Intractability
Traveling salesperson problem (TSP)

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

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

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

Q. Which algorithms are useful in practice?


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

Ex 1. Sorting N elements

Insertion sort takes less than \(aN^2\) steps for all inputs. **efficient**

Ex 2. TSP on N cities

Exhaustive search could take \(aN!\) steps. **not efficient**

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

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

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

many known poly-time algorithms for sorting
no known poly-time algorithm for TSP
Four Fundamental Problems

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

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

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

Variables are real numbers.

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

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

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

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 solution.

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

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

Variables are "true" or "false".

---

---
Four Fundamental Problems

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

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

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

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

Q. Which of these problems have guaranteed poly-time solutions?

A. No easy answers.

✓ **LSOLVE.** Yes. Gaussian elimination solves $n$-by-$n$ system in $n^3$ time.

✓ **LP.** Yes. Ellipsoid algorithm is poly-time.

? **ILP, SAT.** No poly-time algorithm known or believed to exist!
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.

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

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

instance $I$  
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.

! or report none exists

poly-time in size of instance $I$

---

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

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

instance $I$

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

solution $S$

• To check solution $S$, check that values are 0/1, then 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 solution.*

- $(x_0 \text{ and } x_1 \text{ and } x_2) \text{ or } (x_1 \text{ and } x_2) \text{ or } (x_0 \text{ and } x_2) = \text{true} \quad (x_0 \text{ or } x_1) \text{ or } (x_1 \text{ and } x_2) = \text{false} \quad (x_1 \text{ and } x_2) \text{ or } (x_0 \text{ and } x_2) \text{ or } (x_0) = \text{true}$

- $x_0 = \text{false}$
- $x_1 = \text{true}$
- $x_2 = \text{true}$

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

- $2^{67} - 1$
  
  \[
  147573952589676412927 \quad \text{or report none exists}
  \]

  poly-time in size of instance $I$

  \[
  193707721
  \]

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

Mersenne

Cole
**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</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></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>$48x_0 + 16x_1 + 119x_2 \leq 88$</td>
<td>$x_0 = 1$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$5x_0 + 4x_1 + 35x_2 \geq 13$</td>
<td>$x_1 = 1$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$15x_0 + 4x_1 + 20x_2 \geq 23$</td>
<td>$x_2 = \frac{1}{2}$</td>
</tr>
<tr>
<td>ILP</td>
<td>Find a binary vector $x$ that satisfies $Ax \leq b$.</td>
<td>???</td>
<td>$x_1 + x_2 \geq 1$</td>
<td>$x_0 = 0$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$x_0 + x_2 \geq 1$</td>
<td>$x_1 = 1$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$x_0 + x_1 + x_2 \leq 2$</td>
<td>$x_2 = 1$</td>
</tr>
<tr>
<td>SAT</td>
<td>Find a boolean vector $x$ that satisfies $Ax = b$.</td>
<td>???</td>
<td>$(x_1 \text{ and } x_2) \text{ or } (x_0 \text{ and } x_2) = \text{true}$</td>
<td>$x_0 = \text{false}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$(x_0 \text{ and } x_1) \text{ or } (x_1 \text{ and } x_2) = \text{false}$</td>
<td>$x_1 = \text{true}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$(x_0 \text{ and } x_2) \text{ or } (x_0) = \text{true}$</td>
<td>$x_2 = \text{true}$</td>
</tr>
<tr>
<td>FACTOR</td>
<td>Find a nontrivial factor of the integer $x$.</td>
<td>???</td>
<td>8784561</td>
<td>10657</td>
</tr>
</tbody>
</table>

**Significance.** What scientists, engineers, and applications programmers aspire to compute feasibly.
**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><strong>STCONN</strong></td>
<td>Find a path from $s$ to $t$ in digraph $G$.</td>
<td>depth-first search</td>
<td><img src="image" alt="diagram" /></td>
<td><img src="image" alt="diagram" /></td>
</tr>
<tr>
<td>$(G, s, t)$</td>
<td></td>
<td>(Theseus)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SORT</strong></td>
<td>Find permutation that puts $a$ in ascending order.</td>
<td>mergesort</td>
<td>$2.3\ 8.5\ 1.2\ 9.1\ 2.2\ 0.3$</td>
<td>$5\ 2\ 4\ 0\ 1\ 3$</td>
</tr>
<tr>
<td>$(a)$</td>
<td></td>
<td>(von Neumann 1945)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>LSOLVE</strong></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>$(A, b)$</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><strong>LP</strong></td>
<td>Find a vector $x$ that satisfies $Ax \leq b$.</td>
<td>ellipsoid</td>
<td>$48x_0 + 16x_1 + 119x_2 \leq 88$</td>
<td>$x_0 = 1$</td>
</tr>
<tr>
<td>$(A, b)$</td>
<td></td>
<td>(Khachiyan, 1979)</td>
<td>$5x_0 + 4x_1 + 35x_2 \geq 13$</td>
<td>$x_1 = 1$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$15x_0 + 4x_1 + 20x_2 \geq 23$</td>
<td>$x_2 = \frac{1}{3}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$x_0, \ x_1, \ x_2 \geq 0$</td>
<td></td>
</tr>
</tbody>
</table>

**Significance.** What scientists and engineers, and applications programmers do compute feasibly.
Other types of problems

Search problem. Find a solution.

Decision problem. Is there a solution?

Optimization problem. Find the best solution.

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

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

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

Nondeterministic machine can guess the desired solution

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

- Java: values are all 0
- nondeterministic machine: values are the answer!

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

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

instance $I$

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

solution $S$

Ex. Turing machine

- deterministic: state, input determines next state
- nondeterministic: more than one possible next state

NP: Search problems solvable in Poly time on a Nondeterministic machine.
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
P vs. NP
The Central Question

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

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

**Does P = NP?**

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

Two possible universes.

*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

Periodic Table of the Elements

<table>
<thead>
<tr>
<th>IA</th>
<th>IIA</th>
<th>IIIB</th>
<th>IVA</th>
<th>VA</th>
<th>VIA</th>
<th>VIIA</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>He</td>
</tr>
<tr>
<td>1</td>
<td>H</td>
<td>Li</td>
<td>Be</td>
<td>Na</td>
<td>Mg</td>
<td>Al</td>
<td>Si</td>
</tr>
<tr>
<td>2</td>
<td>Be</td>
<td>B</td>
<td>C</td>
<td>N</td>
<td>O</td>
<td>F</td>
<td>Ne</td>
</tr>
<tr>
<td>3</td>
<td>Na</td>
<td>Mg</td>
<td>Ca</td>
<td>19</td>
<td>K</td>
<td>20</td>
<td>Ca</td>
</tr>
<tr>
<td>4</td>
<td>K</td>
<td>Ca</td>
<td>Sc</td>
<td>Ti</td>
<td>V</td>
<td>Cr</td>
<td>Mn</td>
</tr>
<tr>
<td>5</td>
<td>Sc</td>
<td>Ti</td>
<td>V</td>
<td>Cr</td>
<td>Mn</td>
<td>Fe</td>
<td>Co</td>
</tr>
<tr>
<td>6</td>
<td>Ti</td>
<td>V</td>
<td>Cr</td>
<td>Mn</td>
<td>Fe</td>
<td>Co</td>
<td>Ni</td>
</tr>
<tr>
<td>7</td>
<td>Cr</td>
<td>Mn</td>
<td>Fe</td>
<td>Co</td>
<td>Ni</td>
<td>Cu</td>
<td>Zn</td>
</tr>
<tr>
<td>8</td>
<td>Mn</td>
<td>Fe</td>
<td>Co</td>
<td>Ni</td>
<td>Cu</td>
<td>Zn</td>
<td>Ga</td>
</tr>
<tr>
<td>9</td>
<td>Fe</td>
<td>Co</td>
<td>Ni</td>
<td>Cu</td>
<td>Zn</td>
<td>Ga</td>
<td>Ge</td>
</tr>
<tr>
<td>10</td>
<td>Co</td>
<td>Ni</td>
<td>Cu</td>
<td>Zn</td>
<td>Ga</td>
<td>Ge</td>
<td>As</td>
</tr>
<tr>
<td>11</td>
<td>Ni</td>
<td>Cu</td>
<td>Zn</td>
<td>Ga</td>
<td>Ge</td>
<td>As</td>
<td>Se</td>
</tr>
<tr>
<td>12</td>
<td>Cu</td>
<td>Zn</td>
<td>Ga</td>
<td>Ge</td>
<td>As</td>
<td>Se</td>
<td>Br</td>
</tr>
<tr>
<td>13</td>
<td>Zn</td>
<td>Ga</td>
<td>Ge</td>
<td>As</td>
<td>Se</td>
<td>Br</td>
<td>Kr</td>
</tr>
<tr>
<td>14</td>
<td>Ga</td>
<td>Ge</td>
<td>As</td>
<td>Se</td>
<td>Br</td>
<td>Kr</td>
<td>Ar</td>
</tr>
<tr>
<td>15</td>
<td>Ge</td>
<td>As</td>
<td>Se</td>
<td>Br</td>
<td>Kr</td>
<td>Ar</td>
<td>19</td>
</tr>
<tr>
<td>16</td>
<td>Se</td>
<td>Br</td>
<td>Kr</td>
<td>Ar</td>
<td>19</td>
<td></td>
<td>19</td>
</tr>
<tr>
<td>17</td>
<td>Br</td>
<td>Kr</td>
<td>Ar</td>
<td>19</td>
<td></td>
<td></td>
<td>Ne</td>
</tr>
<tr>
<td>18</td>
<td>Kr</td>
<td>Ar</td>
<td></td>
<td></td>
<td></td>
<td>Ne</td>
<td></td>
</tr>
</tbody>
</table>

* Lanthanide Series
+ Actinide Series

| 58 | Ce | 59 | Pr | 60 | Nd | 61 | Pm | 62 | Sm | 63 | Eu | 64 | Gd | 65 | Tb | 66 | Dy | 67 | Ho |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 58 | Ce | 59 | Pr | 60 | Nd | 61 | Pm | 62 | Sm | 63 | Eu | 64 | Gd | 65 | Tb | 66 | Dy | 67 | Ho |
| 58 | Ce | 59 | Pr | 60 | Nd | 61 | Pm | 62 | Sm | 63 | Eu | 64 | Gd | 65 | Tb | 66 | Dy | 67 | Ho |

* Lanthanide Series
+ Actinide Series

| 100 | Th | 101 | Pa | 102 | U  | 103 | Np | 104 | Pu | 105 | Am | 106 | Cm | 107 | Bk | 108 | Cf | 109 | Es |
|-----|----|-----|----|-----|----|-----|----|-----|----|-----|----|-----|----|-----|----|-----|----|-----|
| 100 | Th | 101 | Pa | 102 | U  | 103 | Np | 104 | Pu | 105 | Am | 106 | Cm | 107 | Bk | 108 | Cf | 109 | Es |
| 100 | Th | 101 | Pa | 102 | U  | 103 | Np | 104 | Pu | 105 | Am | 106 | Cm | 107 | Bk | 108 | Cf | 109 | Es |

* Lanthanide Series
+ Actinide Series
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.
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.
**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 $Y$. 

![Diagram of reductions](attachment:image.png)
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 $Y$.

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 $Y$. 

Reductions: Consequences
**LSOLVE Reduces to LP**

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

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

*LSOLVE instance with n variables*

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

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

*corresponding LP instance with n variables and 2n inequalities*
SAT Reduces to ILP

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

\[
\Phi = (x'_1 \text{ or } x_2 \text{ or } x_3) \land (x_1 \text{ or } x'_2 \text{ or } x_3) \land (x'_1 \text{ or } x'_2 \text{ or } x'_3) \land (x'_1 \text{ or } x'_2 \text{ or } x_4)
\]

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

\[
\begin{align*}
C_1 & \geq 1 - x_1 \\
C_1 & \geq x_2 \\
C_1 & \geq x_3 \\
C_1 & \leq (1 - x_1) + x_2 + x_3
\end{align*}
\]

\[
\begin{align*}
\Phi & \leq C_1 \\
\Phi & \leq C_2 \\
\Phi & \leq C_3 \\
\Phi & \leq C_4 \\
\Phi & \geq C_1 + C_2 + C_3 + C_4 - 3
\end{align*}
\]

\[C_1 = 1 \text{ iff clause 1 is satisfied} \]

\[\Phi = 1 \text{ iff } C_1 = C_2 = C_3 = C_4 = 1\]

**SAT instance with** \( n \) **variables,** \( k \) **clauses**

**corresponding ILP instance with** \( n + k + 1 \) **variables and** \( 4n + k + 1 \) **inequalities**

solution to this ILP instance gives solution to SAT instance
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 $a_1, ..., a_n$, compute $\int_0^{2\pi} \prod_{i=1}^n \cos(a_i \theta) \times \cos(a_2 \theta) \times \cdots \times \cos(a_n \theta) \, d\theta$

Mechanical engineering. Structure of turbulence in sheared flows.

Medicine. Reconstructing 3d shape from biplane angiocardiogram.

Operations research. Traveling salesperson problem, integer programming.

Physics. Partition function of 3d Ising model.

Politics. Shapley-Shubik voting power.

Pop culture. Versions of Sudoku, Checkers, Minesweeper, Tetris.

Statistics. Optimal experimental design.

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

6,000+ scientific papers per year.
NP-completeness
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

3-COLOR reduces to 3-SAT

All NP problems reduce to SAT.

Stephen Cook '82 Turing award
3-COLOR reduces to SAT
SAT reduces to 3-COLOR

All Karp problems are different manifestations of one "really hard" universal problem.
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 and all other search problems, such as factoring and graph isomorphism.

[Third possibility: Extended Church-Turing thesis is wrong.]
Implications of NP-completeness

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

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

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

Search for closed formula appears doomed since 3D-ISING is intractable if $P \neq NP$. 

Coping With Intractability
Coping With Intractability

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

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

Complexity theory deals with worst case behavior.
• Instance(s) you want to solve may have easy-to-find answer.
• Chaff solves real-world SAT instances with $\sim 10k$ variables.

[Matthew Moskewicz '00, Conor Madigan '00, Sharad Malik]

PU senior independent work (!)
Coping With Intractability

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

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

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

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

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

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

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

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

each clause has at most one un-negated literal
Exploiting Intractability: Cryptography

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

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

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

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.