What is the computational cost of automating brilliance or serendipity?
(Computational complexity and P vs NP question)
Combination lock

Why is it secure?
(Assume it cannot be picked)

Ans: Combination has 3 numbers 0-39…
thief must try $39^3 = 59,319$ combinations
Exponential running time

$2^n$ time to solve instances of “size” $n$

Increase $n$ by 1 $\rightarrow$ running time doubles!

Main fact to remember:

For $n = 300$, $2^n > \text{number of atoms in the visible universe.}$
Boolean satisfiability

\[(A + B + C) \cdot (\overline{D} + F + G) \cdot (\overline{A} + G + K) \cdot (\overline{B} + P + Z) \cdot (C + \overline{U} + \overline{X})\]

- Does it have a satisfying assignment?
- What if instead we had 100 variables?
- 1000 variables?
- How long will it take to determine the assignment?
Discussion

Is there an inherent difference between being creative / brilliant and being able to appreciate creativity / brilliance?

What is a computational analogue of this phenomenon?
A Proposal

Brilliance = Ability to find “needle in a haystack”

Beethoven found “satisfying assignments” to our neural circuits for music appreciation

Comments??
There are many computational problems where finding a solution involves “finding a needle in a haystack”...
CLIQUE Problem

- In this social network, is there a CLIQUE with 5 or more students?

- CLIQUE: Group of students, every pair of whom are friends

- What is a good algorithm for detecting cliques?

- How does efficiency depend on network size and desired clique size?
Rumor mill problem

- Social network for COS 116
- Each node represents a student
- Two nodes connected by edge if those students are friends
- Johanna starts a rumor
- Will it reach Kieran?
- Suggest an algorithm
- How does running time depend on network size?
- Internet servers solve this problem all the time ("traceroute" in Lab 9).
Exhaustive Search / Combinatorial Explosion

Naïve algorithms for many “needle in a haystack” tasks involve checking all possible answers → exponential running time.

- Ubiquitous in the computational universe
- Can we design smarter algorithms (as for “Rumor Mill”)? Say, \( n^2 \) running time.
Harmonious Dorm Floor

Given: Social network involving n students.

Edges correspond to pairs of students who don’t get along.

Decide if there is a set of k students who would make a harmonious group (everybody gets along).

Just the Clique problem in disguise!
Traveling Salesman Problem (aka UPS Truck problem)

- **Input:** $n$ points and all pairwise inter-point distances, and a distance $k$

- **Decide:** is there a path that visits all the points ("salesman tour") whose total length is at most $k$?
Finals scheduling

- Input: $n$ students, $k$ classes, enrollment lists, $m$ time slots in which to schedule finals

- Define “conflict”: a student is in two classes that have finals in the same time slot

- Decide: if schedule with at most 100 conflicts exists?
The P vs NP Question

- **P**: problems for which solutions can be found in polynomial time \( (n^c \text{ where } c \text{ is a fixed integer and } n \text{ is “input size”}) \). Example: Rumor Mill

- **NP**: problems where a good solution can be checked in \( n^c \) time. Examples: Boolean Satisfiability, Traveling Salesman, Clique, Finals Scheduling

- **Question**: Is P = NP?
  “Can we automate brilliance?”

(Note: Choice of computational model --- Turing-Post, pseudocode, C++ etc. --- irrelevant.)
NP-complete Problems

Problems in NP that are “the hardest”
- If they are in P then so is every NP problem.

Examples: Boolean Satisfiability, Traveling Salesman, Clique, Finals Scheduling, 1000s of others

How could we possibly prove these problems are “the hardest”?
“Reduction”

“If you give me a place to stand, I will move the earth.”
– Archimedes (~ 250BC)

“If you give me a polynomial-time algorithm for Boolean Satisfiability, I will give you a polynomial-time algorithm for every NP problem.” --- Cook, Levin (1971)

“Every NP problem is a satisfiability problem in disguise.”
Dealing with NP-complete problems

1. **Heuristics** (algorithms that produce reasonable solutions in practice)

2. **Approximation algorithms** (compute provably near-optimal solutions)
Computational Complexity Theory: Study of Computationally Difficult problems.

A new lens on the world?

- Study matter → look at mass, charge, etc.
- Study processes → look at computational difficulty
Example 1: Economics

General equilibrium theory:

- Input: \( n \) agents, each has some initial endowment (goods, money, etc.) and preference function

- General equilibrium: system of prices such that for every good, demand = supply.

- Equilibrium exists [Arrow-Debreu, 1954]. Economists assume markets find it (“invisible hand”)

- But, no known efficient algorithm to compute it. How does the market compute it?
Example 2: Factoring problem

Given a number n, find two numbers p, q (neither of which is 1) such that \( n = p \times q \).

Any suggestions how to solve it?

Fact: This problem is believed to be hard. It is the basis of much of cryptography. (More next time.)
Example 3: Quantum Computation

- Central tenet of quantum mechanics: when a particle goes from A to B, it takes all possible paths all at the same time.

- [Shor’97] Can use quantum behavior to efficiently factor integers (and break cryptosystems!)

- Can quantum computers be built, or is quantum mechanics not a correct description of the world?
Example 4: Artificial Intelligence

What is computational complexity of language recognition?

Chess playing?

Etc. etc.

Potential way to show the brain is not a computer: Show it routinely solves some problem that provably takes exponential time on computers.

(Will talk more about AI in a couple weeks)
Why is P vs NP a Million-dollar open problem?

- If $P = NP$ then Brilliance becomes routine (best schedule, best route, best design, best math proof, etc…)

- If $P \neq NP$ then we know something new and fundamental not just about computers but about the world (akin to “Nothing travels faster than light”).
Next time: Cryptography (practical use of computational complexity)