BBM 202 - ALGORITHMS

HACETTEPE UNIVERSITY

DEPT. OF COMPUTER ENGINEERING

ERKUT ERDEM

INTRACTABILITY

May. 20, 2014

Acknowledgement: The course slides are adapted from the slides prepared by R. Sedgewick and K. Wayne of Princeton University.

TODAY

- Intractability
- Search problems
- P vs. NP
- Classifying problems
- NP-completeness

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

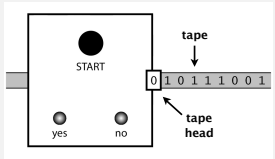

Alonzo Church

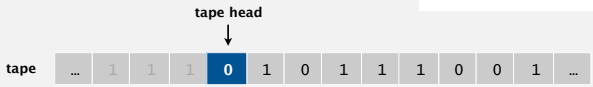

John von Neumann

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.



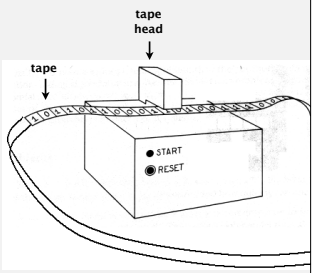


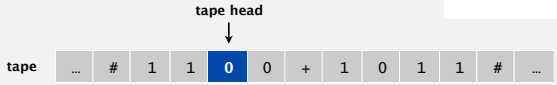
Q. Is there a more powerful model of computation?
A. Yes.

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.

Use simulation to prove models equivalent. ← but can be falsified
- 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.

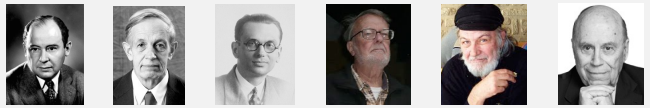
Church-Turing thesis: evidence

- 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.
Practice. Poly-time algorithms scale to huge problems. ← constants a and b tend to be small, e.g., $3 N^2$

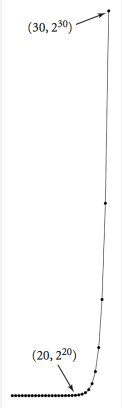
Exponential growth

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^{80}
supercomputer instructions per second	10^{13}
age of universe in seconds	10^{17}

† estimated



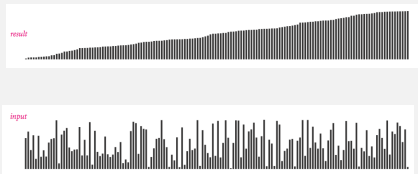
- Will not help solve 1,000 city TSP problem via brute force.

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

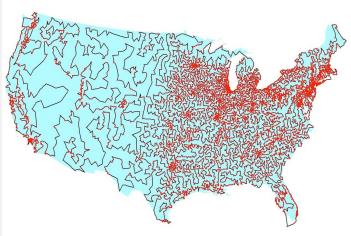
Questions about problems

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.

input size = c + lg K

- 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?

using forced capture rule

Alan designed the perfect computer

Frustrating news. Very few successes.

Intractability

- Search problems
- P vs. NP
- Classifying problems
- NP-completeness

Four fundamental problems

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

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

variables are real numbers

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

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

variables are real numbers

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

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

variables are 0 or 1

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

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

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.

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

instance I **solution S**

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

Search problems

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.

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

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.

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

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`.

Intractability

- Search problems
- P vs. NP
- Classifying problems
- NP-completeness

NP

Def. NP is the class of all search problems.

Note: classic definition limits NP to yes-no problems

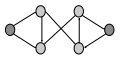
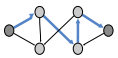
problem	description	poly-time algorithm	instance I	solution S
LSOLVE	Find a vector x that satisfies $Ax = b$	Gaussian elimination	$\begin{aligned} 0x_0 + 1x_1 + 1x_2 &= 4 \\ 2x_0 + 4x_1 - 2x_2 &= 2 \\ 0x_0 + 3x_1 + 15x_2 &= 36 \end{aligned}$	$\begin{aligned} x_0 &= -1 \\ x_1 &= 2 \\ x_2 &= 2 \end{aligned}$
LP	Find a vector x that satisfies $Ax \le b$	ellipsoid	$\begin{aligned} 48x_0 + 16x_1 + 119x_2 &\le 88 \\ 5x_0 + 4x_1 + 35x_2 &\ge 13 \\ 15x_0 + 4x_1 + 20x_2 &\ge 23 \\ x_0 \quad x_1 \quad x_2 &\ge 0 \end{aligned}$	$\begin{aligned} x_0 &= 1 \\ x_1 &= 1 \\ x_2 &= 1/5 \end{aligned}$
ILP	Find a binary vector x that satisfies $Ax \le b$	???	$\begin{aligned} x_1 + x_2 &\ge 1 \\ x_0 + x_2 &\ge 1 \\ x_0 + x_1 + x_2 &\le 2 \end{aligned}$	$\begin{aligned} x_0 &= 0 \\ x_1 &= 1 \\ x_2 &= 1 \end{aligned}$
SAT	Find a boolean vector x that satisfies $\Phi(x) = b$	???	$\begin{aligned} (x'_1 \text{ or } x'_2) \text{ and } (x_0 \text{ or } x_2) &= \text{true} \\ (x_0 \text{ or } x_1) \text{ and } (x_1 \text{ or } x'_2) &= \text{false} \\ (x_0 \text{ or } x_2) \text{ and } (x'_0) &= \text{true} \end{aligned}$	$\begin{aligned} x_0 &= \text{false} \\ x_1 &= \text{false} \\ x_2 &= \text{true} \end{aligned}$
FACTOR	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	Find a vector x that satisfies $Ax = b$	Gaussian elimination (Edmonds 1967)	$\begin{aligned} 0x_0 + 1x_1 + 1x_2 &= 4 \\ 2x_0 + 4x_1 - 2x_2 &= 2 \\ 0x_0 + 3x_1 + 15x_2 &= 36 \end{aligned}$	$\begin{aligned} x_0 &= -1 \\ x_1 &= 2 \\ x_2 &= 2 \end{aligned}$
LP	Find a vector x that satisfies $Ax \le b$	ellipsoid (Khachiyan 1979)	$\begin{aligned} 48x_0 + 16x_1 + 119x_2 &\le 88 \\ 5x_0 + 4x_1 + 35x_2 &\ge 13 \\ 15x_0 + 4x_1 + 20x_2 &\ge 23 \\ x_0 \quad x_1 \quad x_2 &\ge 0 \end{aligned}$	$\begin{aligned} x_0 &= 1 \\ x_1 &= 1 \\ x_2 &= 1/5 \end{aligned}$
SORT	Find a permutation that puts array a in order	mergesort (von Neumann 1945)	2.3 8.5 1.2 9.1 2.2 0.3	5 2 4 0 1 3
STCONN	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];

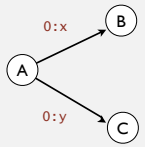
- Java: initializes entries to 0.
- Nondeterministic machine: initializes entries to the solution!

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

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

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.

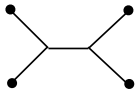
Extended Church-Turing thesis

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

Evidence supporting thesis. True for all physical computers.

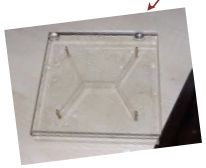
Natural computers? No successful attempts (yet).

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 ?

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.

P ≠ NP P = NP

If P = NP… Poly-time algorithms for SAT, ILP, TSP, FACTOR, …
If P ≠ NP… Would learn something fundamental about our universe.

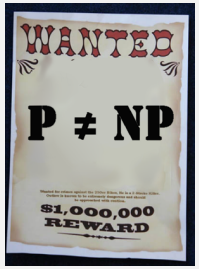
Overwhelming consensus. P ≠ NP.

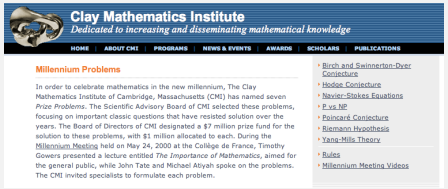
The central question

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]

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





INTRACTABILITY

- Search problems
- P vs. NP
- **Classifying problems**
- NP-completeness

A key problem: satisfiability

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

$$x'_1 \text{ or } x_2 \text{ or } x_3 = true$$
$$x_1 \text{ or } x'_2 \text{ or } x_3 = true$$
$$x'_1 \text{ or } x'_2 \text{ or } x'_3 = true$$
$$x'_1 \text{ or } x'_2 \text{ or } x_4 = true$$

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"

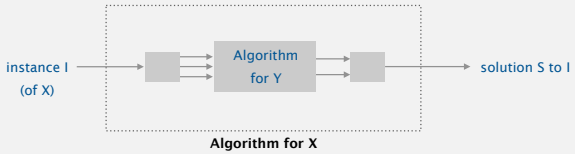
Classifying problems

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.

$$
\begin{aligned}
x'_1 \text{ or } x_2 \text{ or } x_3 &= \text{true} \\
x_1 \text{ or } x'_2 \text{ or } x_3 &= \text{true} \\
x'_1 \text{ or } x'_2 \text{ or } x'_3 &= \text{true} \\
x'_1 \text{ or } x'_2 \text{ or } x_4 &= \text{true}
\end{aligned}
$$

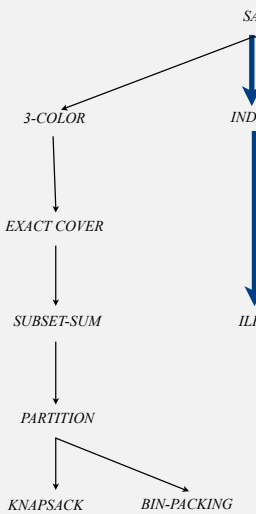
can to reduce any SAT problem to this form

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

$$
\begin{aligned}
1 &\le (1 - x_1) + x_2 + x_3 \\
1 &\le x_1 + (1 - x_2) + x_3 \\
1 &\le (1 - x_1) + (1 - x_2) + (1 - x_3) \\
1 &\le (1 - x_1) + (1 - x_2) + x_4
\end{aligned}
$$

solution to this ILP instance gives solution to original SAT instance

More poly-time reductions from boolean satisfiability



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.
Physics. Partition function of 3d Ising model.
Politics. Shapley-Shubik voting power.
Recreation. Versions of Sudoko, Checkers, Minesweeper, Tetris.
Statistics. Optimal experimental design.

plus over 6,000 scientific papers per year

Intractability

- Search problems
- P vs. NP
- Classifying problems
- NP-completeness

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

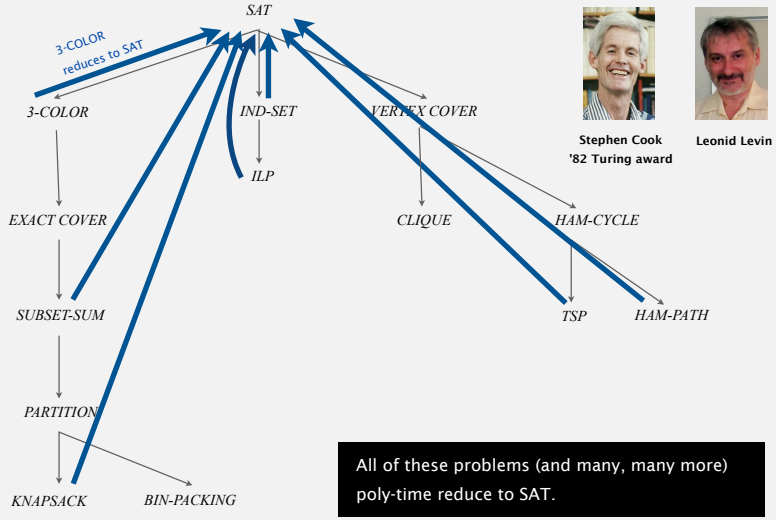
Extremely brief proof sketch:

- 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.

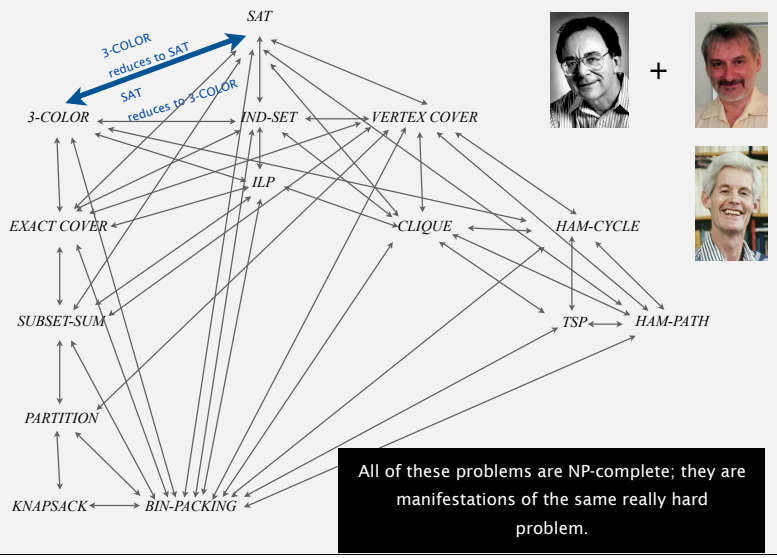
You NP-complete me

Implications of Cook-Levin theorem



All of these problems (and many, many more) poly-time reduce to SAT.

Implications of Karp + Cook-Levin



+

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

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 ⇒ none for SAT.

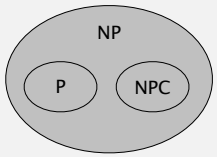
Remark. Can replace SAT with any of Karp's problems.

Proving a problem NP-complete guides scientific inquiry.

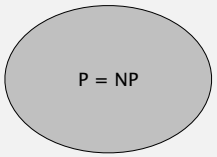
- 1926: Ising introduces simple model for phase transitions.
- 1944: Onsager finds closed form solution to 2D version in tour de force.
- 19xx: Feynman and other top minds seek 3D solution. a holy grail of statistical mechanics
- 2000: 3D-ISING proved NP-complete. search for closed formula appears doomed

Two worlds (more detail)

Overwhelming consensus (still). P ≠ NP.



P ≠ NP



P = NP

Why we believe P ≠ 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 ≠ NP and that such problems are intractable.
- Identify these situations and proceed accordingly.

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

23 × 67 ⟷ 1,541

Factor = HARD

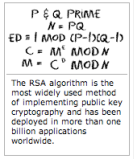
Exploiting intractability

Challenge. Factor this number.

7403756347956171282804679609742957314259318888923128908493623263897276503402826627689199641962511784399589433050212758537011896809828673317327310893090055250511687706329907239638076710086096962537934650563796359

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-conomy 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 !

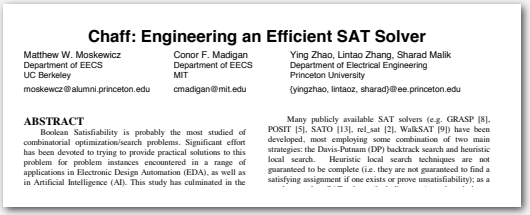
Coping with intractability

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 ~ 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

Combinatorial search

Exhaustive search. Iterate through all elements of a search space.

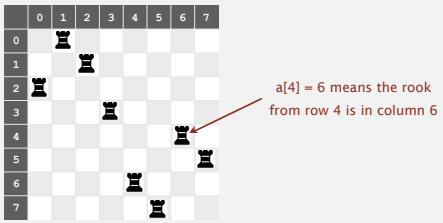
Applicability. Huge range of problems (include intractable ones).

Caveat. Search space is typically exponential in size $\Rightarrow$ effectiveness may be limited to relatively small instances.

Backtracking. Systematic method for examining feasible solutions to a problem, by systematically pruning infeasible ones.

N-rooks problem

Q. How many ways are there to place N rooks on an N-by-N board so that no rook can attack any other?




```
int[] a = { 2, 0, 1, 3, 6, 7, 4, 5 };
```

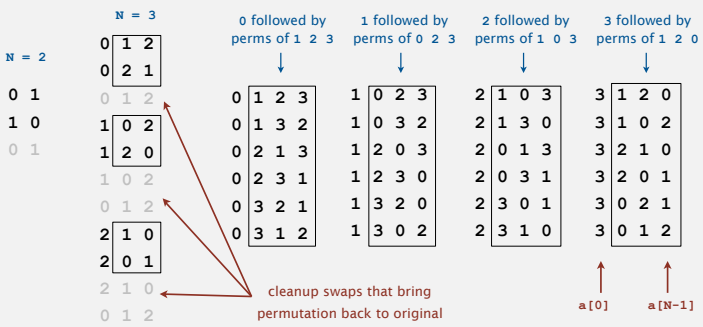
Representation. No two rooks in the same row or column $\Rightarrow$ permutation.

Challenge. Enumerate all $N!$ permutations of N integers 0 to $N - 1$.

Enumerating permutations

Recursive algorithm to enumerate all $N!$ permutations of N elements.

- Start with permutation `a[0]` to `a[N-1]`.
- For each value of `i`:
  - swap `a[i]` into position 0
  - enumerate all $(N-1)!$ permutations of `a[1]` to `a[N-1]`
  - clean up (swap `a[i]` back to original position)



Enumerating permutations

Recursive algorithm to enumerate all $N!$ permutations of N elements.

- Start with permutation `a[0]` to `a[N-1]`.
- For each value of `i`:
  - swap `a[i]` into position 0
  - enumerate all $(N-1)!$ permutations of `a[1]` to `a[N-1]`
  - clean up (swap `a[i]` back to original position)

```
// place N-k rooks in a[k] to a[N-1]
private void enumerate(int k)
{
   if (k == N)
   {  process(); return;  }

   for (int i = k; i < N; i++)
   {
      exch(k, i);
      enumerate(k+1);
      exch(i, k);          ← clean up
   }
}
```

```
% java Rooks 4
0 1 2 3
0 1 3 2
0 2 1 3
0 2 3 1
0 3 2 1
0 3 1 2
1 0 2 3
1 0 3 2
1 2 0 3
1 2 3 0
1 3 2 0
1 3 0 2
2 1 0 3
2 1 3 0
2 0 1 3
2 0 3 1
2 3 0 1
2 3 1 0
3 1 2 0
3 1 0 2
3 2 1 0
3 2 0 1
3 0 2 1
3 0 1 2
```

0 followed by perms of 1 2 3

1 followed by perms of 0 2 3

2 followed by perms of 1 0 3

3 followed by perms of 1 2 0

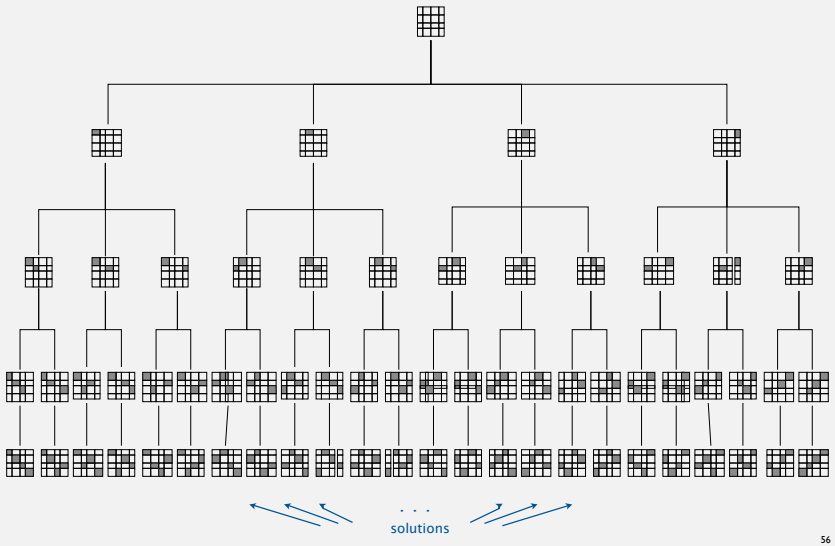
a[0] a[N-1]

Enumerating permutations

```
public class Rooks
{
   private int N;
   private int[] a;  // bits (0 or 1)

   public Rooks(int N)
   {
      this.N = N;
      a = new int[N];
      for (int i = 0; i < N; i++)
         a[i] = i;          ← initial permutation
      enumerate(0);
   }

   private void enumerate(int k)
   { /* see previous slide */  }

   private void exch(int i, int j)
   {  int t = a[i]; a[i] = a[j]; a[j] = t;  }

   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      new Rooks(N);
   }
}
```

```
% java Rooks 2
0 1
1 0

% java Rooks 3
0 1 2
0 2 1
1 0 2
1 2 0
2 1 0
2 0 1
```


4-rooks search tree



N-rooks problem: back-of-envelope running time estimate

Slow way to compute $N!$.

```
% java Rooks 7 | wc -l
5040

% java Rooks 8 | wc -l
40320

% java Rooks 9 | wc -l
362880

% java Rooks 10 | wc -l
3628800

% java Rooks 25 | wc -l
...
```

← instant

← 1.6 seconds

← 15 seconds

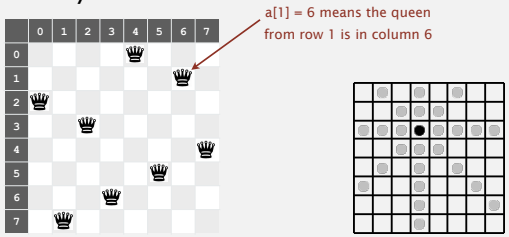
← 170 seconds

← forever

Hypothesis. Running time is about $2\,(N!\ /\ 8!\,)$ seconds.

N-queens problem

Q. How many ways are there to place N queens on an N-by-N board so that no queen can attack any other?




```
int[] a = { 2, 7, 3, 6, 0, 5, 1, 4 };
```

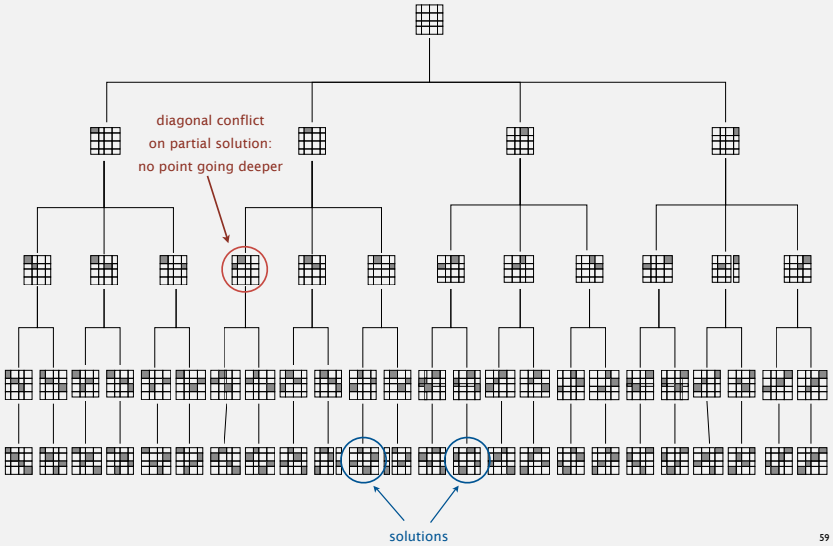
Representation. No two queens in the same row or column ⇒ permutation.

Additional constraint. No diagonal attack is possible.

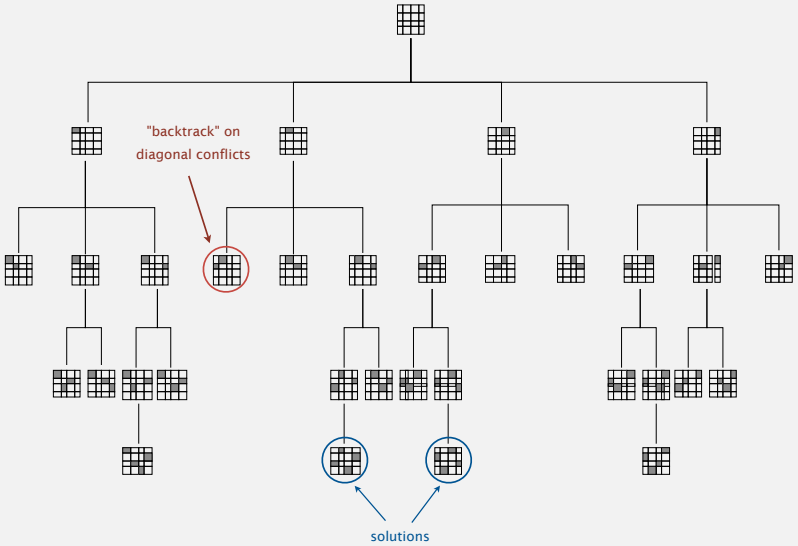
← unlike N-rooks problem, nobody knows answer for N > 30

Challenge. Enumerate (or even count) the solutions.

4-queens search tree



4-queens search tree (pruned)



Backtracking

Backtracking paradigm. Iterate through elements of search space.

- When there are several possible choices, make one choice and recur.
- If the choice is a dead end, backtrack to previous choice, and make next available choice.

Benefit. Identifying dead ends allows us to prune the search tree.

Ex. [backtracking for N-queens problem]

- Dead end: a diagonal conflict.
- Pruning: backtrack and try next column when diagonal conflict found.

Applications. Puzzles, combinatorial optimization, parsing, ...

N-queens problem: backtracking solution

```
private boolean canBacktrack(int k)
{
   for (int i = 0; i < k; i++)
   {
      if ((a[i] - a[k]) == (k - i)) return true;
      if ((a[k] - a[i]) == (k - i)) return true;
   }
   return false;
}

// place N-k queens in a[k] to a[N-1]
private void enumerate(int k)
{
   if (k == N)
   {  process(); return;  }

   for (int i = k; i < N; i++)
   {
      exch(k, i);
      if (!canBacktrack(k)) enumerate(k+1);
      exch(i, k);
   }
}
```

stop enumerating if adding queen k leads to a diagonal violation

```
% java Queens 4
1 3 0 2
2 0 3 1

% java Queens 5
0 2 4 1 3
0 3 1 4 2
1 3 0 2 4
1 4 2 0 3
2 0 3 1 4
2 4 1 3 0
3 1 4 2 0
3 0 2 4 1
4 1 3 0 2
4 2 0 3 1

% java Queens 6
1 3 5 0 2 4
2 5 1 4 0 3
3 0 4 1 5 2
4 2 0 5 3 1
```

a[0] a[N-1]

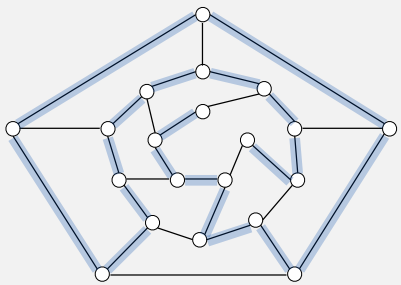
N-queens problem: effectiveness of backtracking

Pruning the search tree leads to enormous time savings.

N	Q(N)	N
2	0	2
3	0	6
4	2	24
5	10	120
6	4	720
7	40	5,040
8	92	40,320
9	352	362,880
10	724	3,628,800
11	2,680	39,916,800
12	14,200	479,001,600
13	73,712	6,227,020,800
14	365,596	87,178,291,200

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: backtracking solution

Backtracking solution. To find Hamilton path starting at v :

- Add v to current path.
- For each vertex w adjacent to v
  - find a simple path starting at w using all remaining vertices
- Clean up: remove v from current path.

Q. How to implement?

A. Add cleanup to DFS (!!)

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