Intractability

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.

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</td>
<td>10^13</td>
</tr>
<tr>
<td>instructions per second</td>
<td>10^17</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.

A difficult problem

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
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.

**Reasonable Questions about Problems**

- 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.

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

Variables are real numbers.

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

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

Variables are real numbers.

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

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

Variables are 0 or 1.

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

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

Variables are "true" or "false".

---

**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 \( I \).
Search Problems

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

\text{poly-time in size of instance $I$}

or report none exists

**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*}

$\begin{cases}
x_0 = -1 \\
x_1 = 2 \\
x_2 = 2
\end{cases}$

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

**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
\end{align*}

$\begin{cases}
x_0 = 1 \\
x_1 = 1 \\
x_2 = \frac{7}{5}
\end{cases}$

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

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

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

$\begin{cases}
x_0 = 0 \\
x_1 = 1 \\
x_2 = 1
\end{cases}$

• 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.

\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{false} \\
(x_0 \text{ and } x_1) = \text{true}
\end{align*}

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

• 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.

- poly-time in size of instance $I$

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

$147573952589676412927$ $193707721$

instance $I$ solution $S$

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

Def. $P$ is the class of all 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</td>
<td>Find a path from $u$ to $v$ in digraph $G$.</td>
<td>depth-first search (Theorell)</td>
<td>$1.3 8.5 1.2$ $9.1 2.2 0.3$</td>
<td>$5 2 4 0 1 3$</td>
</tr>
<tr>
<td>SORT</td>
<td>Find permutation that puts $a$ in ascending order.</td>
<td>mergesort (von Neumann 1945)</td>
<td>$2.3$</td>
<td>$9.1 2.2 0.3$ $5 2 4 0 1 3$</td>
</tr>
<tr>
<td>LSOLVE</td>
<td>Find a vector $x$ that satisfies $Ax = b$.</td>
<td>Gaussian elimination (Edmonds, 1967)</td>
<td>$0x_0 + 1x_1 + 3x_2 = 0$ $1x_0 + 1x_1 + 3x_2 = 0$ $2x_0 + 1x_1 + 3x_2 = 0$</td>
<td>$x_0 = 2$ $x_1 = 2$</td>
</tr>
<tr>
<td>LP</td>
<td>Find a vector $x$ that satisfies $Ax = b$.</td>
<td>ellipsoid (Khachiyan, 1979)</td>
<td>$0x_0 + 5x_1 + 10x_2 = 0$ $1x_0 + 2x_1 + 4x_2 = 0$ $2x_0 + 8x_1 + 15x_2 = 0$ $3x_0 + 7x_1 + 14x_2 = 0$</td>
<td>$x_0 = 1$ $x_1 = 1$ $x_2 = 1$</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 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>$0x_0 + 1x_1 + 3x_2 = 0$ $1x_0 + 1x_1 + 3x_2 = 0$ $2x_0 + 1x_1 + 3x_2 = 0$</td>
<td>$x_0 = -1$ $x_1 = 2$ $x_2 = 2$</td>
</tr>
<tr>
<td>LP</td>
<td>Find a vector $x$ that satisfies $Ax = b$.</td>
<td>ellipsoid</td>
<td>$0x_0 + 5x_1 + 10x_2 = 0$ $1x_0 + 2x_1 + 4x_2 = 0$ $2x_0 + 8x_1 + 15x_2 = 0$ $3x_0 + 7x_1 + 14x_2 = 0$</td>
<td>$x_0 = 1$ $x_1 = 1$ $x_2 = 1$</td>
</tr>
<tr>
<td>SAT</td>
<td>Find a boolean vector $x$ that satisfies $Ax = b$.</td>
<td></td>
<td>$0x_0 + 5x_1 + 10x_2 = 0$ $1x_0 + 2x_1 + 4x_2 = 0$ $2x_0 + 8x_1 + 15x_2 = 0$ $3x_0 + 7x_1 + 14x_2 = 0$</td>
<td>$x_0 = 1$ $x_1 = 1$ $x_2 = 1$</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.

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.

P vs. NP

Extended Church-Turing Thesis

Extended Church-Turing thesis.

P = 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 ≠ NP.
Fame and Fortune through CS

Some writers for the Simpsons and Futurama.

Classifying Problems

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.

Exhaustive Search

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.
**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 $Y$.

**LSOLVE Reduces to LP**

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

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

**LSOLVE instance with $n$ variables**

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

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

**corresponding LP instance with $n$ variables and $2n$ inequalities**

**SAT Reduces to ILP**

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

\[
\Phi = (x_1' \lor x_2' \lor x_3) \land (x_1 \lor x_2' \lor x_3) \land (x_1' \lor x_2' \lor x_4) \land (x_1' \lor x_2' \lor x_4)
\]

**SAT instance with $n$ variables, $k$ clauses**

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

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

**corresponding ILP instance with $n + k + 1$ variables and $4k + k + 1$ inequalities**

**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 $Y$.

3-SAT

your research problem

previously solved problem

your research problem

3-SAT

LSOLVE Reduces to LP

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

**ILP**. Given a system of linear inequalities, find a 0-1 solution.
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 3-SAT, you can solve any problem in NP

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

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 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.

[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]

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.
  \[\text{but if you can guarantee to satisfy 87.51% as many clauses as possible in poly-time, then } P \neq NP\]
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]

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.

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

7403756347956171282804679609742957314259318888892312890849362
3263897276503402826262768919964196251178439958943305021275853
7011896809828673317327310893090055525051168770632990723963807
86710086096962537934460583796359

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.