**Intractability**

Given: A set of N cities and $M for gas.

Problem: Does a traveling salesperson have enough $ for gas to visit all the cities?

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

A difficult problem

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


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

Ex 1. Sorting N elements

**Insertion sort** takes less than $aN^2$ steps for all inputs.

Ex 2. TSP on N cities

**Exhaustive search** could take $aN!$ steps.

In theory: Definition is broad and robust (since a and b tend to be small).

In practice: Poly-time algorithms tend to scale to handle large problems.

**Exponential Growth**

Exponential growth dwarfs technological change.

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

<table>
<thead>
<tr>
<th>quantity</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>electrons in universe †</td>
<td>$10^{79}$</td>
</tr>
<tr>
<td>supercomputer instructions per second</td>
<td>$10^{13}$</td>
</tr>
<tr>
<td>age of universe in seconds ‡</td>
<td>$10^{17}$</td>
</tr>
</tbody>
</table>

† estimated

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

$1000! \gg 10^{1000} \gg 10^{79} \times 10^{13} \times 10^{17}$
**Reasonable Questions about Problems**

**Q.** Which problems can we solve in practice?

**A.** Those with easy-to-find answers or with guaranteed poly-time algorithms.

**Q.** Which problems have guaranteed poly-time algorithms?

**A.** Not so easy to know. Focus of today's lecture.

**Four Fundamental Problems**

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

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

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

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

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

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

\[
\begin{align*}
x_1 + x_2 &\leq 1 \\
x_3 + x_2 &\leq 1 \\
x_0 + x_1 + x_2 &\leq 2
\end{align*}
\]

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

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

\[
\begin{align*}
(x_0 \text{ and } x_1 \text{ and } x_2) &\text{ or } (x_1 \text{ and } x_2) &\text{ or } (x_0 \text{ and } x_2) = \text{true} \\
(x_0 \text{ and } x_1) &\text{ or } (x_1 \text{ and } x_2) &\text{ or } (x_2 \text{ and } x_3) = \text{false} \\
(x_1 \text{ and } x_2) &\text{ or } (x_0 \text{ and } x_2) &\text{ or } (x_0) = \text{true}
\end{align*}
\]

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

**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.

\[
\text{poly-time in size of instance } I
\]
**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.

<table>
<thead>
<tr>
<th>Instance $I$</th>
<th>Solution $S$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0x_0 + 1x_1 + 1x_2 = 4$</td>
<td>$x_0 = -1$</td>
</tr>
<tr>
<td>$2x_0 + 4x_1 - 2x_2 = 2$</td>
<td>$x_1 = 2$</td>
</tr>
<tr>
<td>$0x_0 + 3x_1 + 5x_2 = 36$</td>
<td>$x_2 = 2$</td>
</tr>
</tbody>
</table>

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

### LP

Given a system of linear inequalities, find a solution.

<table>
<thead>
<tr>
<th>Instance $I$</th>
<th>Solution $S$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$-4x_0 + 16x_1 + 119x_2 \leq 88$</td>
<td>$x_0 = 1$</td>
</tr>
<tr>
<td>$5x_0 + 4x_1 + 35x_2 \geq 13$</td>
<td>$x_1 = 1$</td>
</tr>
<tr>
<td>$15x_0 + 4x_1 + 20x_2 \geq 23$</td>
<td>$x_2 = \frac{3}{2}$</td>
</tr>
</tbody>
</table>

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

### ILP

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

<table>
<thead>
<tr>
<th>Instance $I$</th>
<th>Solution $S$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1 + x_2 \geq 1$</td>
<td>$x_0 = 0$</td>
</tr>
<tr>
<td>$x_0 + x_2 \geq 1$</td>
<td>$x_1 = 1$</td>
</tr>
<tr>
<td>$x_0 + x_1 + x_2 \leq 2$</td>
<td>$x_2 = 1$</td>
</tr>
</tbody>
</table>

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

### SAT

Given a system of boolean equations, find a solution.

<table>
<thead>
<tr>
<th>Instance $I$</th>
<th>Solution $S$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(x_0, x_1, x_2)$ or $(x_1, x_2)$ or $(x_0, x_1)$</td>
<td>$(x_0 = false)$</td>
</tr>
<tr>
<td>$(x_0, x_1)$ or $(x_1, x_2)$ or $(x_0, x_2)$</td>
<td>$(x_1 = true)$</td>
</tr>
<tr>
<td>$(x_0, x_1, x_2)$ or $(x_1, x_2)$</td>
<td>$(x_2 = true)$</td>
</tr>
</tbody>
</table>

- 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.

poly-time in size of instance \( I \)

FACTOR. Find a nontrivial factor of the integer \( x \).

\[ 2^{67} - 1 = 14757952589676412927 \]

\[ 193707721 \]

instance \( I \) solution \( S \)

- To check solution \( S \), long divide 193707721 into 14757952589676412927.

Def. \( P \) is the class of search problems solvable in poly-time.

A search problem that is not in \( P \) is said to be intractable.

<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>STCONN ((G, s, t))</td>
<td>Find a path from ( s ) to ( t ) in digraph ( G ).</td>
<td>depth-first search (Theex)</td>
<td>15, 14, 12, 13, 15</td>
<td>13, 14, 15, 16, 15</td>
</tr>
<tr>
<td>SORT ((a))</td>
<td>Find permutation that puts ( a ) in ascending order.</td>
<td>mergesort (von Neumann 1945)</td>
<td>8, 5, 1, 2, 3</td>
<td>1, 2, 3, 4, 5</td>
</tr>
<tr>
<td>LSOLVE ((A, b))</td>
<td>Find a vector ( x ) that satisfies ( Ax = b ).</td>
<td>Gaussian elimination (Edmonds, 1967)</td>
<td>8, 4, 2, 1, 3</td>
<td>1, 1, 1, 1, 1</td>
</tr>
<tr>
<td>LP ((A, b))</td>
<td>Find a vector ( x ) that satisfies ( Ax \leq b ).</td>
<td>ellipsoid (Khachiyan, 1979)</td>
<td>3, 4, 5, 6, 7</td>
<td>1, 1, 1, 1, 1</td>
</tr>
</tbody>
</table>

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

Def. \( NP \) is the class of all search problems — problems with poly-time checkable solutions.

<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 ((A, b))</td>
<td>Find a vector ( x ) that satisfies ( Ax = b ).</td>
<td>Gaussian elimination</td>
<td>0, 3, 1, 2, 3</td>
<td>1, 1, 1, 1, 1</td>
</tr>
<tr>
<td>LP ((A, b))</td>
<td>Find a vector ( x ) that satisfies ( Ax \leq b ).</td>
<td>ellipsoid</td>
<td>0, 3, 1, 2, 3</td>
<td>1, 1, 1, 1, 1</td>
</tr>
</tbody>
</table>

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

Other types of problems

Search problem. Find a solution.

Decision problem. Is there a solution?

Optimization problem. Find the best solution.

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

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

Note: Standard definitions of \( P \) and \( NP \) are in terms of decision problems.
Nondeterminism

Nondeterministic machine can **guess** the desired solution

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

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

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

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

---

Extended Church-Turing Thesis

**Extended Church-Turing thesis.**

\[
P = \text{search problems solvable in poly-time in this universe.}
\]

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

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

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

---

**The Central Question**

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

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

Two possible universes.

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

**Overwhelming consensus.** \( P \neq NP \).
Classifying Problems

Q. Which search problems are in P?
Q. Which search problems are not in P (intractable)?
A. No easy answers (we don’t even know whether P = NP).

First step. Formalize notion:

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

Exhaustive Search

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

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

SAT is intractable

Reductions

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

To solve X, use:
- a poly number of standard computational steps, plus
- a poly number of calls to a method that solves instances of Y.
**Def.** Problem \( X \) reduces to problem \( Y \) if you can solve \( X \) given:
- A poly number of standard computational steps, plus
- A poly number of calls to a subroutine for solving instances of \( Y \).

**Problem**

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

**Design algorithms.** If poly-time algorithm for \( Y \), then one for \( X \) too.

**Establish intractability.** If no poly-time algorithm for \( X \), then none for \( Y \).

---

**SAT Reduces to ILP**

**SAT.** Given a boolean equation \( \Phi \), find a satisfying truth assignment.

\[ \begin{align*}
\Phi &= (x_1 \text{ or } x_2 \text{ or } x_3) \text{ and } (x_1 \text{ or } x_2' \text{ or } x_4) \text{ and } (x_1' \text{ or } x_2' \text{ or } x_4') \\
\end{align*} \]

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

\[ \begin{align*}
C_1 &= 1 - x_1 \\
C_2 &= x_2 \\
C_3 &= x_3 \\
C_4 &= (1 - x_1 + x_2 + x_3) \\
C_5 &= 1 \text{ iff clause 1 is satisfied} \\
\Phi &= C_1 \\
\Phi &= C_2 \\
\Phi &= C_3 \\
\Phi &= C_4 \\
\Phi &\geq C_1 + C_2 + C_3 + C_4 - 3 \\
\Phi &= 1 \text{ iff } C_1 = C_2 = C_3 = C_4 = 1
\end{align*} \]

**More Reductions From SAT**

Dick Karp

'85 Turing award

Conjecture: SAT is intractable.
Implication: all of these problems are intractable.
**NP-Completeness**

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

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

**Theorem.** [Cook 1971] SAT is NP-complete.

Extremely brief Proof Sketch:
- convert non-deterministic TM notation to SAT notation
- if you can solve SAT, you can solve any problem in NP

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

**Cook's Theorem**

- All NP problems reduce to SAT.
Implications of NP-completeness

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

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

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

Coping With Intractability

Two possible universes

$P \neq NP$:
- Intractable search problems exist.
- Nondeterminism makes machines more efficient.
- Can prove that a problem is intractable by reduction from an NP-complete problem.
- Some search problems are neither NP-complete or in P.
- Some search problems are still not classified.

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

[Third possibility: Extended Church-Turing thesis is wrong.]
You have an NP-complete problem.
• It's safe to assume that it is intractable.
• What to do?

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

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

Chaff solves real-world SAT instances with ~10k variables.

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 87.5% as many clauses as possible in poly-time, then P = NP !
Exploiting Intractability: Cryptography

**FACTOR.** Given an $n$-bit integer $x$, find a nontrivial factor. Not 1 or $x$.

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

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

---

**Summary**

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

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

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