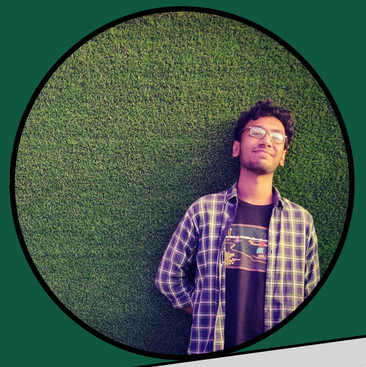


Mr. Alvi Haque

NP Completeness

WILLIAM & MARY

CHARTERED 1693

Spring 2026

P vs NP, SAT and NP-Completeness

Is P = NP?

The biggest open question in computer science — for over 55 years.

THE PRIZE

$1,000,000

FIRST POSED

1971

Stephen Cook

Turing Award 1982

STATUS

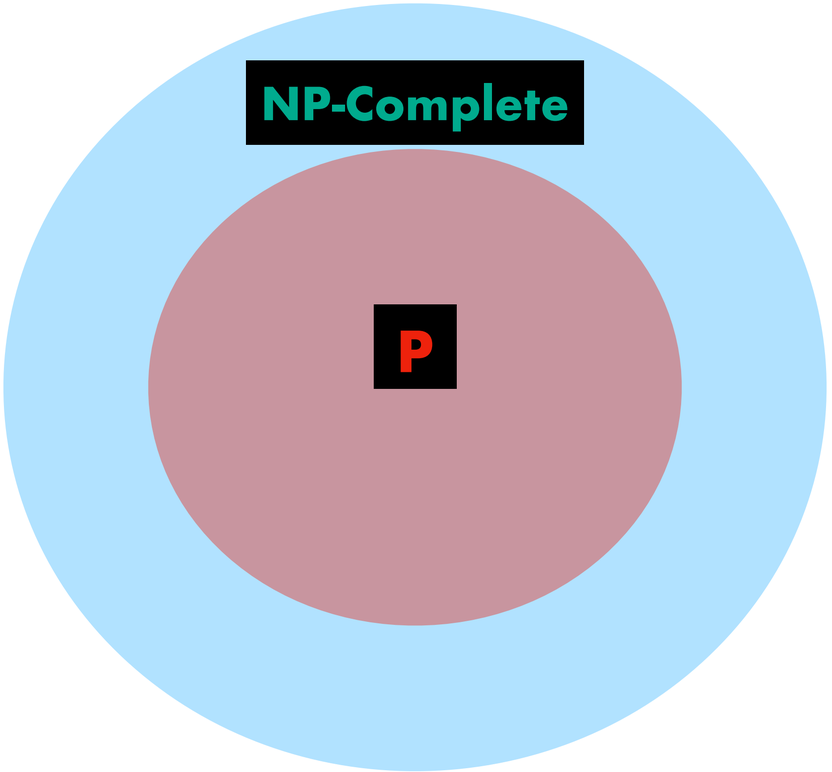
Unsolved

Most believe P ≠ NP

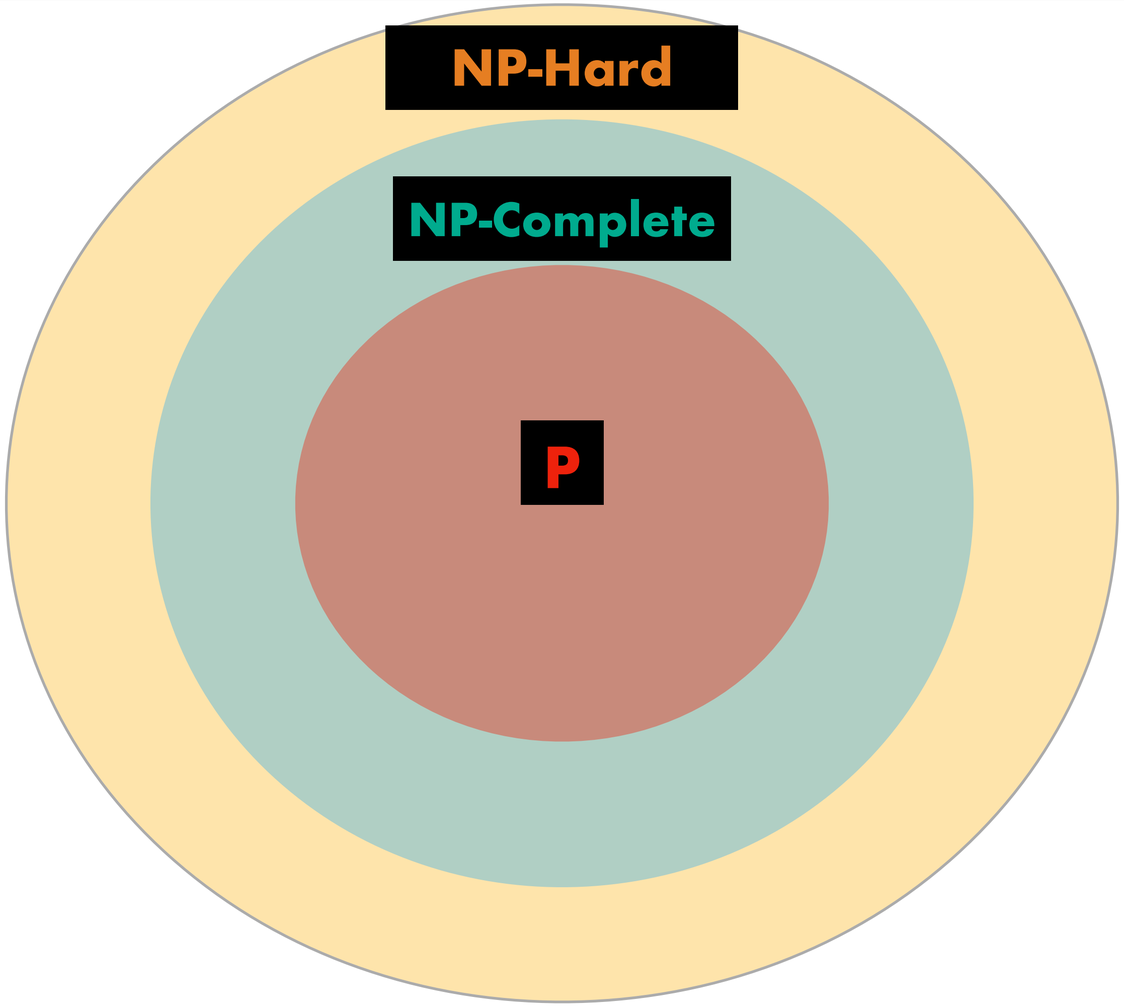
but no one has proven it

Source: claymath.org/millennium/p-vs-np/

P vs NP, SAT and NP-Completeness

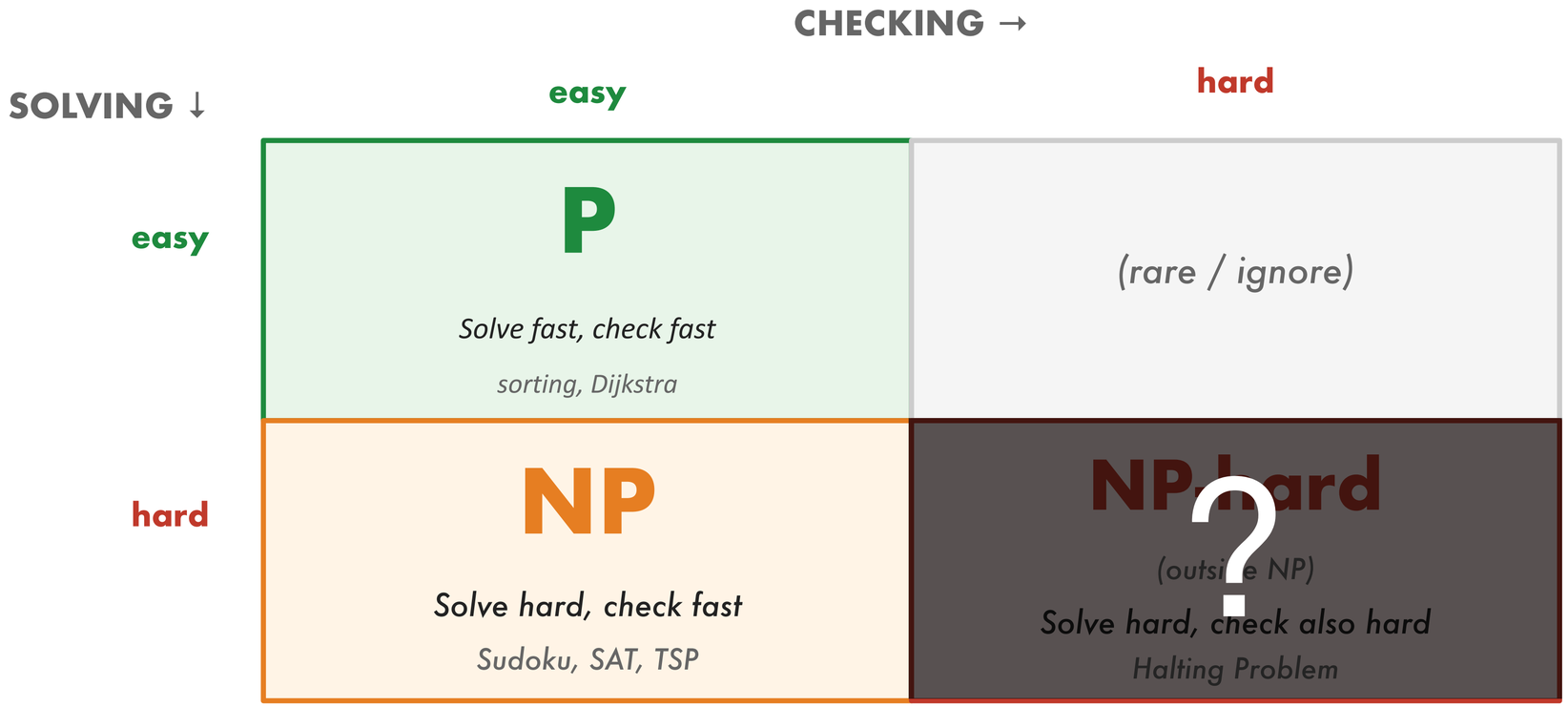


P vs NP, SAT and NP-Completeness



P vs NP, SAT and NP-Completeness

Every problem lives on TWO axes — solving and checking. They're independent.



NP-complete = hardest problems in NP (hard to solve, but still easy to check)

Every Problem is Secretly Yes/No

Problem	Underlying yes/no decision
Knapsack	*Should item X go in the bag?*
Graph coloring	*Should node A be red?*
Routing	*Should edge X be in the path?*
Scheduling	*Should exam X be on Monday?*

SAT = the raw form

Variables:
x_1, x_2, x_3 ... (each TRUE or FALSE)

Operators:
AND OR NOT

Question:
can we make the whole formula TRUE?

SAT is the skeleton of decision problems.
Remove knapsacks, graphs, cities — what's left is: "assign TRUE/FALSE to satisfy the rules."

CNF — A Required Writing Format

Like an essay template — CNF is a standard shape for writing SAT formulas.

The CNF rule:

Inside each ( ) → only OR
Between each ( ) → only AND

AND = ∧ (both must be true)
OR = ∨ (at least one must be true)
NOT = ¬ (flip the value)

Each ( ) is called a clause.

Example CNF formula

$$(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2)$$

clause 1: x_1 OR x_2 OR (NOT x_3)
clause 2: (NOT x_1) OR x_2

Joined by AND → all clauses must be true

Why NOT matters: without it, you could only say "this must be true."
NOT lets you express **forbidden combinations**. Example: "A and B can't both be red" → ¬(A_red ∧ B_red).

Reduction = Translation

To prove a NEW problem is hard, don't start from scratch. Translate SAT into it.

The logic

1. SAT is known to be hard (Cook 1971).
2. Translate SAT into your new problem.
3. Suppose your problem had a fast solver.
4. Then SAT could be solved fast too.
5. But SAT can't be. → Contradiction.
6. So your problem is also hard.

Translator analogy

You speak English.
Friend speaks Japanese.
Book is in Japanese.

Question: "Is this book hard?"

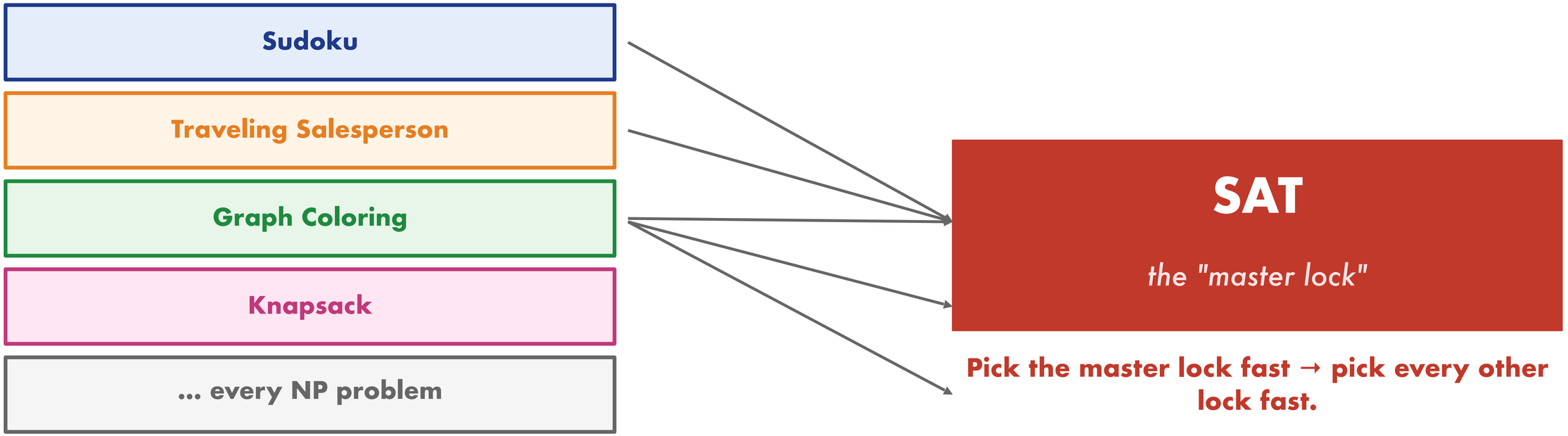
You translate Japanese → English.
If English version is hard →
original must also be hard.

Difficulty transfers through translation.

One reduction = lifetime proof of hardness.

SAT is the Master Lock

Cook (1971): every NP problem can be rewritten as a SAT formula.



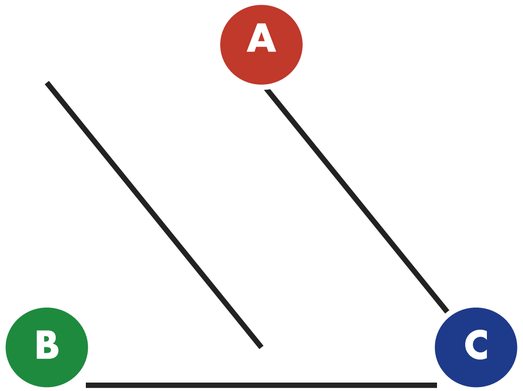
One fast algorithm for SAT = one fast algorithm for every NP problem.

That's why SAT is called NP-complete — solve it, solve them all.

Concrete Reduction: Graph Coloring → SAT

Goal: color 3 nodes with 3 colors so connected nodes don't share a color.

The graph



3 nodes, 3 edges, 3 colors

translate to →

The SAT formula

9 variables (one per node+color):
A_red, A_green, A_blue ...

Each node must get a color:
(A_red ∨ A_green ∨ A_blue) ∧ ...

Connected nodes can't share color:
¬(A_red ∧ B_red) ∧ ...

Solve SAT → read off the colors.

Fast SAT solver → fast graph coloring solver.

Since no fast SAT solver exists → no fast coloring solver either. Graph Coloring is NP-complete.

So Why Do We Care?

Proving NP-completeness is a permission slip to stop chasing perfection.

1. **You discover a hard-looking problem.**

   Your boss says "solve this perfectly."

2. **You prove it's NP-complete via reduction.**

   Now there's mathematical evidence: no fast algorithm exists (probably).

3. **You stop chasing perfection.**

   Free to use heuristics, approximations, randomness, learning.

4. **You ship something useful.**

   Imperfect but fast. Real-world problem solved.

This is why heuristics dominate modern AI.

Why Your LLM Exists

Every "good enough" trick in modern AI = a response to a hard problem.

Best summary *$32{,}000^{128}$ possibilities*	→ **Greedy / beam search** *argmax at each step*
Best neural network weights *training is NP-hard*	→ **Gradient descent (Adam, SGD)** *follow the slope locally*
Best code from a spec *code generation is undecidable*	→ **LLMs themselves** *predict plausible code*

All of deep learning is one giant set of heuristics for problems classical CS gave up on.

Three Things to Take Away

1

Every NP problem = yes/no decisions. SAT is their skeleton.

CNF is the required format. Cook (1971) proved every NP problem reduces to SAT.

2

Reduction = translation. Hardness transfers.

Translate SAT into your problem → your problem inherits SAT's hardness. That's how NP-completeness is proven.

3

Heuristics aren't lazy — they're necessary.

Greedy decoding, SGD, beam search, sampling — the only way forward when the math says "perfect is impossible."

A question to leave with: What are you still trying to solve perfectly — when you should be using a heuristic?