Lecture T4: NP-Completeness

Overview

Lecture T1, T2:
- What is an algorithm?
  - Turing machine
- Which problems can be solved on a computer?
  - not the halting problem

Lecture T3:
- Which ALGORITHMS will be useful in practice?
  - analysis of algorithms

This lecture:
- Which PROBLEMS can be solved in practice?
  - probably not TSP

Some Hard Problems

3-COLOR: Given a planar map, can it be colored using 3 colors so that no adjacent regions have the same color?

YES instance.

NO instance.
CIRCUIT-SAT: Is there a way to assign inputs to a given Boolean (combinational) circuit that makes it true?

FACTOR: Given two positive integers x and U, is there a nontrivial factor of x that is less than U?
- Factoring is at the heart of RSA encryption.

Example 1: \( x = 23,536,481,273 \), \( U = 110,000 \).
  - YES: \( x = 104,729 \times 224,737 \).

Example 2: \( x = 23,536,481,273 \), \( U = 100,000 \).
  - NO: \( 104,729 \times 224,737 \) is prime factorization of x.

Example 3: \( x = 23,536,481,277 \), \( U = 23,536,481,277 \).
  - NO: x is prime.

TSP: A travelling salesperson needs to visit N cities. Is there a route of length at most D?

Biology: protein folding.
Chemistry: chemical synthesis.
Civil engineering: equilibrium of urban traffic flow.
Finance: find minimum risk portfolio of given return.
Electrical engineering: VLSI layout.
Medicine: reconstructing 3-D shape from biplane angiogram.
Operations research: optimal resource allocation.
Physics: anti-ferromagnetic Potts model.
Politics: Shapley-Shubik voting power.
Pop culture: Minesweeper consistency, playing optimal Tetris.
Statistics: optimal experimental design.
Properties of Algorithms

A given problem can be solved by many different algorithms (TMs).
- Which ones are useful in practice?

A working definition: (Jack Edmonds, 1962)
- Efficient: polynomial time for ALL inputs.
  - mergesort requires $N \log_2 N$ steps
- Inefficient: "exponential time" for SOME inputs.
  - brute force TSP takes $N! > 2^n$ steps

Broad and robust definition has led to explosion of useful algorithms for wide spectrum of problems.

Exponential Growth

Exponential growth dwarfs technological change.
- Suppose each electron in the universe had power of today's supercomputers . . .
- And each works for the life of the universe in an effort to solve TSP problem via brute force $N!$ algorithm.

<table>
<thead>
<tr>
<th>Some Numbers</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supercomputer instructions per second</td>
<td>$10^{12}$</td>
</tr>
<tr>
<td>Second per year</td>
<td>$10^9$</td>
</tr>
<tr>
<td>Age of universe in years †</td>
<td>$10^{13}$</td>
</tr>
<tr>
<td>Electrons in universe †</td>
<td>$10^{79}$</td>
</tr>
</tbody>
</table>

- Will not succeed for 1,000 city TSP!
  $1000! \gg 10^{1000} \gg 10^{79} \times 10^{13} \times 10^9 \times 10^{12}$

Properties of Problems

Which PROBLEMS will we be able to solve in practice?
- Those with efficient (polynomial-time) algorithms.

How can I tell if I am trying to solve such a problem?
- 2-COLOR: yes, linear algorithms.
- 3-COLOR: probably no.
- 4-COLOR: yes, trivial algorithm. (finding coloring is complicated)
- No easy answers!
  Theory of "NP-completeness" helps.

Definition of P:
- Set of all DECISION problems solvable in POLYNOMIAL TIME on a DETERMINISTIC Turing machine.

MULTIPLE: Is the integer $y$ a multiple of $x$?
- YES: $(x, y) = (17, 51)$.
- NO: $(x, y) = (17, 50)$.

RELPRIME: Are the integers $x$ and $y$ relatively prime?
- YES: $(x, y) = (34, 39)$.
- NO: $(x, y) = (34, 51)$.

Definition important because of Extended Church-Turing thesis.
Strong Church-Turing Thesis

Extended Church-Turing thesis:
- If function is computable by piece of hardware in time $T(n)$ for input of size $n$, then computable by TM in time $(T(n))^k$ for some $k$.
- $P$ is the set of all decision problems solvable in polynomial time on REAL computers.

Evidence supporting thesis:
- True for all physical computers.
  - can create deterministic TM that EFFICIENTLY simulates any existing digital computer

Possible exception:
- Quantum computers?

NP

$EXP$: set of all decision problems solvable in EXPONENTIAL TIME on a deterministic Turing machine.

$NP$: does NOT mean "not polynomial."

$NP$: set of all decision problems with efficient CERTIFICATION algorithm.
- Efficient: polynomial number of steps on deterministic TM.
- Certifier: check whether a proposed "solution" is correct.
  - proposed solution is called CERTIFICATE (a hint)
  - technical condition: certificate must be of polynomial size

Certifiers and Certificates

COMPOSITE: Given integer $s$, is $s$ composite?

Observation. $s$ is composite $\iff$ there exists an integer $1 < t < s$ such that $s$ is a multiple of $t$.
- YES instance: $s = 437,669$.
  - certificate $t = 541$ or $809$ (a factor)
- NO instance: $s = 437,677$.
  - no certificate can fool verifier into saying YES

Conclusion: COMPOSITE $\in NP$. 
Certifiers and Certificates

3-COLOR: Given planar map, can it be colored with 3 colors?

Certifier:
1. Check that s and t describe same map.
2. Count number of distinct colors in t.
3. Check all pairs of adjacent states.

NO

Input s: Certificate t:

s is a YES instance NO conclusion

3-COLOR ∈ NP.

Alternate Definition of NP

NP: set of decision problems with efficient certification algorithms.

NP: set of all decision problems solvable in polynomial time on a NONDETERMINISTIC Turing machine.

- Equivalent definition.
- Intuition: nondeterministic TM can guess and check all possible solutions in parallel.
- Real computer can simulate nondeterministic TM, but takes exponential time unless you get "lucky."

\[ P \subseteq NP \subseteq EXP \]

The Main Question

Does P = NP? (Edmonds, 1962)
- Is the original DECISION problem as easy as CERTIFICATION?
- Does nondeterminism help you solve problems faster?

Most important open problem in computer science.
- If yes, staggering practical significance.
- Clay Foundation Millennium $1 million prize.

The Main Question

Does P = NP?
- Is the original DECISION problem as easy as CERTIFICATION?

If yes, then:
- Efficient algorithms for 3-COLOR, TSP, FACTOR.
- Cryptography is theoretically impossible (except for one-time pads) on conventional machines.
- Modern banking system will collapse.

If no, then:
- Can't hope to write efficient algorithm for TSP.
  - see NP-completeness
The Main Question

Does P = NP?
- Is the original DECISION problem as easy as CERTIFICATION?

Probably no, since:
- Thousands of researchers have spent four decades in search of polynomial algorithms for many fundamental NP problems without success.
- Consensus opinion: P $\neq$ NP.

But maybe yes, since:
- No success in proving P $\neq$ NP either.

NP-Complete

Definition of NP-complete:
- A problem in NP with the property that if it can be solved efficiently, then it can be used as a subroutine to solve any other problem in NP efficiently.
- "Hardest computational problems" in NP.

Links together a huge and diverse number of fundamental problems:
- TSP, 3-COLOR, CIRCUIT-SAT, thousands more.
- Given an efficient algorithm for 3-COLOR, can efficiently solve TSP, CIRCUIT-SAT, FACTOR, etc.
- Can implement any program in 3-COLOR.

Note: FACTOR not known to be NP-complete.

Notorious complexity class.
- Only exponential algorithms known for these problems.
- Called INTRACTABLE - unlikely that they can be solved given limited computing resources.

Reduction

Reduction is a general technique for showing that one problem is harder (easier) than another.
- For problems Y and X, we can often show: if Y can be solved efficiently, then so can X.
- In this case, we say X reduces to Y. (X is "easier" than Y).

Warmup: PRIMALITY reduces to FACTOR.
- Given an efficient algorithm for FACTOR(x, U), want to design an efficient algorithm for PRIMALITY(p).
  - Step 1: Compute FACTOR(p, p).
  - Step 2: If answer = YES, return NO; otherwise return YES.

- Original problem: Is p = 437,669 prime?
The World's First NP-Complete Problem

SAT is NP-complete. (Cook-Levin, 1960s)

Idea of proof:
- Given problem \( X \in \text{NP} \), by definition there exists nondeterministic TM \( M \) that solves \( X \) in polynomial time.
- Use Boolean variables to model which symbol occupies cell \( i \) at step \( t \), location of read head at step \( t \), state of finite control at step \( t \), etc.
- Use logic gates to ensure machine makes legal moves, etc.
- SAT instance is satisfiable if and only if TM outputs YES.

Minesweeper Consistency Problem

Minesweeper.
- Start: blank grid of squares.
- Some squares conceal mines; the rest are safe.
- Find location of all mines without detonating any.
- Choose a square.
  - if mine underneath, it detonates and you lose
  - If no mine, computer tells you # mines in neighboring squares
- Repeat.
Minesweeper Consistency Problem

Minesweeper consistency problem.
- Given a state of what purports to be a Minesweeper game, is it logically consistent?

Claim. Minesweeper consistency is NP-complete.
- Proof idea: reduce from circuit satisfiability.
- Build circuit by laying out appropriate minesweeper configurations.

\[
\begin{array}{cccccccccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & x & x' & 1 & x & x' & 1 & x & x' & 1 & x & x' & 1 & x & x' & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\end{array}
\]

A Minesweeper Wire

Coping With NP-Completeness

Hope that worst case doesn’t occur.
- Complexity theory deals with worst case behavior. The instance(s) you want to solve may be “easy.”
  - TSP where all points are on a line or circle
  - 13,509 US city TSP problem solved

(Cook et. al., 1998)
Coping With NP-Completeness

Hope that worst case doesn’t occur.

Change the problem.
- Develop a heuristic, and hope it produces a good solution.
  - TSP assignment
  - Metropolis algorithm, simulating annealing, genetic algorithms
- Design an approximation algorithm: algorithm that is guaranteed to find a high-quality solution in polynomial time.
  - active area of research, but not always possible!
  - Euclidean TSP tour within 1% of optimal

Sanjeev Arora (1997)

Coping With NP-Completeness

Hope that worst case doesn’t occur.

Change the problem.

Exploit intractability.

Keep trying to prove P = NP.

Summary

Many fundamental problems are NP-complete.
- TSP, CIRCUIT-SAT, 3-COLOR.

Theory says we probably won’t be able to design efficient algorithms for NP-complete problems.
- You will surely run into these problems in your scientific life.
- If you know about NP-completeness, you can identify them and avoid wasting time and energy.

A person can be at most two of the following three things:
- Honest.
- Intelligent.
- A politician.

If a problem is NP-complete, you can design an algorithm to do at most two of the following three things:
- Solve the problem to optimality.
- Solve the problem in polynomial time.
- Solve arbitrary instances of the problem.