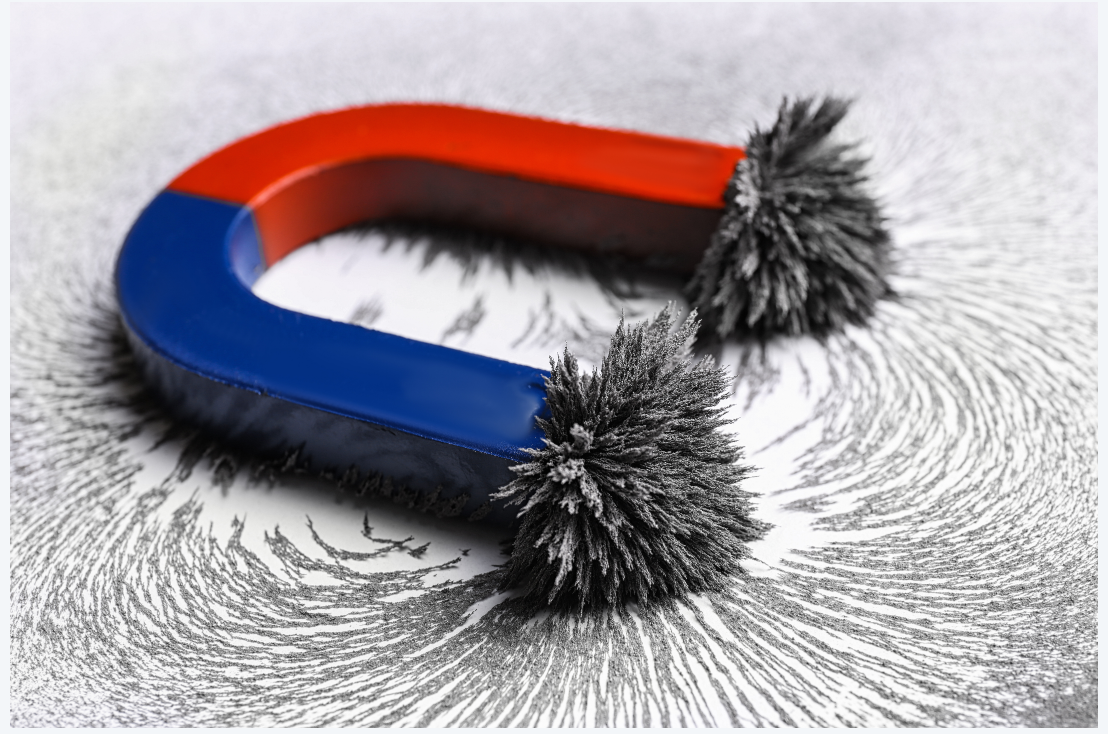
1930s. Closed form solution is a holy grail of statistical mechanics.

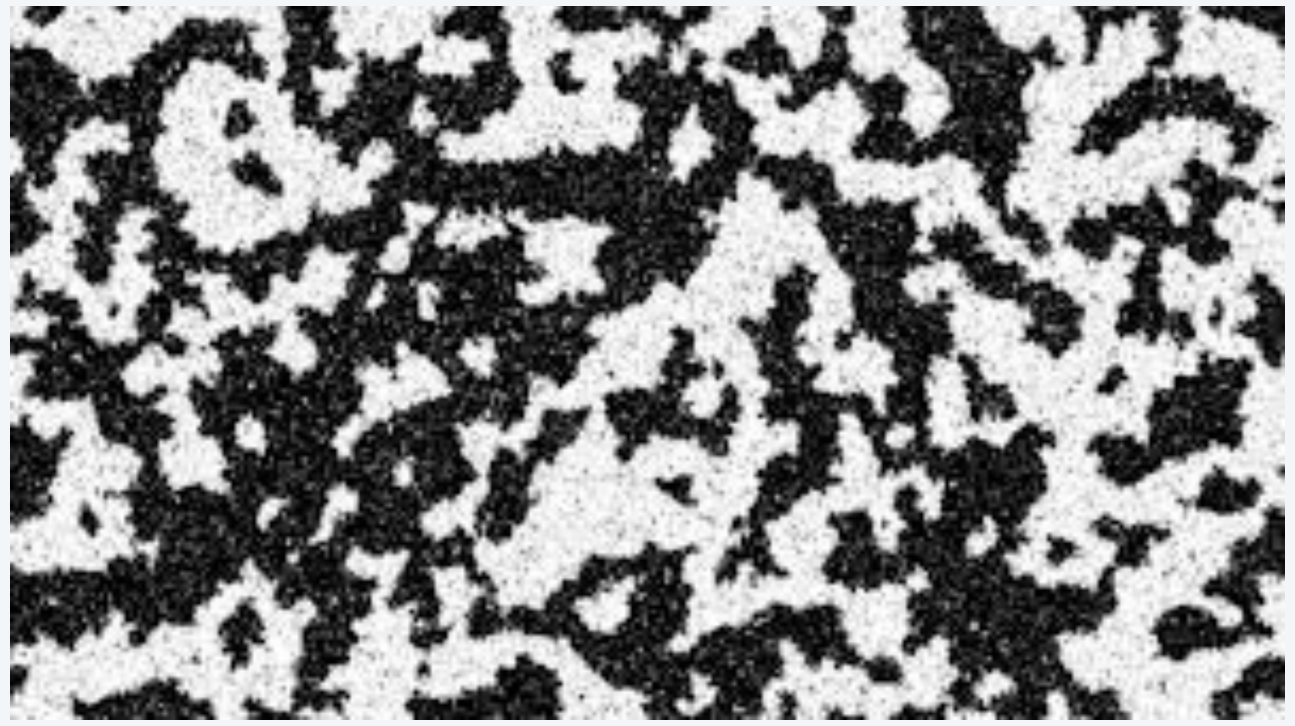
1944. Onsager finds closed form solution to 2D version in tour de force.

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.





Leveraging intractability: cryptography

Secure password system. A user creates a password to enable login to their account.

How can the server store the password securely?

Solution. Convert password into two large primes p, q. Server stores only the product $N = pq$.

To log in, user provides p, q. The server computes the product and compares to N.

Server: Multiply two integers (efficient).

Malicious user: Solve factorization (conjectured to be intractable).

Cryptographic schemes (e.g., RSA encryption) require malicious parties to solve intractable (?) **NP** problems.

P = **NP** $\Longrightarrow$ *no crypto!*

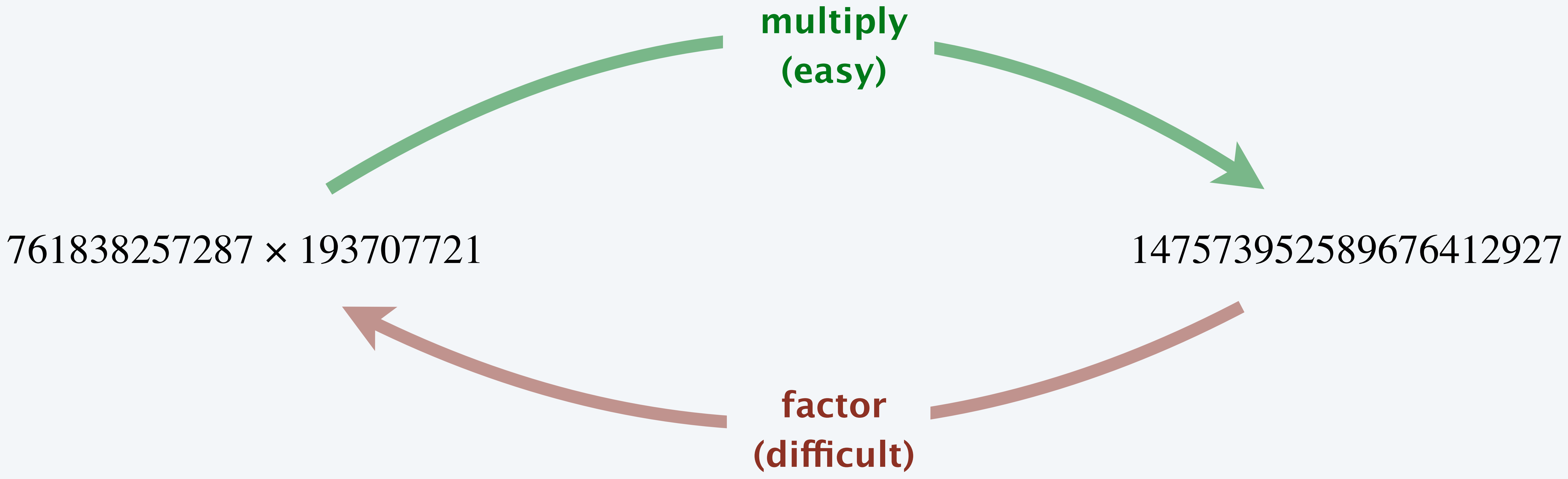

Ron Rivest


Adi Shamir


Len Adelman

MasterCard SecureCode

Verified by VISA



Leveraging intractability: derandomization

Fun game. I toss a coin; you guess how it will land. What's the probability you guess correctly? *50%*

Fun game 2. I toss a coin; you can use your computer to guess how it will land.
What's the probability you guess correctly? *still 50%…*

Fun game 3. I toss a coin; you are a Martian with complete knowledge of the physics of the universe and access to sophisticated equipment.
You guess how it will land—what's the probability you guess correctly? *100%?*

Randomness is in the ~~eye~~ of the beholder!
computational power

Hardness vs. Randomness. The outcome of intractable problems often appears random.
We can feed such outcomes to randomized algorithms instead of real randomness,
thereby making them deterministic.

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