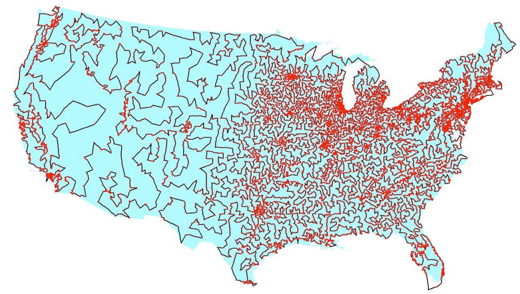
# Intractability

## A difficult problem

Traveling salesperson problem (TSP)

Given: A set of N cities and $M for gas.
Problem: Does a traveling salesperson have enough $ for gas to visit all the cities?



An algorithm ("exhaustive search"):
Try all N! orderings of the cities to find one that can be visited for $M

## A Reasonable Question about Algorithms

Q. Which algorithms are useful in practice?

A. [von Neumann 1953, Gödel 1956, Cobham 1964, Edmonds 1965, Rabin 1966]

- Model of computation = deterministic Turing machine.
- Measure running time as a function of input size N.
- Polynomial time: Number of steps less than $aN^b$ for some constants a, b.
- Useful in practice ("efficient") = polynomial time for all inputs.

Ex 1. Sorting N elements
Insertion sort takes less than $aN^2$ steps for all inputs.
efficient

Ex 2. TSP on N cities
Exhaustive search could take $aN!$ steps.
not efficient

In theory: Definition is broad and robust (since a and b tend to be small).
In practice: Poly-time algorithms tend to scale to handle large problems.

## Exponential Growth

Exponential growth dwarfs technological change.

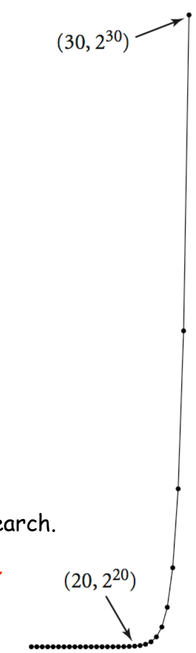
- Suppose you have a giant parallel computing device…
- With as many processors as electrons in the universe…
- And each processor has power of today's supercomputers…
- And each processor works for the life of the universe…

| quantity | value |
|---|---|
| electrons in universe † | $10^{79}$ |
| supercomputer instructions per second | $10^{13}$ |
| age of universe in seconds † | $10^{17}$ |

† estimated

- Will not help solve 1,000 city TSP problem via exhaustive search.

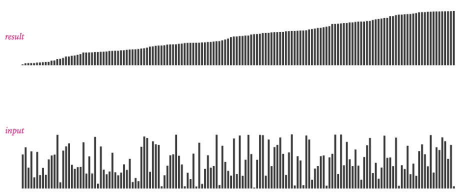
$1000! \gg 10^{1000} \gg 10^{79} \times 10^{13} \times 10^{17}$



## Reasonable Questions about Problems

Q. Which problems can we solve in practice?
A. Those with easy-to-find answers or with guaranteed poly-time algorithms.

Q. Which problems have guaranteed poly-time algorithms?
A. Not so easy to know. Focus of today's lecture.




many known poly-time algorithms for sorting

no known poly-time algorithm for TSP

## Four Fundamental Problems

LSOLVE. Given a system of linear equations, find a solution.

$$\begin{aligned} 0x_0 + 1x_1 + 1x_2 &= 4 \\ 2x_0 + 4x_1 - 2x_2 &= 2 \\ 0x_0 + 3x_1 + 15x_2 &= 36 \end{aligned} \qquad \begin{aligned} x_0 &= -1 \\ x_1 &= 2 \\ x_2 &= 2 \end{aligned}$$

← variables are real numbers

LP. Given a system of linear inequalities, find a solution.

$$\begin{aligned} 48x_0 + 16x_1 + 119x_2 &\le 88 \\ 5x_0 + 4x_1 + 35x_2 &\ge 13 \\ 15x_0 + 4x_1 + 20x_2 &\ge 23 \\ x_0 \;,\; x_1 \;,\; x_2 &\ge 0 \end{aligned} \qquad \begin{aligned} x_0 &= 1 \\ x_1 &= 1 \\ x_2 &= 1/5 \end{aligned}$$

← variables are real numbers

ILP. Given a system of linear inequalities, find a 0-1 solution.

$$\begin{aligned} x_1 + x_2 &\ge 1 \\ x_0 + x_2 &\ge 1 \\ x_0 + x_1 + x_2 &\le 2 \end{aligned} \qquad \begin{aligned} x_0 &= 0 \\ x_1 &= 1 \\ x_2 &= 1 \end{aligned}$$

← variables are 0 or 1

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

$$\begin{aligned} (x_0 \text{ and } x_1 \text{ and } x_2) \text{ or } (x_1 \text{ and } x_2) \text{ or } (x_0 \text{ and } x_2) &= \text{true} \\ (x_0 \text{ and } x_1) \text{ or } (x_1 \text{ and } x_2) &= \text{false} \\ (x_1 \text{ and } x_2) \text{ or } (x_0 \text{ and } x_2) \text{ or } (x_0) &= \text{true} \end{aligned} \qquad \begin{aligned} x_0 &= \text{false} \\ x_1 &= \text{true} \\ x_2 &= \text{true} \end{aligned}$$

← variables are "true" or "false"

## Four Fundamental Problems

LSOLVE. Given a system of linear equations, find a solution.
LP. Given a system of linear inequalities, find a solution.
ILP. Given a system of linear inequalities, find a binary solution.
SAT. Given a system of boolean equations, find a solution.

Q. Which of these problems have guaranteed poly-time solutions?
A. No easy answers.

✓ LSOLVE. Yes. Gaussian elimination solves $n$-by-$n$ system in $n^3$ time.
✓ LP. Yes. Ellipsoid algorithm is poly-time. ← problem was open for decades
? ILP, SAT. No poly-time algorithm known or believed to exist!

## Search Problems

or report none exists

Search problem. Given an instance $I$ of a problem, find a solution $S$.
Requirement. Must be able to efficiently check that $S$ is a solution.

poly-time in size of instance $I$

## Search Problems

or report none exists

Search problem. Given an instance $I$ of a problem, find a solution $S$.
Requirement. Must be able to efficiently check that $S$ is a solution.

poly-time in size of instance $I$

LSOLVE. Given a system of linear equations, find a solution.

$$\begin{array}{rrrcr} 0x_0 & +\ 1x_1 & +\ 1x_2 & = & 4 \\ 2x_0 & +\ 4x_1 & -\ 2x_2 & = & 2 \\ 0x_0 & +\ 3x_1 & +\ 15x_2 & = & 36 \end{array} \qquad \begin{array}{rcr} x_0 & = & -1 \\ x_1 & = & 2 \\ x_2 & = & 2 \end{array}$$

instance $I$ — solution $S$

- To check solution $S$, plug in values and verify each equation.

## Search Problems

or report none exists

Search problem. Given an instance $I$ of a problem, find a solution $S$.
Requirement. Must be able to efficiently check that $S$ is a solution.

poly-time in size of instance $I$

LP. Given a system of linear inequalities, find a solution.

$$\begin{array}{rrrcr} 48x_0 & +\ 16x_1 & +\ 119x_2 & \le & 88 \\ 5x_0 & +\ 4x_1 & +\ 35x_2 & \ge & 13 \\ 15x_0 & +\ 4x_1 & +\ 20x_2 & \ge & 23 \\ x_0 & ,\ x_1 & ,\ x_2 & \ge & 0 \end{array} \qquad \begin{array}{rcr} x_0 & = & 1 \\ x_1 & = & 1 \\ x_2 & = & 1/5 \end{array}$$

instance $I$ — solution $S$

- To check solution $S$, plug in values and verify each inequality.

## Search Problems

or report none exists

Search problem. Given an instance $I$ of a problem, find a solution $S$.
Requirement. Must be able to efficiently check that $S$ is a solution.

poly-time in size of instance $I$

ILP. Given a system of linear inequalities, find a binary solution.

$$\begin{array}{rrrcr} & x_1 & +\ x_2 & \ge & 1 \\ x_0 & & +\ x_2 & \ge & 1 \\ x_0 & +\ x_1 & +\ x_2 & \le & 2 \end{array} \qquad \begin{array}{rcr} x_0 & = & 0 \\ x_1 & = & 1 \\ x_2 & = & 1 \end{array}$$

instance $I$ — solution $S$

- To check solution $S$, check that values are 0/1 , then plug in values and verify each inequality.

## Search Problems

or report none exists

Search problem. Given an instance $I$ of a problem, find a solution $S$.
Requirement. Must be able to efficiently check that $S$ is a solution.

poly-time in size of instance $I$

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

$$\begin{array}{llll} (x_0 \text{ and } x_1 \text{ and } x_2) \text{ or } (x_1 \text{ and } x_2) \text{ or } (x_0 \text{ and } x_2) & = \text{true} \\ (x_0 \text{ and } x_1) \text{ or } (x_1 \text{ and } x_2) & = \text{false} \\ (x_1 \text{ and } x_2) \text{ or } (x_0 \text{ and } x_2) \text{ or } (x_0) & = \text{true} \end{array} \qquad \begin{array}{l} x_0 = \text{false} \\ x_1 = \text{true} \\ x_2 = \text{true} \end{array}$$

instance $I$ — solution $S$

- To check solution $S$, plug in values and verify each equation.

## Search Problems

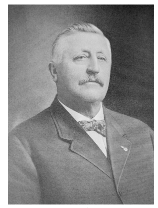
or report none exists

Search problem. Given an instance $I$ of a problem, find a solution $S$.
Requirement. Must be able to efficiently check that $S$ is a solution.

poly-time in size of instance $I$

FACTOR. Find a nontrivial factor of the integer $x$.

Mersenne



Cole

$2^{67} - 1$

`147573952589676412927` — instance $I$

`193707721` — solution $S$

- To check solution $S$, long divide `193707721` into `147573952589676412927`.

## NP

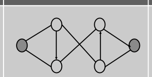
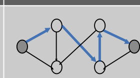
Def. NP is the class of all search problems ← problems with poly-time checkable solutions

| problem | description | poly-time algorithm | instance $I$ | solution $S$ |
|---|---|---|---|---|
| LSOLVE $(A, b)$ | Find a vector $x$ that satisfies $Ax = b$. | Gaussian elimination | $\begin{aligned} 0x_0 + 1x_1 + 1x_2 &= 4 \\ 2x_0 + 4x_1 - 2x_2 &= 2 \\ 0x_0 + 3x_1 + 15x_2 &= 36 \end{aligned}$ | $x_0 = -1$, $x_1 = 2$, $x_2 = 2$ |
| LP $(A, b)$ | Find a vector $x$ that satisfies $Ax \le b$. | ellipsoid | $\begin{aligned} 48x_0 + 16x_1 + 119x_2 &\le 88 \\ 5x_0 + 4x_1 + 35x_2 &\ge 13 \\ 15x_0 + 4x_1 + 20x_2 &\ge 23 \\ x_0 , x_1 , x_2 &\ge 0 \end{aligned}$ | $x_0 = 1$, $x_1 = 1$, $x_2 = 1/5$ |
| ILP $(A, b)$ | Find a binary vector $x$ that satisfies $Ax \le b$. | ??? | $\begin{aligned} x_1 + x_2 &\ge 1 \\ x_0 + x_2 &\ge 1 \\ x_0 + x_1 + x_2 &\le 2 \end{aligned}$ | $x_0 = 0$, $x_1 = 1$, $x_2 = 1$ |
| SAT $(A, b)$ | Find a boolean vector $x$ that satisfies $Ax = b$. | ??? | $(x_1 \text{ and } x_2) \text{ or } (x_0 \text{ and } x_2) = \text{true}$; $(x_0 \text{ and } x_1) \text{ or } (x_1 \text{ and } x_2) = \text{false}$; $(x_0 \text{ and } x_2) \text{ or } (x_0) = \text{true}$ | $x_0 = \text{false}$, $x_1 = \text{true}$, $x_2 = \text{true}$ |
| FACTOR $(x)$ | Find a nontrivial factor of the integer $x$. | ??? | 8784561 | 10657 |

Significance. What scientists, engineers, and applications programmers aspire to compute feasibly.

## P

Def. P is the class of search problems solvable in poly-time.
A search problem that is not in P is said to be intractable.

| problem | description | poly-time algorithm | instance $I$ | solution $S$ |
|---|---|---|---|---|
| STCONN $(G, s, t)$ | Find a path from $s$ to $t$ in digraph $G$. | depth-first search (Theseus) |  |  |
| SORT $(a)$ | Find permutation that puts a in ascending order. | mergesort (von Neumann 1945) | 2.3 8.5 1.2 9.1 2.2 0.3 | 5 2 4 0 1 3 |
| LSOLVE $(A, b)$ | Find a vector $x$ that satisfies $Ax = b$. | Gaussian elimination (Edmonds, 1967) | $\begin{aligned} 0x_0 + 1x_1 + 1x_2 &= 4 \\ 2x_0 + 4x_1 - 2x_2 &= 2 \\ 0x_0 + 3x_1 + 15x_2 &= 36 \end{aligned}$ | $x_0 = -1$, $x_1 = 2$, $x_2 = 2$ |
| LP $(A, b)$ | Find a vector $x$ that satisfies $Ax \le b$. | ellipsoid (Khachiyan, 1979) | $\begin{aligned} 48x_0 + 16x_1 + 119x_2 &\le 88 \\ 5x_0 + 4x_1 + 35x_2 &\ge 13 \\ 15x_0 + 4x_1 + 20x_2 &\ge 23 \\ x_0 , x_1 , x_2 &\ge 0 \end{aligned}$ | $x_0 = 1$, $x_1 = 1$, $x_2 = 1/5$ |

Significance. What scientists and engineers, and applications programmers do compute feasibly.

## Other types of problems

Search problem. Find a solution.

Decision problem. Is there a solution?

Optimization problem. Find the best solution.

Some problems are more naturally formulated in one regime than another.
Ex. TSP is usually "find the shortest tour that connects all the cities."

Not technically equivalent, but main conclusions that we draw apply to all 3.

Note: Standard definitions of P and NP are in terms of decision problems.

## Nondeterminism

Nondeterministic machine can guess the desired solution

Ex. `int[] a = new a[N];`
- Java: values are all 0
- nondeterministic machine: values are the answer!

ILP. Given a system of linear inequalities, guess a 0/1 solution.

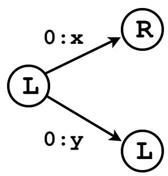
$$\begin{aligned} x_1 + x_2 &\geq 1 \\ x_0 + x_2 &\geq 1 \\ x_0 + x_1 + x_2 &\leq 2 \end{aligned}$$

instance $I$

$$\begin{aligned} x_0 &= 0 \\ x_1 &= 1 \\ x_2 &= 1 \end{aligned}$$

solution $S$

Ex. Turing machine
- deterministic: state, input determines next state
- nondeterministic: more than one possible next state




NP: Search problems solvable in Poly time on a Nondeterministic machine.

all of them!

## Extended Church-Turing Thesis

Extended Church-Turing thesis.

P = search problems solvable in poly-time in this universe.

Evidence supporting thesis.
- True for all physical computers.
- Simulating one computer on another adds poly-time cost factor.
- Nondeterministic machine seems to be a fantasy.

Implication. To make future computers more efficient, suffices to focus on improving implementation of existing designs.

A new law of physics? A constraint on what is possible.
Possible counterexample? Quantum computer

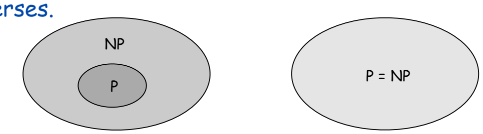
# P vs. NP

## The Central Question

P. Class of search problems solvable in poly-time.
NP. Class of all search problems.

Does P = NP?
- can you always avoid brute-force search and do better??
- does nondeterminism make a computer more efficient??
- are there any intractable search problems??

Two possible universes.

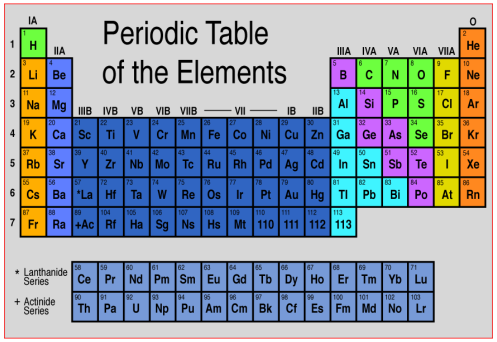



P ≠ NP  P = NP

If yes... Poly-time algorithms for 3-SAT, ILP, TSP, FACTOR, ...
If no... Would learn something fundamental about our universe.

Overwhelming consensus. P ≠ NP.

# Classifying Problems




## Exhaustive Search

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

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

SAT is intractable


## Classifying Problems

Q. Which search problems are in P?
Q. Which search problems are not in P (intractable)?

A. No easy answers (we don't even know whether P = NP).

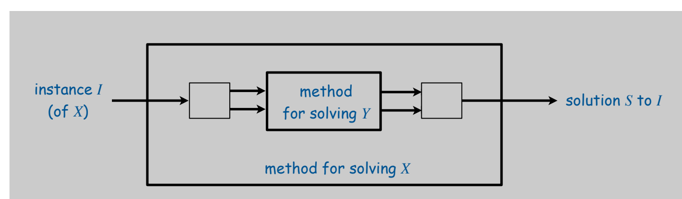
First step. Formalize notion:

*Problem X is computationally not much harder than problem Y.*

## Reductions

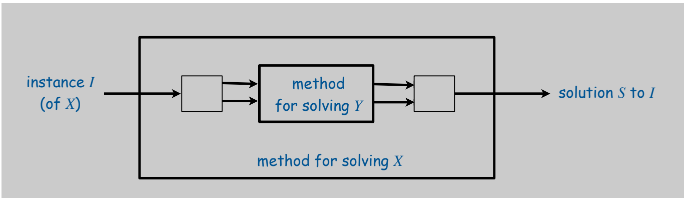
Def. Problem $X$ reduces to problem $Y$ if you can use an efficient solution to $Y$ to develop an efficient solution to $X$

To solve $X$, use:
- a poly number of standard computational steps, plus
- a poly number of calls to a method that solves instances of $Y$.



## Reductions: Consequences

Def. Problem $X$ reduces to problem $Y$ if you can solve $X$ given:
- A poly number of standard computational steps, plus
- A poly number of calls to a subroutine for solving instances of $Y$.




previously solved problem    your research problem

Design algorithms. If poly-time algorithm for $Y$, then one for $X$ too.
Establish intractability. If no poly-time algorithm for $X$, then none for $Y$.

SAT    your research problem

## LSOLVE Reduces to LP

LSOLVE. Given a system of linear equations, find a solution.

$$\begin{array}{rcrcrcr} 0x_0 & + & 1x_1 & + & 1x_2 & = & 4 \\ 2x_0 & + & 4x_1 & - & 2x_2 & = & 2 \\ 0x_0 & + & 3x_1 & + & 15x_2 & = & 36 \end{array}$$

*LSOLVE instance with n variables*

LP. Given a system of linear inequalities, find a solution.

$$\left.\begin{array}{rcrcrcr} 0x_0 & + & 1x_1 & + & 1x_2 & \le & 4 \\ 0x_0 & + & 1x_1 & + & 1x_2 & \ge & 4 \end{array}\right\} \Rightarrow 0x_0 + 1x_1 + 1x_1 = 4$$
$$\begin{array}{rcrcrcr} 2x_0 & + & 4x_1 & - & 2x_2 & \le & 2 \\ 2x_0 & + & 4x_1 & - & 2x_2 & \ge & 2 \\ 0x_0 & + & 3x_1 & + & 15x_2 & \le & 36 \\ 0x_0 & + & 3x_1 & + & 15x_2 & \ge & 36 \end{array}$$

*corresponding LP instance with n variables and 2n inequalities*

## SAT Reduces to ILP

SAT. Given a boolean equation $\Phi$, find a satisfying truth assignment.

$$\Phi = (x_1' \text{ or } x_2 \text{ or } x_3) \text{ and } (x_1 \text{ or } x_2' \text{ or } x_3) \text{ and } (x_1' \text{ or } x_2' \text{ or } x_3') \text{ and } (x_1' \text{ or } x_2' \text{ or } x_4)$$

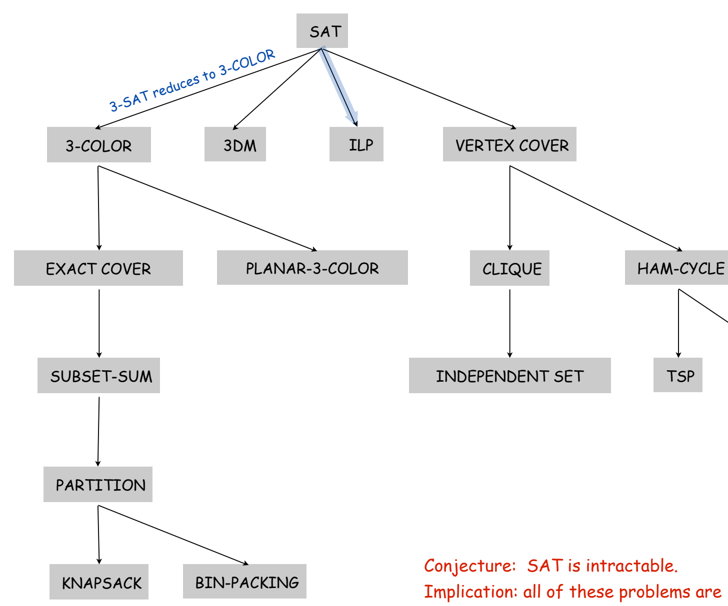
*SAT instance with n variables, k clauses*

ILP. Given a system of linear inequalities, find a 0-1 solution.

$$\underbrace{\begin{array}{rcl} C_1 & \ge & 1 - x_1 \\ C_1 & \ge & x_2 \\ C_1 & \ge & x_3 \\ C_1 & \le & (1 - x_1) + x_2 + x_3 \end{array}}_{C_1 = 1 \text{ iff clause 1 is satisfied}} \qquad \underbrace{\begin{array}{rcl} \Phi & \le & C_1 \\ \Phi & \le & C_2 \\ \Phi & \le & C_3 \\ \Phi & \le & C_4 \\ \Phi & \ge & C_1 + C_2 + C_3 + C_4 - 3 \end{array}}_{\Phi = 1 \text{ iff } C_1 = C_2 = C_3 = C_4 = 1}$$

*corresponding ILP instance with n + k + 1 variables and 4n + k + 1 inequalities*
*solution to this ILP instance gives solution to SAT instance*

## More Reductions From SAT




Dick Karp
'85 Turing award

Conjecture: SAT is intractable.
Implication: all of these problems are intractable.

## Still More Reductions from 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.
Mathematics. Given integer $a_1, \ldots, a_n$, 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, integer programming.
Physics. Partition function of 3d Ising model.
Politics. Shapley-Shubik voting power.
Pop culture. Versions of Sudoko, Checkers, Minesweeper, Tetris.
Statistics. Optimal experimental design.

Conjecture: no poly-time algorithm for SAT.
Implication: all of these problems are intractable.

6,000+ scientific papers per year.

# NP-completeness

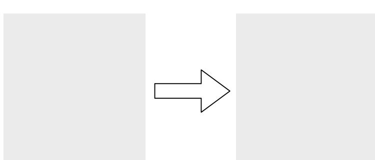
## NP-Completeness

Q. Why do we believe SAT has no poly-time algorithm?

Def. An NP problem is NP-complete if all problems in NP reduce to it.

every NP problem is a SAT problem in disguise

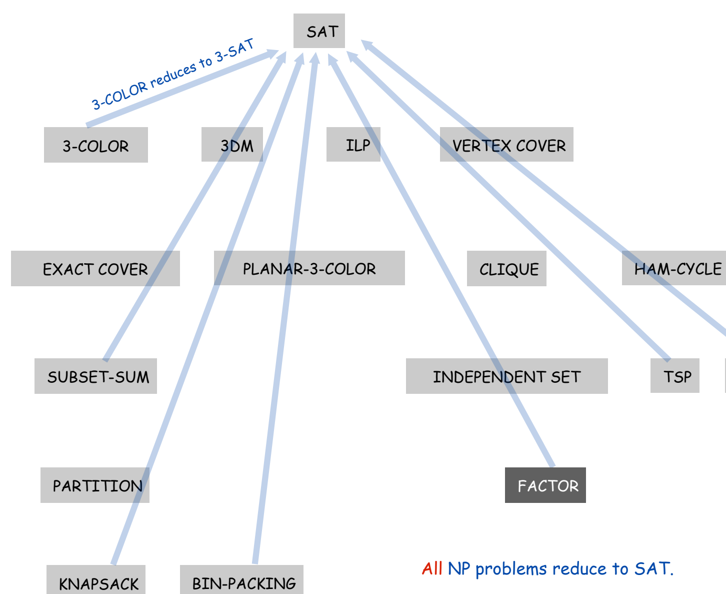
Theorem. [Cook 1971] SAT is NP-complete.
Extremely brief Proof Sketch:
- convert non-deterministic TM notation to SAT notation
- if you can solve SAT, you can solve any problem in NP

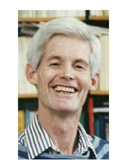


*nondeterministic TM* *SAT instance*

Corollary. Poly-time algorithm for SAT $\Rightarrow$ P = NP.

## Cook's Theorem

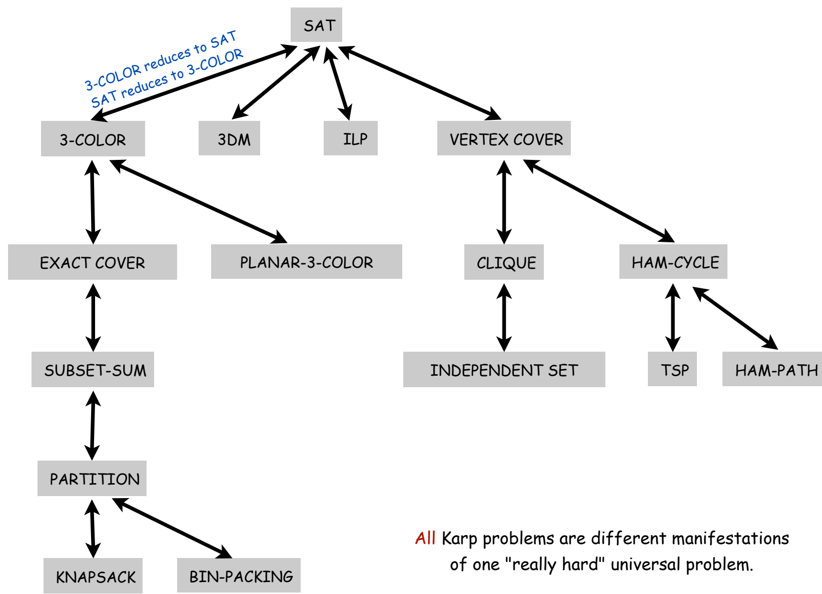


Stephen Cook
'82 Turing award

All NP problems reduce to SAT.

## Cook + Karp

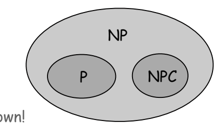



All Karp problems are different manifestations of one "really hard" universal problem.

## Two possible universes

P ≠ NP .
- Intractable search problems exist.
- Nondeterminism makes machines more efficient.
- Can prove that a problem is intractable by reduction from an NP-complete problem. ← no other way is known!
- Some search problems are neither NP-complete or in P. we don't know any useful ones
- Some search problems are still not classified.

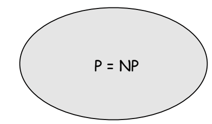
examples: factoring, graph isomorphism




P ≠ NP

P = NP.
- No intractable search problems exist.
- Nondeterminism is no help.
- Poly-time solutions exist for NP-complete problems

and all other search problems, such as factoring and graph isomorphism




P = NP

[Third possibility: Extended Church-Turing thesis is wrong.]

## Implications of NP-completeness

Implication. [SAT captures difficulty of whole class NP.]
- Poly-time algorithm for SAT iff P = NP (no intractable search problems exist).
- If some search problem is provably intractable, then so is SAT.

Remark. Can replace SAT above with any NP-complete problem.

Example: Proving a problem NP-complete guides scientific inquiry.
- 1926: Ising introduces simple model for phase transitions.
- 1944: Onsager finds closed form solution to 2D version in tour de force.
- 19xx: Feynman and other top minds seek 3D solution.
- 2000: SAT reduces to 3D-ISING.

a big goal of statistical mechanics

search for closed formula appears doomed since 3D-ISING is intractable if P ≠ NP

# Coping With Intractability

## Coping With Intractability

You have an NP-complete problem.
- It's safe to assume that it is intractable.
- What to do?

Relax one of desired features.
- Solve the problem in poly-time.
- Solve the problem to optimality.
- Solve arbitrary instances of the problem.

Complexity theory deals with worst case behavior.
- Instance(s) you want to solve may have easy-to-find answer.
- `Chaff` solves real-world SAT instances with ~ 10k variables.
  [Matthew Moskewicz '00, Conor Madigan '00, Sharad Malik]

PU senior independent work (!)

## Coping With Intractability

You have an NP-complete problem.
- It's safe to assume that it is intractable.
- What to do?

Relax one of desired features.
- Solve the problem in poly-time.
- Solve the problem to optimality.
- Solve arbitrary instances of the problem.

Develop a heuristic, and hope it produces a good solution.
- No guarantees on quality of solution.
- Ex. TSP assignment heuristics.
- Ex. Metropolis algorithm, simulating annealing, genetic algorithms.

Approximation algorithm. Find solution of provably good quality.
- Ex. MAX-3SAT: provably satisfy 87.5% as many clauses as possible.

but if you can guarantee to satisfy 87.51% as many clauses as possible in poly-time, then P = NP !

## Coping With Intractability

You have an NP-complete problem.
- It's safe to assume that it is intractable.
- What to do?

Relax one of desired features.
- Solve the problem in poly-time.
- Solve the problem to optimality.
- Solve arbitrary instances of the problem.

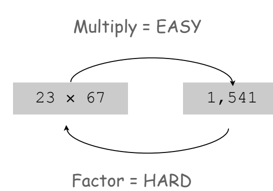
Special cases may be tractable.
- Ex: Linear time algorithm for 2-SAT.
- Ex: Linear time algorithm for Horn-SAT.

each clause has at most one un-negated literal

## Exploiting Intractability: Cryptography

Modern cryptography.
- Ex. Send your credit card to Amazon.
- Ex. Digitally sign an e-document.
- Enables freedom of privacy, speech, press, political association.

RSA cryptosystem.
- To use: multiply two $n$-bit integers. [poly-time]
- To break: factor a $2n$-bit integer. [unlikely poly-time]



## Exploiting Intractability: Cryptography

FACTOR. Given an $n$-bit integer $x$, find a nontrivial factor.

not 1 or x

```
74037563479561712828046796097429573142593188889231289084936232638972765034028266276891996419625117843995894330502127585370118968098286733173273108930900552505116877063299072396380786710086096962537934650563796359
```

Q. What is complexity of FACTOR?
A. In NP, but not known (or believed) to be in P or NP-complete.

Q. Is it safe to assume that FACTOR is intractable?
A. Maybe, but not as safe an assumption as for an NP-complete problem.

## Summary

P. Class of search problems solvable in poly-time.
NP. Class of all search problems, some of which seem wickedly hard.
NP-complete. Hardest problems in NP.
Intractable. Search problems not in P (if P ≠ NP).

Many fundamental problems are NP-complete
- TSP, SAT, 3-COLOR, ILP, (and thousands of others)
- 3D-ISING.

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