Problem Complexity

Do all problems have efficient solutions?

When should I apply heuristics (rules of thumb) instead of looking for an exact algorithm?

What the heck are P, NP, and NP-complete?

P: algorithms that can be solved efficiently

NP: algorithms that probably can be solved heuristically

NP-complete/NP-hard: Problems that almost definitely cannot be solved both exactly & efficiently.
P = Polynomial Time Algorithms

"Never met a problem I couldn't solve"

P is the class of problems for which there is a polynomial time solution
O\(n^C\) for some constant C

Sort is O\(n\log_2 n\)

Search is O\(\log_2 n\) for sorted lists, O\(n\) for unsorted lists

Fibonacci is O\(n\)

Note: O\(n^{57}\) may not be efficient, but most problems in P are relatively efficiently solved
NP = Nondeterministic Polynomial Algorithms

"I'll know it when I see it"

Assume you are given a devious Oracle or Genie
   Knows the answer to all problems
   Will lie to you unless you thoroughly check its work
   Some similarity to Quantum Computing

Any problem whose answer can be checked in polynomial time is in NP

Why NP?
   Many interesting problems are easy to check, but hard to compute
   Some heuristic techniques (Simulated Annealing, Simulated Evolution, etc.)
   can help guess the solution given a check on correctness.
NP Example: Travelling Salesman Problem

A salesman must visit a set of N cities in any order. Given current airline rates, find a set of flights that visits all of these cities for less than X dollars.

Polynomial Algorithm?
NP Algorithm? \[ \sum \text{flight costs} \leq X \text{ dollars}; \text{ make sure each city reached; make sure flight } I \text{ start } = \text{ flight } I-1 \text{ end } \]
Heuristic? Cheapest unvisited first

Departure Flight I > 45 min + Arrival Flight I-1
P vs. NP

Are all algorithms in P also in NP?
NP algorithm for a P problem: Don’t use oracle, just create a solution.

Are any algorithms in NP not in P?
If you can prove it, there’s a Full Professorship at M.I.T. waiting for you.

What do we know?
It is likely that P ≠ NP
There are “NP-Complete” problems which are the least likely to be in P

NP-hard: a problem which, if it were in P would mean all problems in NP are in P

NP-complete: an NP-hard problem which is in NP
NP-Complete

A problem is NP-Complete if:

It is in NP.

It is NP-hard - The creation of a polynomial-time algorithm for it provides a polynomial-time algorithm for all problems in NP.

The first NP-Complete algorithm: Boolean Satisfiability (SAT)

Is there an assignment of values to variables that makes a given Boolean Equation true?

In NP: Take the solution from the Oracle and check that the Boolean Equation becomes true (Can be done in $O(n)$).

Is NP-hard: We can show that any problem in NP can be converted into a Boolean equation in polynomial time. If there was a polynomial time solution to SAT, we would thus have a polynomial time solution to all NP problems. Details in CSE 431 or CSE 531.
NP-Hard

Given that we have at least one NP-Complete problem, showing others are NP-hard becomes easier:

For a problem under consideration X
1.) Take any NP-complete problem Y (pick wisely!)
2.) Show that there is a polynomial time method for changing any instance of problem Y into an instance of problem X

Now, a polynomial solution to X means there is a polynomial solution to Y. A polynomial solution to Y implies a polynomial solution to all of NP.
Example NP-Completeness Proof

Vertex Cover:

*Given an undirected graph* \( G(V, E) \), *where* \( V \) *are the vertices and* \( E \) *are the edges, and a size* \( S \). *Find a set of vertices* \( C, C \subseteq V \), *such that for every edge at least one of its endpoints is in* \( C \), *and* \( C \) *has at most* \( S \) *elements.*

This graph has a cover of 3 vertices \( \{A, C, E\} \) and others, but not of 2.
Example NP-Completeness Proof (cont.)

Vertex cover is in NP:

- For each edge, see if one of the endpoints is in C
- Count the # of elements in C, make sure ≤ S

Vertex cover is NP-hard: Pick the NP-complete problem 3-SAT to reduce to vertex cover

3-SAT: Given a Boolean equation in Product of Sums form, where each Sum term has 3 variables, find an assignment of values to variables that makes the equation true.

\[ F = (A + B + C) \cdot (\overline{A} + C + C) \cdot (A + \overline{B} + B) \cdot (\overline{A} + \overline{B} + \overline{C}) \]

- For each Sum, at least 1 of the terms must be true
- \( \overline{A} \) and \( \overline{B} \) cannot both be true

Note: 3-SAT was shown to be NP-Complete because any SAT equation can be converted efficiently to 3-SAT form.
Example NP-Completeness Proof (cont.)

Mapping 3-SAT to Vertex cover:

Each literal becomes a vertex. **If the vertex ISN’T in the cover, it is true.**

Must ensure:

1.) At least one literal per term is true/uncircled

   Sum of \( s \) terms.

   \( s + 5 = 2 \times (\# \text{ of sum terms}) \)

   \( \implies \) forces each sum term to have 1 uncircled vertex

2.) Both \( X \) and not(\( X \)) can’t both be true

   Draw an edge between each \( X \) and \( \overline{X} \)

---

\[ F = (A + B + C) \cdot (\overline{A} + C + C) \cdot (A + \overline{B} + B) \cdot (\overline{A} + \overline{B} + \overline{C}) \]

\( A = 1 \quad C = 1 \quad \overline{B} = 1 \)
Why this Works (Intuitive)

We can map Y to X in poly time:
  \( Y \leq X \) (Y is no harder than X, or X is no easier than Y)

Y is NP-hard:
  \( Z \leq Y \) Forall Z in NP

Thus, X is NP-hard:
  \( Z \leq Y \leq X \) Forall Z in NP

  \( Z \leq X \) Forall Z in NP
Why This Works (Mathematical)

X is NP-Hard because we show that there is a polynomial time method for changing any instance of problem Y into an instance of problem X, and Y is NP-Hard.

NP-Hard means that if the problem is in P, then all NP problems are in P.

Why is X NP-Hard? What happens if X is in P?

Algorithm for Y:

Convert problem to an instance of problem X \hspace{1cm} \text{Poly Time}

Solve problem X \hspace{1cm} \text{Poly Time}

Thus, if X is in P, Y is in P.

Y is NP-hard, which means if Y is in P, then all NP problems are in P.

Thus, if X is in P, all NP problems are in P. Therefore, X is NP-hard.
How **Not** to Show NP-Hardness

Common mistake: show that the problem can be transformed to an NP-complete problem. Proves only that the problem is in NP, not that it is NP-complete.

X is NP-hard + Y can be mapped to X:

\[ Z \leq X \quad \text{For all } Z \text{ in NP, and } Y \leq X, \text{ does not mean } Z \leq Y \]

Example - transform 1-SAT to 3-SAT:

1-SAT: \[ (A) \land (\overline{B}) \land (\overline{C}) \]

3-SAT: \[ (A \lor A \lor A) \land (\overline{B} \lor B \lor \overline{B}) \land (\overline{C} \lor \overline{C} \lor \overline{C}) \]

Problem: Just because 3-SAT is capable of solving 1-SAT, doesn't mean 1-SAT is as hard as 3-SAT.

In fact 1-SAT is in P, 3-SAT is NP-Complete

\[
\text{Each literal must be true} \\
\text{If } A \land \overline{A} \Rightarrow \text{unsolvable}
\]
Some NP-Complete Problems

Travelling Salesman Problem: Find the cheapest path that visits all vertices in a graph.

2-level Logic Minimization: Find the simplest Sum-of-Products implementation for a Boolean Equation.

Integer Linear Programming: Given an mxn matrix of integers A and a column vector B of n integers, does there exist a column vector of integers X such that $A \times X \geq B$?

Graph Coloring: Given a graph and an integer K, can the graph be colored with K colors so that no two adjacent vertices have the same color?

Clique: Does the specified graph have a clique of K vertices, where each of these vertices is adjacent to all other members of the clique?

Numerous other NP-hard problems exist, including most of Physical Design!
Who Cares About NP-Hardness?

NP-Hardness means that an efficient, exact algorithm is unlikely
Heuristic algorithms are justified

Some problems believed hard aren’t, and you’re “scooped” with exact algorithm
Routing two-terminal nets in a crossbar system
Heuristic Algorithms

When a problem proves to be too complex to solve exactly, we instead come up with a heuristic algorithm. Heuristics = rules of thumb, ways of getting reasonable solutions, but which may:

1.) Not get the exact answer
2.) Not get an answer for some problems

Examples:

Find the minimum coloring for a graph.
The travelling salesman problem.

CAD example

Espresso for 2-level logic minimization