



<http://algs4.cs.princeton.edu>

## 6.6 INTRACTABILITY

---

- ▶ *introduction*
- ▶ *search problems*
- ▶  *$P$  vs.  $NP$*
- ▶ *classifying problems*
- ▶  *$NP$ -completeness*
- ▶ *coping with intractability*



## 6.6 INTRACTABILITY

---

- ▶ *introduction*
- ▶ *search problems*
- ▶  *$P$  vs.  $NP$*
- ▶ *classifying problems*
- ▶  *$NP$ -completeness*
- ▶ *coping with intractability*

# Questions about computation

---

- Q. What is a general-purpose computer?
- Q. Are there limits on the power of digital computers?
- Q. Are there limits on the power of machines we can build?



David Hilbert



Kurt Gödel



Alan Turing



Alonzo Church



John von Neumann

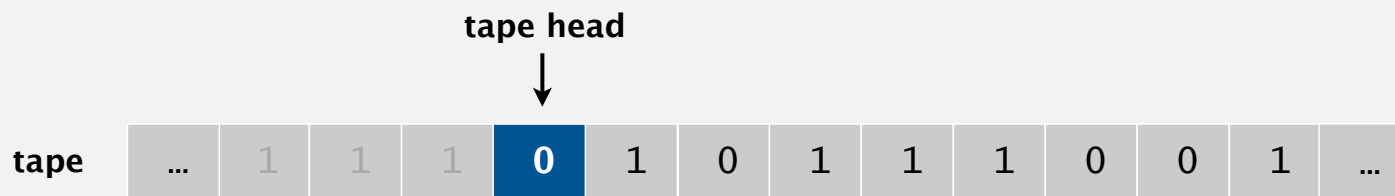
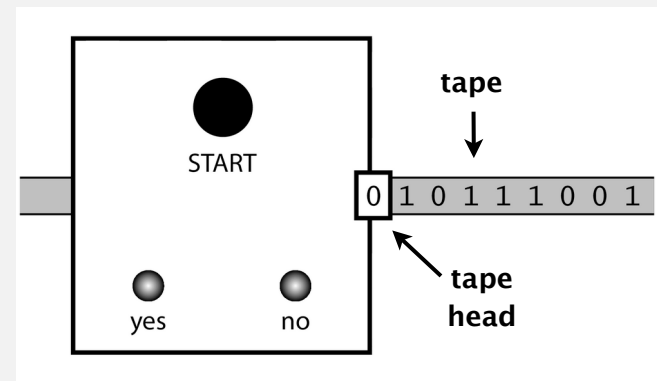
# A simple model of computation: DFAs

## Tape.

- Stores input.
- One arbitrarily long strip, divided into cells.
- Finite alphabet of symbols.

## Tape head.

- Points to one cell of tape.
- Reads a symbol from active cell.
- Moves one cell at a time.



Q. Is there a more powerful model of computation?

A. Yes.

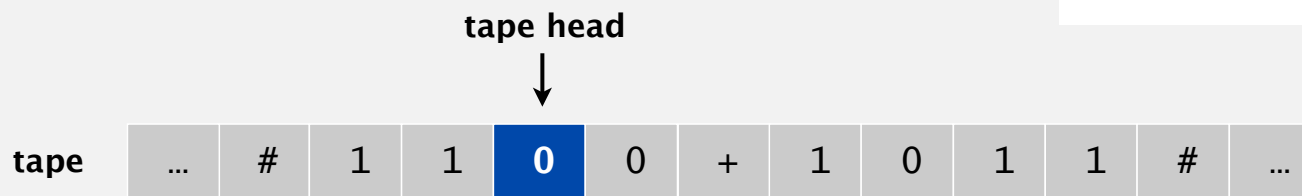
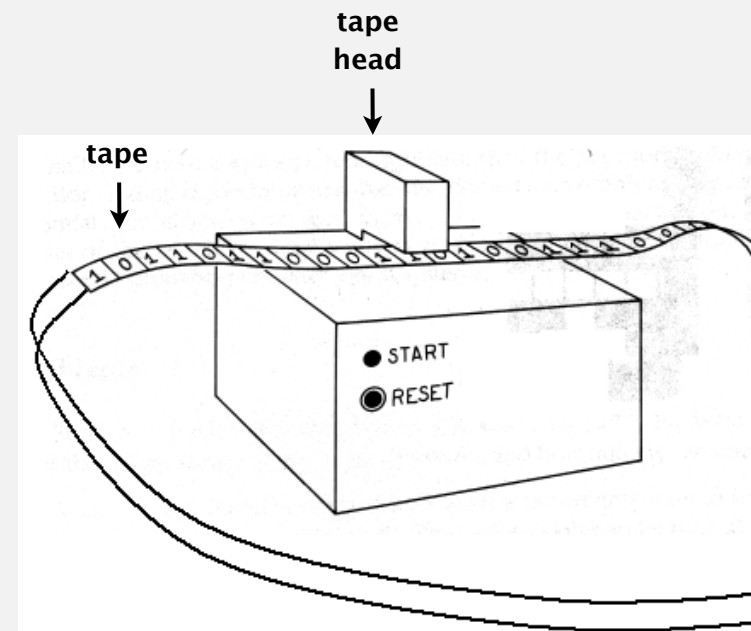
# A universal model of computation: Turing machines

## Tape.

- Stores input, **output**, and **intermediate results**.
- One arbitrarily long strip, divided into cells.
- Finite alphabet of symbols.

## Tape head.

- Points to one cell of tape.
- Reads a symbol from active cell.
- **Writes** a symbol to active cell.
- Moves one cell at a time.



Q. Is there a more powerful model of computation?

A. No! **←** most important scientific result of 20th century?

# Church-Turing thesis (1936)

---

**Turing machines can compute any function that can be computed by a physically harnessable process of the natural world.**

**Remark.** "Thesis" and not a mathematical theorem because it's a statement about the physical world and not subject to proof.

← but can be falsified

Use simulation to prove models equivalent.

- Android simulator on iPhone.
- iPhone simulator on Android.


**Implications.**

- No need to seek more powerful machines or languages.
- Enables rigorous study of computation (in this universe).

**Bottom line.** Turing machine is a **simple** and **universal** model of computation.

# Church-Turing thesis: evidence

---

- 8 decades without a counterexample.
- Many, many models of computation that turned out to be equivalent.  "universal"

model of computation	description
enhanced Turing machines	multiple heads, multiple tapes, 2D tape, nondeterminism
untyped lambda calculus	method to define and manipulate functions
recursive functions	functions dealing with computation on integers
unrestricted grammars	iterative string replacement rules used by linguists
extended L-systems	parallel string replacement rules that model plant growth
programming languages	Java, C, C++, Perl, Python, PHP, Lisp, PostScript, Excel
random access machines	registers plus main memory, e.g., TOY, Pentium
cellular automata	cells which change state based on local interactions
quantum computer	compute using superposition of quantum states
DNA computer	compute using biological operations on DNA

# A question about algorithms

---

**Q.** Which algorithms are useful in practice?

- Measure running time as a function of input size  $N$ .
- Useful in practice ("efficient") = polynomial time for all inputs.

$a N^b$



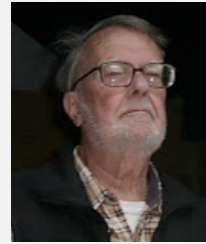
von Neumann  
(1953)



Nash  
(1955)



Gödel  
(1956)



Cobham  
(1964)



Edmonds  
(1965)



Rabin  
(1966)

**Ex 1.** Sorting  $N$  items takes  $N \log N$  compares using mergesort.

**Ex 2.** Finding best TSP tour on  $N$  points takes  $N!$  steps using brute search.

**Theory.** Definition is broad and robust.

constants  $a$  and  $b$  tend to be small, e.g.,  $3 N^2$

**Practice.** Poly-time algorithms scale to huge problems.



# Exponential growth

## Exponential growth dwarfs technological change.

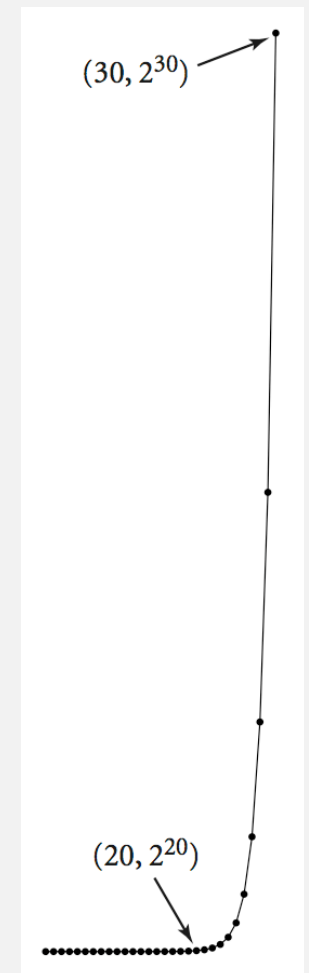
- Suppose you have a giant parallel computing device...
- With as many processors as electrons in the universe...
- And each processor has power of today's supercomputers...
- And each processor works for the life of the universe...

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 brute force.

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



# Questions about problems

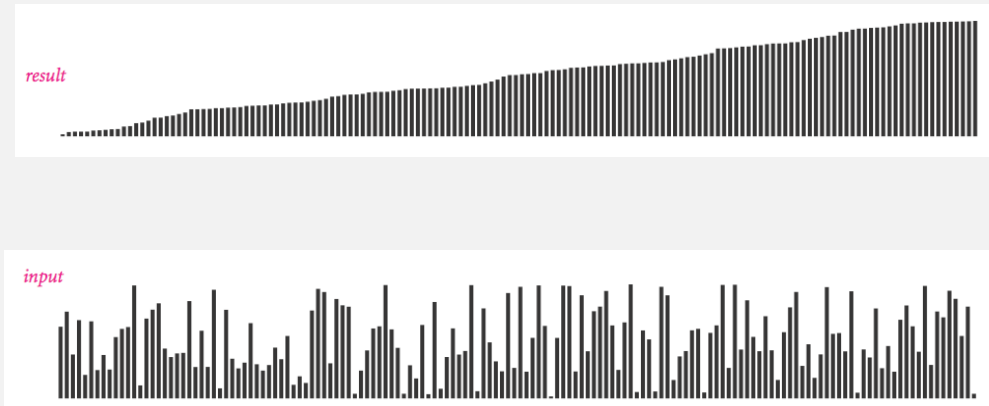
---

Q. Which problems can we solve in practice?

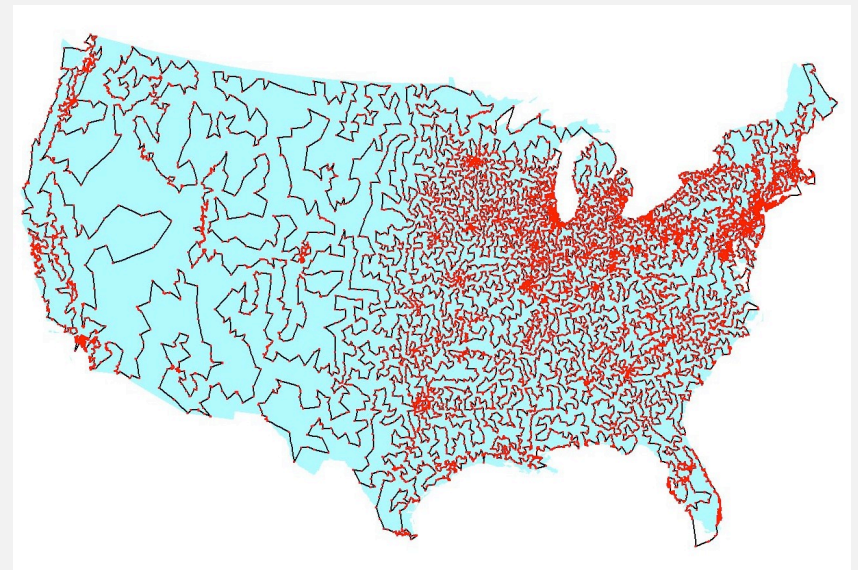
A. Those with poly-time algorithms.

Q. Which problems have 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

# Bird's-eye view

---

**Def.** A problem is **intractable** if it can't be solved in polynomial time.

**Desiderata.** Prove that a problem is intractable.

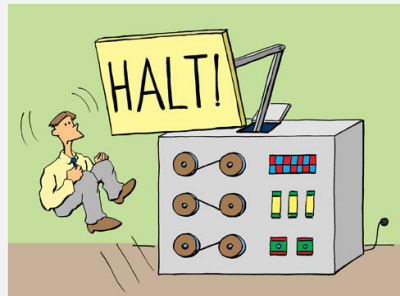
Two problems that provably require exponential time.

- Given a constant-size program, does it halt in at most  $K$  steps?
- Given  $N$ -by- $N$  checkers board position, can the first player force a win?

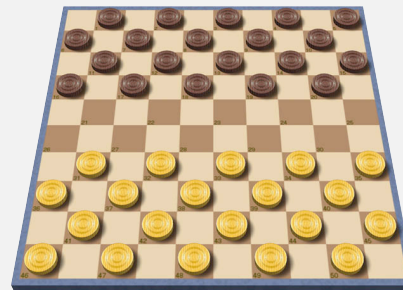
input size =  $c + \lg K$



using forced capture rule



*Alan designed the perfect computer*



**Frustrating news.** Very few successes.



## 6.6 INTRACTABILITY

---

- ▶ *introduction*
- ▶ *search problems*
- ▶  *$P$  vs.  $NP$*
- ▶ *classifying problems*
- ▶  *$NP$ -completeness*
- ▶ *coping with intractability*



## 6.6 INTRACTABILITY

---

- ▶ *introduction*
- ▶ *search problems*
- ▶  *$P$  vs.  $NP$*
- ▶ *classifying problems*
- ▶  *$NP$ -completeness*
- ▶ *coping with intractability*

# Four fundamental problems

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

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

$$\begin{array}{rcl} x_0 & = & -1 \\ x_1 & = & 2 \\ x_2 & = & 2 \end{array}$$

← variables are  
real numbers

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

$$\begin{array}{rrcr} 48x_0 & + & 16x_1 & + & 119x_2 & \leq & 88 \\ 5x_0 & + & 4x_1 & + & 35x_2 & \geq & 13 \\ 15x_0 & + & 4x_1 & + & 20x_2 & \geq & 23 \\ x_0 & , & x_1 & , & x_2 & \geq & 0 \end{array}$$

$$\begin{array}{rcl} x_0 & = & 1 \\ x_1 & = & 1 \\ x_2 & = & 1/5 \end{array}$$

← variables are  
real numbers

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

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

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

← variables are  
0 or 1

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

$$\begin{array}{l} (x'_1 \text{ or } x'_2) \text{ and } (x_0 \text{ or } x_2) = \text{true} \\ (x_0 \text{ or } x_1) \text{ and } (x_1 \text{ or } x'_2) = \text{false} \\ (x_0 \text{ or } x_2) \text{ and } (x'_0) = \text{true} \end{array}$$

$$\begin{array}{l} x_0 = \text{false} \\ x_1 = \text{false} \\ x_2 = \text{true} \end{array}$$

← 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 0-1 solution.

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

**Q.** Which of these problems have **poly-time** algorithms?

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

 but we still don't know for sure

# Search problems

---

**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  $I$

or report  
none exists





# Search problems

---

**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.

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

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

**instance I**

$$\begin{array}{rcl} x_0 & = & -1 \\ x_1 & = & 2 \\ x_2 & = & 2 \end{array}$$

**solution S**

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

# Search problems

---

**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.

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

$$\begin{array}{rclcl} 48x_0 & + & 16x_1 & + & 119x_2 & \leq & 88 \\ 5x_0 & + & 4x_1 & + & 35x_2 & \geq & 13 \\ 15x_0 & + & 4x_1 & + & 20x_2 & \geq & 23 \\ x_0 & , & x_1 & , & x_2 & \geq & 0 \end{array}$$

**instance I**

$$\begin{array}{rcl} x_0 & = & 1 \\ x_1 & = & 1 \\ x_2 & = & 1/5 \end{array}$$

**solution S**

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

# Search problems

---

**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.

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

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

**instance  $I$**

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

**solution  $S$**

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

# Search problems

---

**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.

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

$$\begin{array}{lcl} (x'_1 \text{ or } x'_2) \text{ and } (x_0 \text{ or } x_2) & = & \text{true} \\ (x_0 \text{ or } x_1) \text{ and } (x_1 \text{ or } x'_2) & = & \text{false} \\ (x_0 \text{ or } x_2) \text{ and } (x'_0) & = & \text{true} \end{array}$$

**instance I**

$$\begin{array}{lcl} x_0 & = & \text{false} \\ x_1 & = & \text{false} \\ x_2 & = & \text{true} \end{array}$$

**solution S**

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

# Search problems

---

**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.

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



input size = number of bits

147573952589676412927

instance  $I$

193707721

solution  $S$

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



## 6.6 INTRACTABILITY

---

- ▶ *introduction*
- ▶ *search problems*
- ▶  *$P$  vs.  $NP$*
- ▶ *classifying problems*
- ▶  *$NP$ -completeness*
- ▶ *coping with intractability*



## 6.6 INTRACTABILITY

---

- ▶ *introduction*
- ▶ *search problems*
- ▶  *$P$  vs.  $NP$*
- ▶ *classifying problems*
- ▶  *$NP$ -completeness*
- ▶ *coping with intractability*

# NP

**Def.** NP is the class of all search problems.

Note: classic definition limits  
NP to yes-no problems

problem	description	poly-time algorithm	instance I	solution S
LSOLVE ( $A, b$ )	Find a vector $x$ that satisfies $Ax = b$	Gaussian elimination	$\begin{array}{rcl} 0x_0 + 1x_1 + 1x_2 & = & 4 \\ 2x_0 + 4x_1 - 2x_2 & = & 2 \\ 0x_0 + 3x_1 + 15x_2 & = & 36 \end{array}$	$\begin{array}{rcl} x_0 & = & -1 \\ x_1 & = & 2 \\ x_2 & = & 2 \end{array}$
LP ( $A, b$ )	Find a vector $x$ that satisfies $Ax \leq b$	ellipsoid	$\begin{array}{rcl} 48x_0 + 16x_1 + 119x_2 & \leq & 88 \\ 5x_0 + 4x_1 + 35x_2 & \geq & 13 \\ 15x_0 + 4x_1 + 20x_2 & \geq & 23 \\ x_0, x_1, x_2 & \geq & 0 \end{array}$	$\begin{array}{rcl} x_0 & = & 1 \\ x_1 & = & 1 \\ x_2 & = & 1/5 \end{array}$
ILP ( $A, b$ )	Find a binary vector $x$ that satisfies $Ax \leq b$	???	$\begin{array}{rcl} & x_1 + x_2 & \geq 1 \\ x_0 & & + x_2 \geq 1 \\ x_0 + x_1 + x_2 & \leq & 2 \end{array}$	$\begin{array}{rcl} x_0 & = & 0 \\ x_1 & = & 1 \\ x_2 & = & 1 \end{array}$
SAT ( $\Phi, b$ )	Find a boolean vector $x$ that satisfies $\Phi(x) = b$	???	$\begin{array}{l} (x'_1 \text{ or } x'_2) \text{ and } (x_0 \text{ or } x_2) = \text{true} \\ (x_0 \text{ or } x_1) \text{ and } (x_1 \text{ or } x'_2) = \text{false} \\ (x_0 \text{ or } x_2) \text{ and } (x'_0) = \text{true} \end{array}$	$\begin{array}{l} x_0 = \text{false} \\ x_1 = \text{false} \\ x_2 = \text{true} \end{array}$
FACTOR ( $x$ )	Find a nontrivial factor of the integer $x$	???	147573952589676412927	193707721

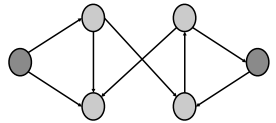
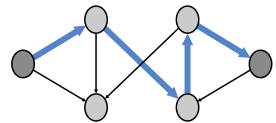
**Significance.** What scientists and engineers **aspire to compute** feasibly.



# P

**Def.** P is the class of search problems solvable in poly-time.

← Note: classic definition limits P to yes-no problems

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

**Significance.** What scientists and engineers **do compute** feasibly.

# Nondeterminism

---

Nondeterministic machine can **guess** the desired solution.

recall NFA implementation

**Ex.** `int[] a = new int[N];`

- Java: initializes entries to 0.
- Nondeterministic machine: initializes entries to the solution!

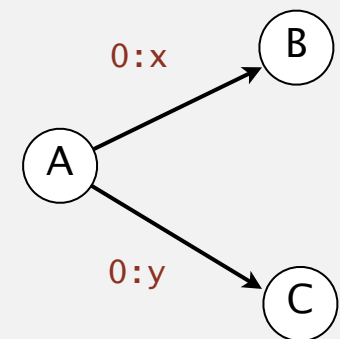
**ILP.** Given a system of linear inequalities, **guess** a 0-1 solution.

$x_1$	+	$x_2$	$\geq$	1		
$x_0$		+	$x_2$	$\geq$	1	
$x_0$	+	$x_1$	+	$x_2$	$\leq$	2

$x_0$	=	0
$x_1$	=	1
$x_2$	=	1

**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 TM.

# Extended Church-Turing thesis

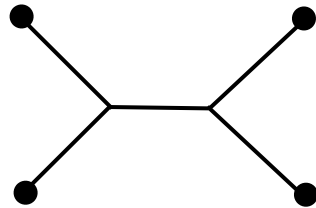
---

**P = search problems solvable in poly-time in the natural world.**

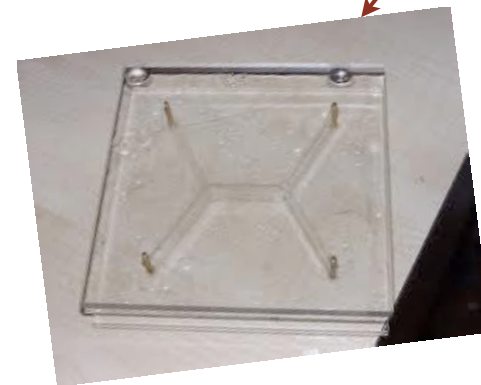
**Evidence supporting thesis.** True for all physical computers.

**Natural computers?** No successful attempts (yet).

Ex. Computing Steiner trees with soap bubbles



STEINER: Find set of lines of minimal length connecting  $N$  given points



doesn't work  
for large  $N$

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

# P vs. NP

---

Does  $P = NP$  ?



Copyright © 1990, Matt Groening



Copyright © 2000, Twentieth Century Fox

# Automating creativity

---

Q. Being creative vs. appreciating creativity?

Ex. Mozart composes a piece of music; our neurons appreciate it.

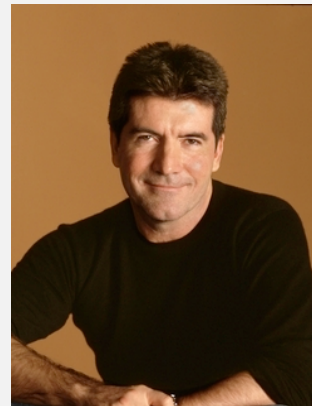
Ex. Wiles proves a deep theorem; a colleague referees it.

Ex. Boeing designs an efficient airfoil; a simulator verifies it.

Ex. Einstein proposes a theory; an experimentalist validates it.



creative



ordinary

Computational analog. Does  $P = NP$ ?

# The central question

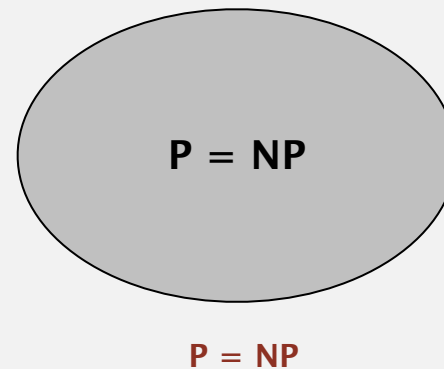
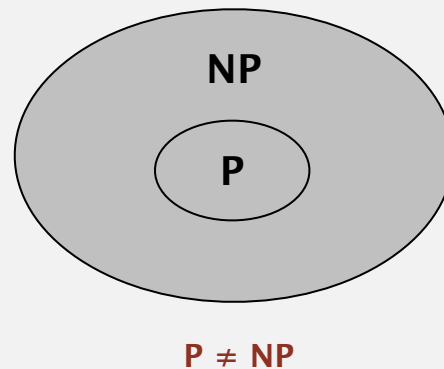
---

**P.** Class of search problems solvable in poly-time.

**NP.** Class of all search problems.

**Does  $P = NP$  ?** [Can you always avoid brute-force searching and do better]

Two worlds.



If  $P = NP$ ... Poly-time algorithms for SAT, ILP, TSP, FACTOR, ...

If  $P \neq NP$ ... Would learn something fundamental about our universe.

Overwhelming consensus.  $P \neq NP$ .


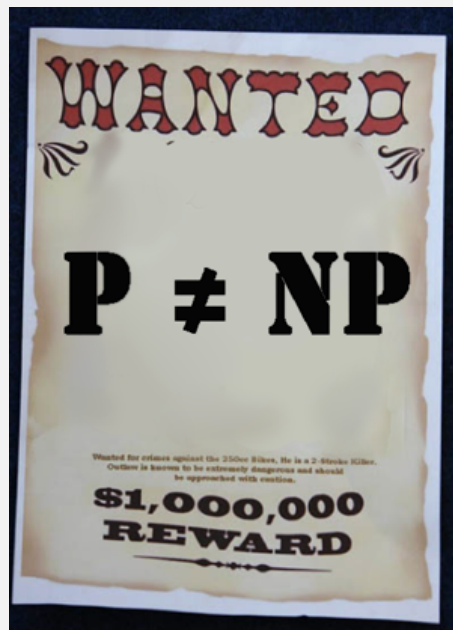
# The central question

**P.** Class of search problems solvable in poly-time.

**NP.** Class of all search problems.

**Does  $P = NP$  ?** [Can you always avoid brute-force searching and do better]

**Millennium prize.** \$1 million for resolution of  $P = NP$  problem.



**Clay Mathematics Institute**  
*Dedicated to increasing and disseminating mathematical knowledge*

HOME | ABOUT CMI | PROGRAMS | NEWS & EVENTS | AWARDS | SCHOLARS | PUBLICATIONS

### Millennium Problems

In order to celebrate mathematics in the new millennium, The Clay Mathematics Institute of Cambridge, Massachusetts (CMI) has named seven *Prize Problems*. The Scientific Advisory Board of CMI selected these problems, focusing on important classic questions that have resisted solution over the years. The Board of Directors of CMI designated a \$7 million prize fund for the solution to these problems, with \$1 million allocated to each. During the [Millennium Meeting](#) held on May 24, 2000 at the Collège de France, Timothy Gowers presented a lecture entitled *The Importance of Mathematics*, aimed for the general public, while John Tate and Michael Atiyah spoke on the problems. The CMI invited specialists to formulate each problem.

- [Birch and Swinnerton-Dyer Conjecture](#)
- [Hodge Conjecture](#)
- [Navier-Stokes Equations](#)
- [P vs NP](#)
- [Poincaré Conjecture](#)
- [Riemann Hypothesis](#)
- [Yang-Mills Theory](#)
- [Rules](#)
- [Millennium Meeting Videos](#)



## 6.6 INTRACTABILITY

---

- ▶ *introduction*
- ▶ *search problems*
- ▶  *$P$  vs.  $NP$*
- ▶ *classifying problems*
- ▶  *$NP$ -completeness*
- ▶ *coping with intractability*





## 6.6 INTRACTABILITY

---

- ▶ *introduction*
- ▶ *search problems*
- ▶  *$P$  vs.  $NP$*
- ▶ *classifying problems*
- ▶  *$NP$ -completeness*
- ▶ *coping with intractability*

# Periodic table of the elements

# Periodic Table of the Elements

	IA																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																		</
--	----	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	----

# A key problem: satisfiability

---

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

$$x'_1 \text{ or } x_2 \text{ or } x_3 = \text{true}$$

$$x_1 \text{ or } x'_2 \text{ or } x_3 = \text{true}$$

$$x'_1 \text{ or } x'_2 \text{ or } x'_3 = \text{true}$$

$$x'_1 \text{ or } x'_2 \text{ or } x_4 = \text{true}$$

## Key applications.

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

# 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.

"intractable"



# Classifying problems

---

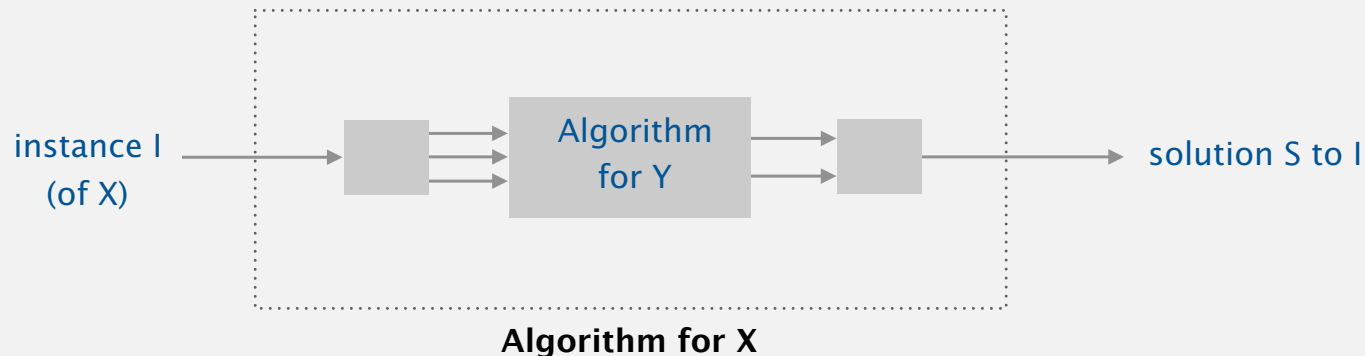
Q. Which search problems are in P?

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

↙ Cook reduction

Problem  $X$  **poly-time reduces** to problem  $Y$  if  $X$  can be solved with:

- Polynomial number of standard computational steps.
- Polynomial number of calls to  $Y$ .



**Consequence.** If SAT poly-time reduces to  $Y$ , then we conclude that  $Y$  is (probably) intractable.

# SAT poly-time reduces to ILP

---

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

$$x'_1 \text{ or } x_2 \text{ or } x_3 = \text{true}$$

$$x_1 \text{ or } x'_2 \text{ or } x_3 = \text{true}$$

$$x'_1 \text{ or } x'_2 \text{ or } x'_3 = \text{true}$$

$$x'_1 \text{ or } x'_2 \text{ or } x_4 = \text{true}$$

← can to reduce any SAT problem to this form

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

$$1 \leq (1 - x_1) + x_2 + x_3$$

$$1 \leq x_1 + (1 - x_2) + x_3$$

$$1 \leq (1 - x_1) + (1 - x_2) + (1 - x_3)$$

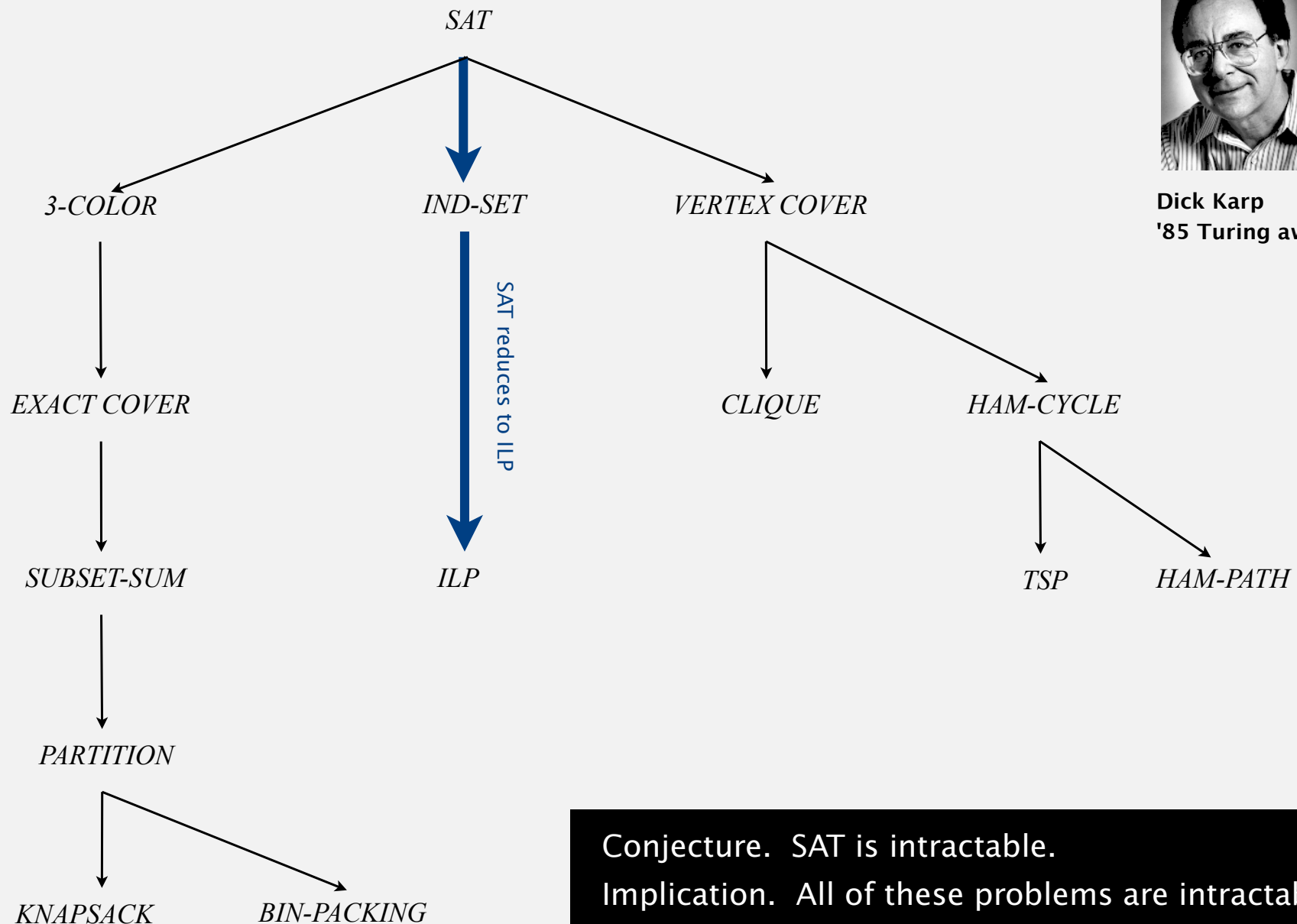
$$1 \leq (1 - x_1) + (1 - x_2) + x_4$$

**solution to this ILP instance gives solution to original SAT instance**

# More poly-time reductions from boolean satisfiability



Dick Karp  
'85 Turing award



## 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  $a_1, \dots, a_n$ , compute  $\int_0^{2\pi} \cos(a_1\theta) \times \cos(a_2\theta) \times \dots \times \cos(a_n\theta) d\theta$   
**Mechanical engineering.** Structure of turbulence in sheared flows.  
**Medicine.** Reconstructing 3d shape from biplane angiocardigram.  
**Operations research.** Traveling salesperson problem.  
**Physics.** Partition function of 3d Ising model.  
**Politics.** Shapley-Shubik voting power.  
**Recreation.** Versions of Sudoku, Checkers, Minesweeper, Tetris.  
**Statistics.** Optimal experimental design.

**plus over 6,000 scientific papers per year**





## 6.6 INTRACTABILITY

---

- ▶ *introduction*
- ▶ *search problems*
- ▶  *$P$  vs.  $NP$*
- ▶ *classifying problems*
- ▶  *$NP$ -completeness*
- ▶ *coping with intractability*



## 6.6 INTRACTABILITY

---

- ▶ *introduction*
- ▶ *search problems*
- ▶  *$P$  vs.  $NP$*
- ▶ *classifying problems*
- ▶  *$NP$ -completeness*
- ▶ *coping with intractability*

# NP-completeness

---

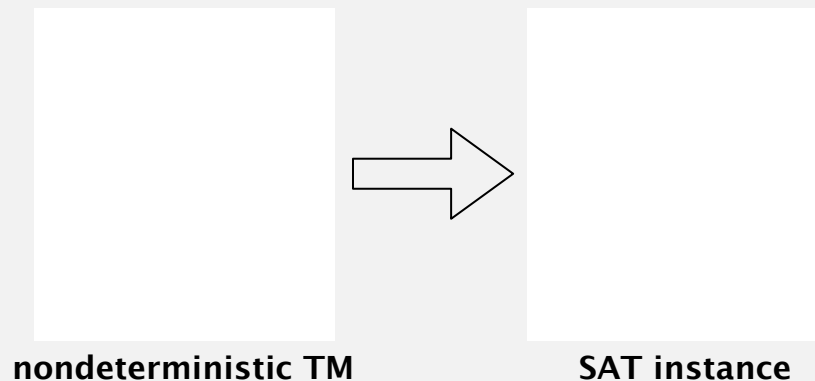
**Def.** An NP problem is **NP-complete** if every problem in NP poly-time reduce to it.

**Proposition.** [Cook 1971, Levin 1973] SAT is NP-complete.

every NP problem is a  
SAT problem in disguise

**Extremely brief proof sketch:**

- Convert non-deterministic TM notation to SAT notation.
- If you can solve SAT, you can solve any problem in NP.



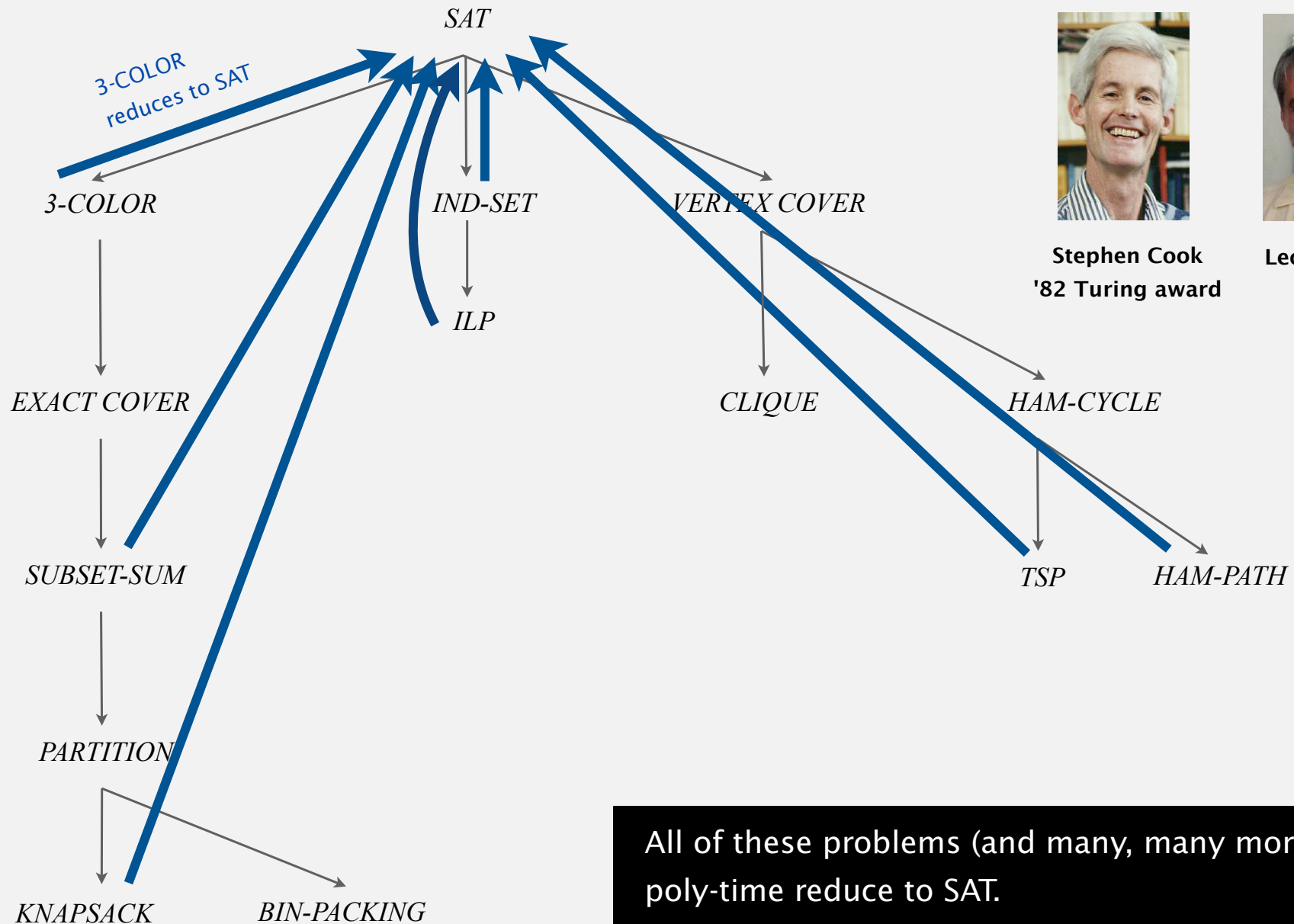
**Corollary.** Poly-time algorithm for SAT iff  $P = NP$ .

# You NP-complete me

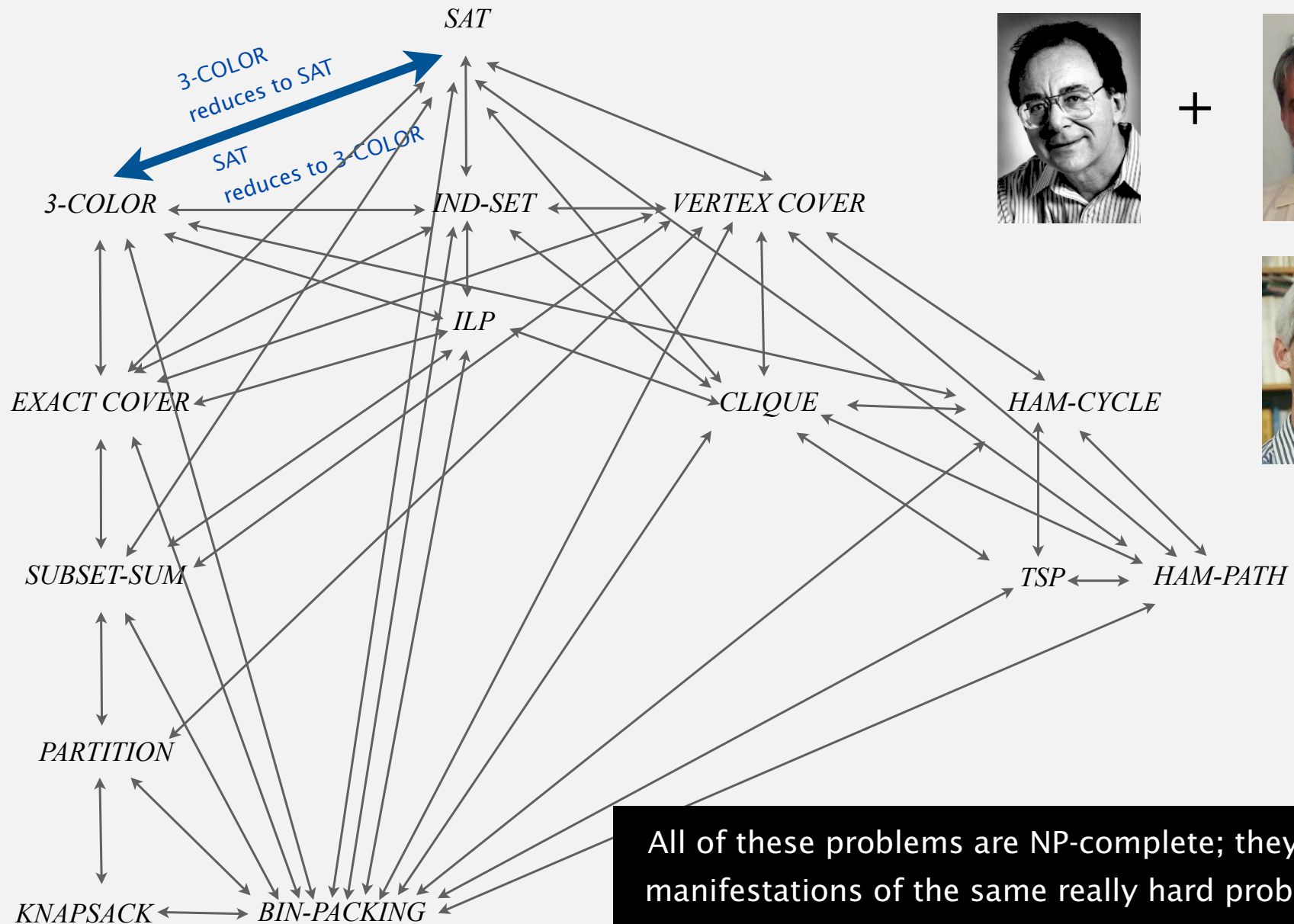
---



# Implications of Cook-Levin theorem



# Implications of Karp + Cook-Levin



# Implications of NP-Completeness

---

**Implication.** [SAT captures difficulty of whole class NP]

- Poly-time algorithm for SAT iff  $P = NP$ .
- No poly-time algorithm for some NP problem  $\Rightarrow$  none for SAT.

**Remark.** Can replace SAT with any of Karp's problems.

**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: 3D-ISING proved NP-complete.

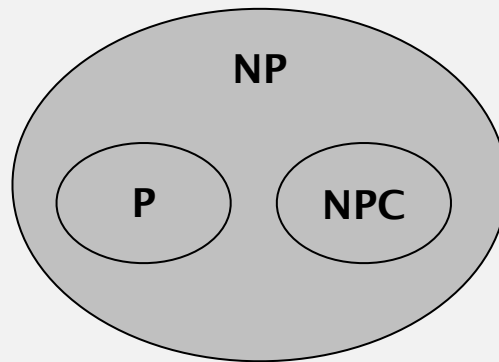
a holy grail of statistical mechanics

search for closed formula appears doomed

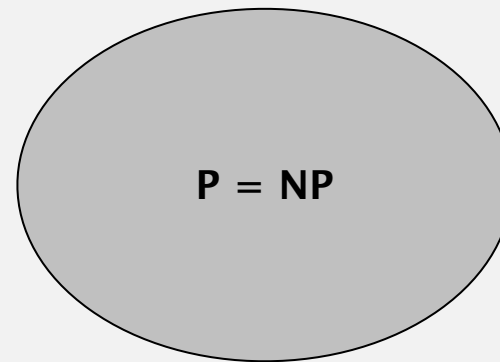
## Two worlds (more detail)

---

Overwhelming consensus (still).  $P \neq NP$ .



$P \neq NP$



$P = NP$

Why we believe  $P \neq NP$ .

*“ We admire Wiles' proof of Fermat's last theorem, the scientific theories of Newton, Einstein, Darwin, Watson and Crick, the design of the Golden Gate bridge and the Pyramids, precisely because they seem to require a leap which cannot be made by everyone, let alone a by simple mechanical device. ”* — *Avi Wigderson*



# 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.** Problem with no poly-time algorithm.

Many fundamental problems are NP-complete.

- SAT, ILP, HAMILTON-PATH, ...
- 3D-ISING, ...

Use theory a guide:

- A poly-time algorithm for an NP-complete problem would be a stunning breakthrough (a proof that  $P = NP$ ).
- You will confront NP-complete problems in your career.
- Safe to assume that  $P \neq NP$  and that such problems are intractable.
- Identify these situations and proceed accordingly.



Princeton CS Building, West Wall, Circa 2001





# Princeton CS Building, West Wall, Circa 2001

---



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



## 6.6 INTRACTABILITY

---

- ▶ *introduction*
- ▶ *search problems*
- ▶  *$P$  vs.  $NP$*
- ▶ *classifying problems*
- ▶  *$NP$ -completeness*
- ▶ *coping with intractability*



## 6.6 INTRACTABILITY

---

- ▶ *introduction*
- ▶ *search problems*
- ▶  *$P$  vs.  $NP$*
- ▶ *classifying problems*
- ▶  *$NP$ -completeness*
- ▶ *coping with intractability*

# Exploiting intractability

---

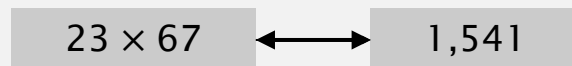
## Modern cryptography.

- 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 2  $n$ -bit integer. [unlikely poly-time]

Multiply = EASY



Factor = HARD

# Exploiting intractability

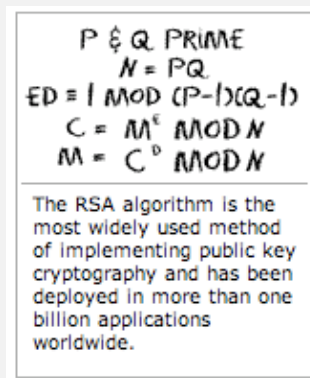
**Challenge.** Factor this number.

740375634795617128280467960974295731425931888892312890849362  
326389727650340282662768919964196251178439958943305021275853  
701189680982867331732731089309005525051168770632990723963807  
86710086096962537934650563796359

RSA-704

(\$30,000 prize if you can factor)

**Can't do it?** Create a company based on the difficulty of factoring.



**RSA algorithm**



**RSA sold  
for \$2.1 billion**



**or design a t-shirt**



# Exploiting intractability

---

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

**Q.** What is complexity of FACTOR?

**A.** In NP, but not known (or believed) to be in P or NP-complete.

**Q.** What if  $P = NP$ ?

**A.** Poly-time algorithm for factoring; modern e-economy collapses.

**Proposition.** [Shor 1994] Can factor an  $n$ -bit integer in  $n^3$  steps on a "quantum computer."

**Q.** Do we still believe the extended Church-Turing thesis???





# Coping with intractability

---

## Relax one of desired features.

- Solve arbitrary instances of the problem.
- Solve the problem to optimality.
- Solve the problem in poly-time.

## Special cases may be tractable.

- Ex: Linear time algorithm for 2-SAT. ← at most two variables per equation
- Ex: Linear time algorithm for Horn-SAT. ← at most one un-negated variable per equation

# Coping with intractability

---

## Relax one of desired features.

- Solve arbitrary instances of the problem.
- Solve the problem to optimality.
- Solve the problem in poly-time.

## 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

---

## Relax one of desired features.

- Solve arbitrary instances of the problem.
- Solve the problem to optimality.
- Solve the problem in poly-time.

## Complexity theory deals with worst case behavior.

- Instance(s) you want to solve may be "easy."
- Chaff solves real-world SAT instances with  $\sim 10K$  variable.

### Chaff: Engineering an Efficient SAT Solver

Matthew W. Moskewicz  
Department of EECS  
UC Berkeley

moskewcz@alumni.princeton.edu

Conor F. Madigan  
Department of EECS  
MIT

cmadigan@mit.edu

Ying Zhao, Lintao Zhang, Sharad Malik  
Department of Electrical Engineering  
Princeton University

{yingzhao, lintaoz, sharad}@ee.princeton.edu

#### ABSTRACT

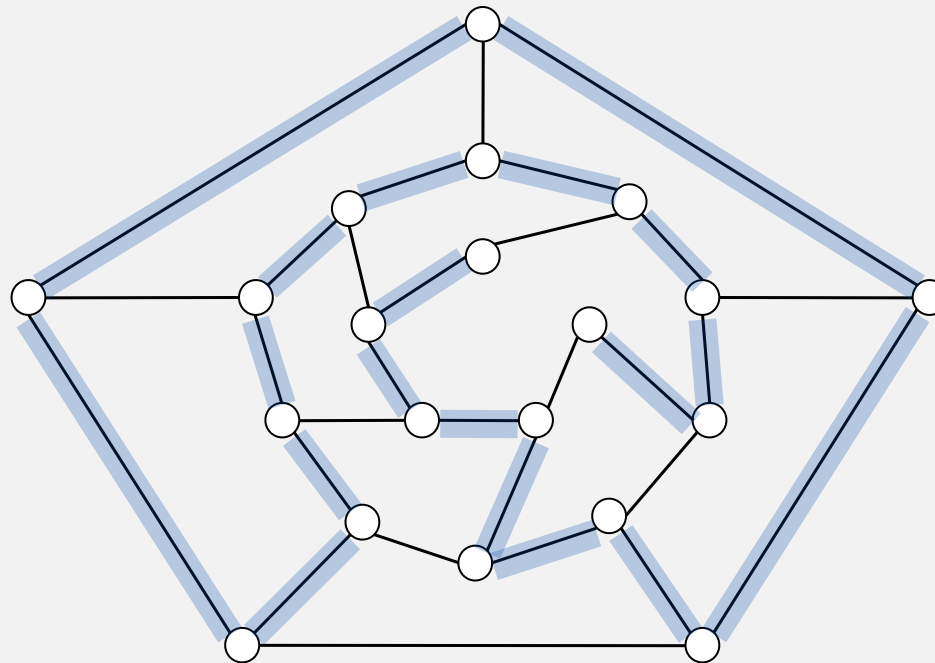
Boolean Satisfiability is probably the most studied of combinatorial optimization/search problems. Significant effort has been devoted to trying to provide practical solutions to this problem for problem instances encountered in a range of applications in Electronic Design Automation (EDA), as well as in Artificial Intelligence (AI). This study has culminated in the

Many publicly available SAT solvers (e.g. GRASP [8], POSIT [5], SATO [13], rel\_sat [2], WalkSAT [9]) have been developed, most employing some combination of two main strategies: the Davis-Putnam (DP) backtrack search and heuristic local search. Heuristic local search techniques are not guaranteed to be complete (i.e. they are not guaranteed to find a satisfying assignment if one exists or prove unsatisfiability); as a

# Hamilton path

---

**Goal.** Find a simple path that visits every vertex exactly once.



visit every edge exactly once



**Remark.** Euler path easy, but Hamilton path is NP-complete.

# Hamilton path: Java implementation

```
public class HamiltonPath
{
    private boolean[] marked;    // vertices on current path
    private int count = 0;      // number of Hamiltonian paths

    public HamiltonPath(Graph G)
    {
        marked = new boolean[G.V()];
        for (int v = 0; v < G.V(); v++)
            dfs(G, v, 1);
    }

    private void dfs(Graph G, int v, int depth)
    {
        marked[v] = true;
        if (depth == G.V()) count++;

        for (int w : G.adj(v))
            if (!marked[w]) dfs(G, w, depth+1);

        marked[v] = false;
    }
}
```

found one →

length of current path  
(depth of recursion)

backtrack if w is  
already part of path

clean up

# The longest path

---



*Woh-oh-oh-oh, find the longest path!*  
*Woh-oh-oh-oh, find the longest path!*

*If you said P is NP tonight,  
There would still be papers left to write.  
I have a weakness;  
I'm addicted to completeness,  
And I keep searching for the longest path.*

*The algorithm I would like to see  
Is of polynomial degree.  
But it's elusive:  
Nobody has found conclusive  
Evidence that we can find a longest path.*

*I have been hard working for so long.  
I swear it's right, and he marks it wrong.  
Some how I'll feel sorry when it's done: GPA 2.1  
Is more than I hope for.*

*Garey, Johnson, Karp and other men (and women)  
Tried to make it order  $N \log N$ .  
Am I a mad fool  
If I spend my life in grad school,  
Forever following the longest path?*

*Woh-oh-oh-oh, find the longest path!*  
*Woh-oh-oh-oh, find the longest path!*  
*Woh-oh-oh-oh, find the longest path.*

**Written by Dan Barrett in 1988 while a student  
at Johns Hopkins during a difficult algorithms take-home final**



That's all, folks: keep searching!

---



**The world's longest path (Sendero de Chile): 9,700 km.  
(originally scheduled for completion in 2010; now delayed until 2038)**



## 6.6 INTRACTABILITY

---

- ▶ *introduction*
- ▶ *search problems*
- ▶  *$P$  vs.  $NP$*
- ▶ *classifying problems*
- ▶  *$NP$ -completeness*
- ▶ *coping with intractability*





<http://algs4.cs.princeton.edu>

## 6.6 INTRACTABILITY

---

- ▶ *introduction*
- ▶ *search problems*
- ▶  *$P$  vs.  $NP$*
- ▶ *classifying problems*
- ▶  *$NP$ -completeness*
- ▶ *coping with intractability*