Intractability
Lecture 34

Today:

- Finite Tile Puzzles
- Exponential-Time Algorithms
- NP-Complete Problems
Decidability (Review)

Decision Problem:
- Answers a “Yes” / “No” question.

Some problems don’t have a solution
- called *undecidable*
- E.g., Tiling the plane
- E.g., Does program $P$ have an infinite loop?
- E.g., Is program $P$ correct?

Some undecidable problems are “harder” than others
- Can write algorithms that:
  - are successful on restricted classes of inputs
  - work 99% of the time
- E.g., Lab grading server

These problems might not be that easy!
Finite Tiling Puzzles

Problem: Given \( N = M^2 \) tiles, can you tile an \( M \times M \) grid?

- harder version: allow to rotate / mirror

A brute force approach:

- Since there are a finitely many ways of arranging the tiles, and each arrangement can easily be tested for legality, try and test all arrangements
- decidable!

What’s the running time?

- \( N! \) arrangements means \( O(N!) \) time
- By the way, \( 9! = 362880 \)
Human Solution to Finite Tile Puzzle

You would also use brute-force, but add one tile at a time.

- If tile doesn’t fit, then try another.
- If all tries lead to failure, then remove the previous tile.

Algorithm is called \textit{backtracking}.

- recursive
- generates partial solutions
- we didn’t do 8! steps, because we rejected many permutations early

What makes a “hard” puzzle?

- many partial solutions, but . . .
- few correct solutions (usually one)
Exponential vs Polynomial Time

$N!$ is the fastest growing function yet
  - faster than any polynomial

Fastest known algorithm to solve every bug puzzle is $2^N$, which also grows very fast
  - but remember: many puzzles can be solved quickly in practice!

Rule of Thumb: A typical computer will do 1 billion operations in around 1 second.

Q. What’s the maximum practical $N$?

<table>
<thead>
<tr>
<th></th>
<th>$N$</th>
<th>$N \log N$</th>
<th>$N^2$</th>
<th>$N^3$</th>
<th>$2^N$</th>
<th>$N!$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 second</td>
<td>$10^9$</td>
<td>$4.0 \times 10^7$</td>
<td>$3.1 \times 10^4$</td>
<td>$10^3$</td>
<td>30</td>
<td>12</td>
</tr>
<tr>
<td>10 years</td>
<td>$3.2 \times 10^{17}$</td>
<td>$6.0 \times 10^{15}$</td>
<td>$5.6 \times 10^8$</td>
<td>$6.8 \times 10^5$</td>
<td>58</td>
<td>19</td>
</tr>
</tbody>
</table>
Reasonable vs Unreasonable Time

The functions $2^N$ and $N!$ easily dwarf the growth of all functions of the form $N^k$, for any fixed $k$.

Two provisos:

- $N^{1000}$ also grows really stinkingly fast
- There are some linear algorithms with massive leading constants

But for the most part, the distinction is valid:

- “good” is polynomial — *tractable*
- “bad” is super-polynomial — *intractable*
The World of All Problems

Perhaps you view the algorithmic world like:

- solvable problems (decidable)
- unsolvable problems (undecidable)

Robustness follows from Church-Turing Thesis:

- All computers have equivalent algorithmic power.

Perhaps you now view the algorithmic world like:

- problems admitting a reasonable (poly-time) algorithm
- problems not admitting a reasonable algorithm
- tractable problems
- intractable problems
- unsolvable problems (undecidable)
Intractability and NPC

Q. Is the finite tile puzzle intractable or tractable? Mathematically speaking, it is in no man’s land.

- no one has proven that exponential time is required
  - exponential lower bounds have been proved for some problems (but not this one)
- no polynomial-time algorithm has yet been found
  - backtracking is one algorithm that works well on most bug puzzles, but not all of them.

The finite tile puzzle belongs to a class called *NPC* — the *NP*-Complete problems.
Other Problems in \textit{NPC}

There are many natural problems which are similar to the finite tile puzzle. They come from a variety of domains.

- \textbf{The Travelling Salesman Problem (TSP)}
  Given a road network connecting $N$ cities, plan the fastest route that passes through all $N$ of them.

- \textbf{Subset-sum.}
  Given a list of $N$ numbers and a target number $t$, find a subset of those numbers that sums to $t$. 
More Examples

- **Knapsacking**
  Given a list of $N$ items of weight $w_1, w_2, \ldots, w_N$ and value $v_1, v_2, \ldots, v_N$, pack a car whose maximum load is $W$ such that value is maximized.

- **Scheduling**
  Given a list of $N$ students each of which has up to 5 final exams, devise the minimum schedule so that no exams overlap for any students.

- **Satisfiability**
  Given a logical expression of length $N$, find a true/false substitution that will yield "true"
NPC — Rising and Falling Together

There are several hundred problems sharing remarkable properties:

- best known algorithm is exponential
- best lower bound is polynomial
- if one is intractable, then they all are, but . . .
- if one is tractable, then they all are!

$P \equiv NP$ (Cook-Levin 1971)

- The most important open problem in CS
- Also considered a major open problem in Mathematics
A Use for Intractability

Sometimes the bad news can be used constructively:

- in cryptography and security

**General Strategy:** Devise a cryptosystem so that unauthorized decryption is expensive.

Most public-key crypto relies on large primes

- you can eavesdrop only if you can factor extremely large numbers efficiently
- integer factorization is believed to be intractable
End of CMPT 125
CMPT 125 — Semester In Review
Battle Plan:

_The set of all things you know how to do_
_The set of all things you could do_
_The set of all things that are do-able, but not in reasonable time_
_The set of all things that are impossible to do_

Comparison of running times
New partition of the set of all problems

A table showing the increase in available N if you increase the processing speed by a factor of 1000

Monkey Puzzles

Exponential time vs Polynomial Time

- _poly time usually is capped at N^6, for all practical purposes_
- _exponential is really super-polynomial - anything above a poly_

Some examples of hard problems

_TSP_
Ham path / longest path

_Subset Sum_

_Knapsacking_
3-coloring
3-sat
The Omnipotence of Computers

“Put the right kind of software into a computer, and it will do whatever you want it to. There may be limits on what you can do with the machines themselves, but there are no limits on what you can do with software.”

- Editor of a software magazine.

Not so!
The Nature of the Problem

There are problems that have no algorithm

Perhaps the failure is due to insufficient:

- money → buy a larger, more sophisticated computer
- time → wait longer for output or use a server farm
- brains → design a cleverer algorithm

But even with unlimited resources, there are some problems that defy solution

- such problems are called \textit{undecidable}
Tiling Problems

Problem: Tile a portion of the integer grid (perhaps all of it), such that adjacent edges have matching colours.

Input: A finite set of tile types, each of which you may use an infinite number of times.

Tiling Puzzle #1

Can tile the entire plane with these!
Tiling Puzzle #2

Can’t even tile a 3x3 square!

Proof:
- #3 must appear somewhere
- adjacent #3’s need green to the right of it (#2’s)
- Contradiction
Tiling The Plane

Problem: Given a finite set of tiles $T$, can $T$ be used to cover all of the integer plane?

Input: $\mathbb{N}, \square, \square, \square, \square, \ldots, \square$

No such algorithm exists!
- No such machine!
- No such program!

Tiling the plane is undecidable!
Undecidable Problems

What other problems are undecidable?
The Halting Problem: Given a program $P$ and an input $x$, ask if $P$ halts on input $x$.

- E.g., $P$: \texttt{while}(x \neq 1) \{ \ x = x - 2; \ \}

Naive algorithm to decide halting:

- simulate $P$ on input $x$
- if $P$ terminates on $x$ then return “Yes”
- else return “No”

Q. What’s wrong with this approach?
Rice’s Theorem

Can you write a program to . . .

- find bugs in other programs?
- determine if two programs are equivalent?
- determine whether a program is malicious?
- determine if a program always outputs integers?
- ...

Rice’s Theorem: You cannot write a program that determines any non-trivial property about all programs

- can’t cover them all, but sometimes can do “most”

virus-checking is undecidable too
Proving Undecidability

There is no algorithm for the halting problem.

Proof: (Alan Turing - 1936)

- Assume one exists and seek a contradiction!
  That is, there is an algorithm $Q$ that correctly decides whether or not algorithm $P$ halts on $x$

- Construct an algorithm $S$ that uses $Q$.
  - Takes $w$ as input
  - Runs $Q(w,w)$
  - if $Q(w,w)$ returns “No”, then return “Yes”
  - if $Q(w,w)$ returns “Yes”, then infinite loop

- Run $S(S)$. Does it halt?
  - If no, then $S(S)$ returns “Yes”
  - If yes, then $S(S)$ runs forever