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.
• 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.

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

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.
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…

\[
\begin{array}{|c|c|}
\hline
\text{quantity} & \text{value} \\
\hline
\text{electrons in universe} & 10 \\
\text{supercomputer instructions per second} & 10 \\
\text{age of universe in seconds} & 10 \\
\hline
\end{array}
\]

† 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.
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?

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.

\[
\begin{align*}
0x_0 + 1x_1 + 1x_2 &= 4 \\
2x_0 + 4x_1 - 2x_2 &= 2 \\
0x_0 + 3x_1 + 15x_2 &= 36
\end{align*}
\]

\[
\begin{align*}
x_0 &= -1 \\
x_1 &= 2 \\
x_2 &= 2
\end{align*}
\]

variables are real numbers

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

\[
\begin{align*}
48x_0 + 16x_1 + 119x_2 &\leq 88 \\
5x_0 + 4x_1 + 35x_2 &\geq 13 \\
15x_0 + 4x_1 + 20x_2 &\geq 23 \\
x_0 , x_1 , x_2 &\geq 0
\end{align*}
\]

\[
\begin{align*}
x_0 &= 1 \\
x_1 &= 1 \\
x_2 &= 1/3
\end{align*}
\]

variables are real numbers

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

\[
\begin{align*}
x_1 + x_2 &\geq 1 \\
x_0 + x_2 &\geq 1 \\
x_0 + x_1 + x_2 &\leq 2
\end{align*}
\]

\[
\begin{align*}
x_0 &= 0 \\
x_1 &= 1 \\
x_2 &= 1
\end{align*}
\]

variables are 0 or 1

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

\[
\begin{align*}
(x'_1 \text{ or } x'_2) \text{ and } (x_0 \text{ or } x_2) &= \text{true} \\
(x_0 \text{ or } x_1) \text{ and } (x_1 \text{ or } x'_2) &= \text{false} \\
(x_0 \text{ or } x_2) \text{ and } (x'_0) &= \text{true}
\end{align*}
\]

\[
\begin{align*}
x_0 &= \text{false} \\
x_1 &= \text{false} \\
x_2 &= \text{true}
\end{align*}
\]

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


Requirement. Must be able to efficiently check that $S$ is a solution.

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

\[
\begin{align*}
0x_0 + 1x_1 + 1x_2 &= 4 \\
2x_0 + 4x_1 - 2x_2 &= 2 \\
0x_0 + 3x_1 + 15x_2 &= 36
\end{align*}
\]

instance $I$

\[
\begin{align*}
x_0 &= -1 \\
x_1 &= 2 \\
x_2 &= 2
\end{align*}
\]

solution $S$

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


Requirement. Must be able to efficiently check that $S$ is a solution.

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

\[
\begin{align*}
48x_0 & + 16x_1 + 119x_2 & \leq 88 \\
5x_0 & + 4x_1 + 35x_2 & \geq 13 \\
15x_0 & + 4x_1 + 20x_2 & \geq 23 \\
x_0, \ x_1, \ x_2 & \geq 0
\end{align*}
\]

instance I

\[
\begin{align*}
x_0 &= 1 \\
x_1 &= 1 \\
x_2 &= \frac{1}{5}
\end{align*}
\]

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.

\[
\begin{align*}
    x_1 + x_2 &\geq 1 \\
    x_0 + x_2 &\geq 1 \\
    x_0 + x_1 + x_2 &\leq 2
\end{align*}
\]

\[
\begin{align*}
    x_0 &= 0 \\
    x_1 &= 1 \\
    x_2 &= 1
\end{align*}
\]

instance $I$  

solution $S$

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


Requirement. Must be able to efficiently check that $S$ is a solution.

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

\[
\begin{align*}
(x'_1 \text{ or } x'_2) \text{ and } (x_0 \text{ or } x_2) &= \text{true} \\
(x_0 \text{ or } x_1) \text{ and } (x_1 \text{ or } x'_2) &= \text{false} \\
(x_0 \text{ or } x_2) \text{ and } (x'_0) &= \text{true}
\end{align*}
\]

instance $I$ solution $S$

\[
\begin{align*}
x_0 &= \text{false} \\
x_1 &= \text{false} \\
x_2 &= \text{true}
\end{align*}
\]

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


Requirement. Must be able to efficiently check that $S$ is a solution.

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

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

- Search problems
- P vs. NP
- Classifying problems
- NP-completeness
### Def. NP is the class of all search problems.

<table>
<thead>
<tr>
<th>problem</th>
<th>description</th>
<th>poly-time algorithm</th>
<th>instance I</th>
<th>solution S</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSOLVE</td>
<td>Find a vector $x$ that satisfies $Ax = b$</td>
<td>Gaussian elimination</td>
<td>0$x_0$ + 1$x_1$ + 1$x_2$ = 4 &lt;br&gt; 2$x_0$ + 4$x_1$ - 2$x_2$ = 2 &lt;br&gt; 0$x_0$ + 3$x_1$ + 15$x_2$ = 36</td>
<td>$x_0 = -1$&lt;br&gt;$x_1 = 2$&lt;br&gt;$x_2 = 2$</td>
</tr>
<tr>
<td>LP</td>
<td>Find a vector $x$ that satisfies $Ax \leq b$</td>
<td>ellipsoid</td>
<td>48$x_0$ + 16$x_1$ + 119$x_2$ $\leq$ 88 &lt;br&gt; 5$x_0$ + 4$x_1$ + 35$x_2$ $\geq$ 13 &lt;br&gt; 15$x_0$ + 4$x_1$ + 20$x_2$ $\geq$ 23 &lt;br&gt; $x_0 , x_1 , x_2 \geq 0$</td>
<td>$x_0 = 1$&lt;br&gt;$x_1 = 1$&lt;br&gt;$x_2 = \frac{1}{2}$</td>
</tr>
<tr>
<td>ILP</td>
<td>Find a binary vector $x$ that satisfies $Ax \leq b$</td>
<td>???</td>
<td>$x_1 + x_2 \geq 1$ &lt;br&gt; $x_0 + x_2 \geq 1$ &lt;br&gt; $x_0 + x_1 + x_2 \leq 2$</td>
<td>$x_0 = 0$&lt;br&gt;$x_1 = 1$&lt;br&gt;$x_2 = 1$</td>
</tr>
<tr>
<td>SAT</td>
<td>Find a boolean vector $x$ that satisfies $\Phi(x) = b$</td>
<td>???</td>
<td>$(x_1 \text{ or } x_2) \text{ and } (x_0 \text{ or } x_2) = \text{ true}$ &lt;br&gt; $(x_0 \text{ or } x_1) \text{ and } (x_1 \text{ or } x_2) = \text{ false}$ &lt;br&gt; $(x_0 \text{ or } x_2) \text{ and } (x_0 \text{'} \text{ or } x_2) = \text{ true}$</td>
<td>$x_0 = \text{ false}$&lt;br&gt;$x_1 = \text{ false}$&lt;br&gt;$x_2 = \text{ true}$</td>
</tr>
<tr>
<td>FACTOR</td>
<td>Find a nontrivial factor of the integer $x$</td>
<td>???</td>
<td>147573952589676412927</td>
<td>193707721</td>
</tr>
</tbody>
</table>

### Significance. What scientists and engineers aspire to compute feasibly.
**Def.** P is the class of search problems solvable in poly-time.

Note: classic definition limits P to yes-no problems

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

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

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

**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.

\[
\begin{align*}
    x_1 + x_2 & \geq 1 \\
    x_0 + x_2 & \geq 1 \\
    x_0 + x_1 + x_2 & \leq 2
\end{align*}
\]

\[
\begin{align*}
    x_0 &= 0 \\
    x_1 &= 1 \\
    x_2 &= 1
\end{align*}
\]

**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.

Recall NFA implementation.
Extended Church-Turing thesis

\[
P = \text{search problems solvable in poly-time in the natural world.}
\]

Evidence supporting thesis. True for all physical computers.

Natural computers? No successful attempts (yet).

- Ex. Computing Steiner trees with soap bubbles

  
  |\begin{tikzpicture}
  |  \draw (0,0) -- (1,1) -- (2,0) -- (1,-1) -- cycle;
  |  \fill (0,0) circle (0.1cm);
  |  \fill (1,1) circle (0.1cm);
  |  \fill (2,0) circle (0.1cm);
  |  \fill (1,-1) circle (0.1cm);
  |\end{tikzpicture}|

  - 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.
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.

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 $\text{P} = \text{NP}$?  [Can you always avoid brute-force searching and do better]

**Millennium prize.** $1$ million for resolution of $\text{P} = \text{NP}$ 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.

\[
\begin{align*}
x'_1 \text{ or } x_2 \text{ or } x_3 &= true \\
x_1 \text{ or } x'_2 \text{ or } x_3 &= true \\
x'_1 \text{ or } x'_2 \text{ or } x'_3 &= true \\
x'_1 \text{ or } x'_2 \text{ or } x_4 &= true \\
x'_1 \text{ or } x'_2 \text{ or } x_4 &= true
\end{align*}
\]

**Key applications.**

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

**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.

\[
\begin{align*}
x_1' \lor x_2 \lor x_3 &= true \\
x_1 \lor x_2' \lor x_3 &= true \\
x_1' \lor x_2' \lor x_3' &= true \\
x_1' \lor x_2' \lor x_4 &= true
\end{align*}
\]

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

\[
\begin{align*}
1 &\leq (1 - x_1) + x_2 + x_3 \\
1 &\leq x_1 + (1 - x_2) + x_3 \\
1 &\leq (1 - x_1) + (1 - x_2) + (1 - x_3) \\
1 &\leq (1 - x_1) + (1 - x_2) + x_4
\end{align*}
\]

solution to this ILP instance gives solution to original SAT instance

can to reduce any SAT problem to this form
More poly-time reductions from boolean satisfiability

SAT

3-COLOR

IND-SET

SAT reduces to ILP

EXACT COVER

SAT reduces to ILP

SUBSET-SUM

VERTEX COVER

CLIQUE

HAM-CYCLE

TSP

HAM-PATH

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.


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 Sudoku, 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.

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

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 ≠ 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. \([\text{poly-time}]\)
- To break: factor a 2 \( n \)-bit integer. \([\text{unlikely \ poly-time}]\)

Multiply = EASY

\[23 \times 67 \rightarrow 1,541\]

Factor = HARD
Exploiting intractability

**Challenge.** Factor this number.

\[
\begin{align*}
740375634795617128280467960974295731425931888892312890849362 \\
326389727650340282862768919964196251178439958943305021275853 \\
701189680982867331732731089309005525051168770632990723963807 \\
86710086096962537934650563796359
\end{align*}
\]

RSA-704
($30,000 prize if you can factor)

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

The RSA algorithm is the most widely used method of implementing public key cryptography and has been deployed in more than one billion applications worldwide.

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 \( \sim 10^4 \) 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 development of Chaff, an efficient SAT solver. Chaff is an implementation of the conflict-driven clause learning (CDCL) method, which uses an in-process decision graph to backtrack and prune large fractions of the search space. A novel feature of Chaff is its efficient implementation of the conflict-directed backjumping technique. Chaff is the first available implementation of this approach that is fast enough to be useful in practice. The technique modulates the number of backtracks based on the number of conflict clauses in the current decision graph. This algorithm has been implemented in a manner that is able to run on a wide variety of computer architectures.

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 result, these techniques are usually employed in conjunction with the DP search. Chaff employs both strategies, but it attempts to optimize the tradeoff between the two by using a simple mechanism to avoid conflicts.

---

50
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 ⇒ 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)

<table>
<thead>
<tr>
<th>$N = 2$</th>
<th>$N = 3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2</td>
<td>0 1 2 3</td>
</tr>
<tr>
<td>0 2 1</td>
<td>1 0 2 3</td>
</tr>
<tr>
<td>1 0 2</td>
<td>1 2 0 3</td>
</tr>
<tr>
<td>1 2 0</td>
<td>1 2 3 0</td>
</tr>
<tr>
<td>0 1 2</td>
<td>1 3 2 0</td>
</tr>
<tr>
<td>0 1 2</td>
<td>1 3 0 2</td>
</tr>
<tr>
<td>2 1 0</td>
<td>2 1 0 3</td>
</tr>
<tr>
<td>2 0 1</td>
<td>2 0 3 1</td>
</tr>
<tr>
<td>2 1 0</td>
<td>2 3 0 1</td>
</tr>
<tr>
<td>2 0 1</td>
<td>2 3 1 0</td>
</tr>
<tr>
<td>2 1 0</td>
<td>3 1 2 0</td>
</tr>
<tr>
<td>2 0 1</td>
<td>3 0 2 1</td>
</tr>
<tr>
<td>0 1 2</td>
<td>3 0 1 2</td>
</tr>
</tbody>
</table>

Cleanup swaps that bring permutation back to original

$N = 2$

$N = 3$
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)

```java
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
    }
}
```
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;
        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);
    }
}
4-rooks search tree
N-rooks problem: back-of-envelope running time estimate

Slow way to compute $N!$.

Hypothesis. Running time is about $2 \left( \frac{N!}{8!} \right)$ seconds.
Q. How many ways are there to place $N$ queens on an $N$-by-$N$ board so that no queen can attack any other?

Representation. No two queens in the same row or column \( \Rightarrow \) permutation.

Additional constraint. No diagonal attack is possible.

Challenge. Enumerate (or even count) the solutions.

Unlike N-rooks problem, nobody knows answer for $N > 30$.
4-queens search tree

diagonal conflict on partial solution: no point going deeper

solutions
4-queens search tree (pruned)

"backtrack" on diagonal conflicts

solutions
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);
    }
}
Pruning the search tree leads to enormous time savings.

<table>
<thead>
<tr>
<th>N</th>
<th>Q(N)</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>24</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>120</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>720</td>
</tr>
<tr>
<td>7</td>
<td>40</td>
<td>5,040</td>
</tr>
<tr>
<td>8</td>
<td>92</td>
<td>40,320</td>
</tr>
<tr>
<td>9</td>
<td>352</td>
<td>362,880</td>
</tr>
<tr>
<td>10</td>
<td>724</td>
<td>3,628,800</td>
</tr>
<tr>
<td>11</td>
<td>2,680</td>
<td>39,916,800</td>
</tr>
<tr>
<td>12</td>
<td>14,200</td>
<td>479,001,600</td>
</tr>
<tr>
<td>13</td>
<td>73,712</td>
<td>6,227,020,800</td>
</tr>
<tr>
<td>14</td>
<td>365,596</td>
<td>87,178,291,200</td>
</tr>
</tbody>
</table>
**Goal.** Find a simple path that visits every vertex 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 (!!!)
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;
    }
}