Topic 5: NP Completeness

Agenda

- Polynomial time & Verification
- NP Completeness & Reducibility
- 3-SAT & Traveling Salesman Problem

Polynomial-time algorithms

A polynomial-time algorithm is an algorithm that runs in $O(n^k)$ for some non-negative k, where n is its input size.

Examples:

- Sorting algorithms: $O(n \log n)$ for Merge Sort.
- Matrix multiplication: $O(n^{2.8})$ for Strassen's algorithm.
- Shortest Path: $O(|E| \cdot |V|) = O(|V|^3)$ for Bellman-Ford algorithm.
- Max. Flow: $O(|V||E|^2) = O(|V|^5)$ for Edmonds-Karp algorithm

Such problems are considered **tractable**!

What if, a problem is intractable?

Note: Several real-world problems fall in this category.

- **Heuristics:** Use a heuristic to construct a greedy approach.
- **Approximation:** Rather than finding exact (optimal) solutions, find a near-optimal solution using efficient approximations.
- **Compromise:** If optimality is of utmost importance, compromise on run-time performance.
- **Assumptions:** Sometimes a reasonable assumption can make the problem tractable.

This lecture is about the identification of such problems.

Binary Decision Problems

So far, in this course, we have studied optimization problems.

- Find a choice that maximizes/minimizes the objective fn.
- Example: Shortest path.

We will henceforth focus only on *binary decision problems*.

- Computational problems with yes/no answer.
- Example: Is the weight on the shortest path below W units?

Why binary decision problems?

- Simple – Easy to develop a rigorous mathematical theory
- Surprisingly general – many optimization problems can be recast as a binary decision problem.

Complexity Classes

Classify problems into classes/sets depending on how hard they are!

- **P:** Problems that can be solved in polynomial-time, i.e. $O(n^k)$.
  - P stands for *polynomial time*.
- **EXP:** Problems that can be solved in exponential-time, i.e. $O(2^{n^k})$. Although this class includes all the above three classes, it is not inclusive of all the problems in this universe.
  - Example: Chess
- **R:** Problems that can be solved by a Turing machine.
  - Set of all recursive programs (languages).

Unsolvable Problems

Lemma

Most binary decision problems are unsolvable.

Proof.

A program is a sequence of statements in some language, i.e.,

- when fed to a processor $\Rightarrow$ binary string.
- binary strings $\Rightarrow$ natural number
- Therefore, each program can be uniquely mapped to a natural number ($\mathbb{N}$).

A decision problem is a function that maps any real-valued input to $\{yes, no\}$, i.e.,

- each decision problem can be mapped to a real number ($\mathbb{R}$).

$|\mathbb{R}| \gg |\mathbb{N}|$, since $\mathbb{R}$ is uncountable, where $\mathbb{N}$ is countable.

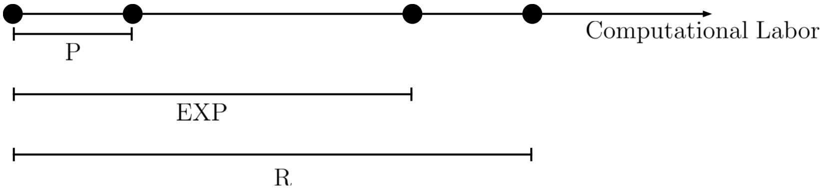
Unsolvable Problems (cont.)

Example: Halting problem

(Given the description of any arbitrary program and a finite input, decide whether the program halts or will run forever.)

There is no algorithm that can solve this problem!

If we represent all the decision problems on a real-line, we have



NP Class

Problems where "lucky" guesses can be tested for correctness in polynomial time.

- NP stands for *non-deterministic polynomial time*.
- Example: Satisfiability (SAT) – Given an arbitrary Boolean formula f such as

$$f = (x_1 \vee (\neg x_2)) \wedge ((\neg x_1) \vee x_2 \vee x_3) \wedge ((\neg x_2) \vee (\neg x_3))$$

  is there an input that makes f true?

More formally, problems verifiable by a non-deterministic Turing machine in polynomial time.

These problems may be solved by a **quantum computer** in polynomial time.

P vs. NP

Every problem in P can be verified in polynomial time.
Therefore, **P** $\in$ **NP**.

Is $P = NP$?

- Solving the problem could be hard.
- One of the 7 millennium prize problems announced by Clay Mathematics Institute (CMI) in Peterborough, NH.
- Award prize: $1,000,000

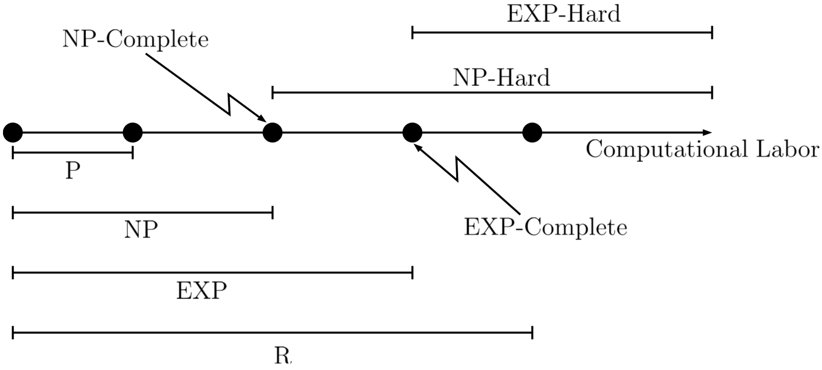
Reasonable belief: $P \neq NP$.

NP-Completeness

NP-Hard: Problems that are as hard as every problem in NP.

NP-Complete: NP $\cap$ NP-Hard

- Hardest problem in NP.



How do we prove that a problem lies in one of these classes?

If possible, find a lower bound on the run-time of a generic algorithm to the given problem – NOT EASY.
(Like how we proved a lower bound for comparison sorts).

Reduction:

- Show that Problem A can be solved in polynomial-time, if solution to Problem B is available.
- Notation: A *reduces* to B, or $A \leq B$.
- Example: Solving a system of linear equations ($A\boldsymbol{x} = \boldsymbol{b}$) reduces to inverting a matrix ($\boldsymbol{x} = A^{-1}\boldsymbol{b}$).

Result: Clusters of equivalent problems.

NP Completeness Proofs

NP Completeness can be proved from its definition.

Step 1: Prove that the problem is in NP.

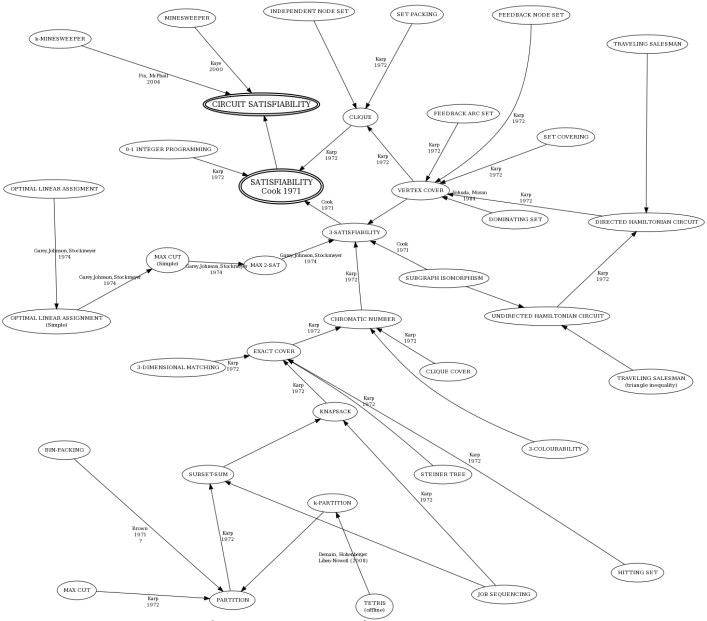
- Note that this is quite straightforward. All we need is a polynomial-time algorithm to verify a solution.

Step 2: Prove that the problem is in NP-Hard.

- Employ reduction techniques to show that the problem reduces to another NP-Complete problem.

NP-Complete cluster is the largest equivalent problem cluster.

Web of Reductions in NP-Complete Cluster



Boolean Satisfiability (SAT) Problem

Given a Boolean expression in conjunctive normal form (CNF), report YES if it can be satisfied, or else report NO?

Example: $\big(x_1 \vee x_2 \vee x_3\big) \wedge \big(x_1 \vee (\neg x_2)\big) \wedge \big(x_2 \vee (\neg x_3)\big) \wedge \big(x_3 \vee (\neg x_1)\big) \wedge \big((\neg x_1) \vee (\neg x_2) \vee (\neg x_3)\big)$

- Applications: Chip Testing, Artificial Intelligence
- First problem to be proven to be NP-Complete – Cook-Levin Theorem (1971).
- Proof beyond the scope of this course.

Let us first discuss a simpler variant (3-SAT) of this problem, when there are only three literals in every clause.

3-SAT

Why 3-SAT? (Because, 2-SAT is in P.)

Notation: Let the Boolean expression be $E = C_1 \wedge \cdots \wedge C_k$, where C_i is the i^{th} clause with exactly three literals.

- One of Richard Karp's 21 NP-Complete problems (1972).
- Run-time: $O(p(n)\alpha^n)$, where n is the number of variables in E.
- **DPLL (Davis-Putnam-Logemann-Loveland) Algorithm:** Pick a new variable x recursively and find a satisfying instance using backtracking technique.

Note: 3-SAT is in NP, since we can verify a given input in polynomial run-time.

3-SAT is NP-Hard!

Say, $C_i = (x_1 \vee \cdots \vee x_j)$. For all $i = 1, \cdots, k$, perform the following polynomial-time reduction to obtain E':

- $j = 1$: Say, $C_i = x_1$. Then, replace C_i with

$$C_i' = (x_1 \vee y \vee z) \wedge (x_1 \vee (\neg y) \vee z) \wedge (x_1 \vee y \vee (\neg z)) \wedge (x_1 \vee (\neg y) \vee (\neg z))$$

- $j = 2$: Say, $C_i = (x_1 \vee x_2)$. Then, replace C_i with

$$C_i' = (x_1 \vee x_2 \vee z) \wedge (x_1 \vee x_2 \vee (\neg z))$$

- $j > 3$: Say, $C_i = (x_1 \vee \cdots \vee x_j)$. Then, replace C_i with

$$\begin{aligned} C_i' &= (x_1 \vee x_2 \vee z_1) \wedge (x_3 \vee (\neg z_1) \vee z_2) \wedge (x_4 \vee (\neg z_2) \vee z_3) \wedge \\ &\quad \cdots \wedge (x_{j-2} \vee (\neg z_{j-4}) \vee z_{j-3}) \wedge (x_{j-1} \vee x_j \vee (\neg z_{j-3})) \end{aligned}$$

3-SAT is NP-Hard! (cont.)

Note: E' is satisfiable if and only if E is satisfiable.

- Easy to verify the first two cases, i.e. $j = 1$ and $j = 2$.
- When $j > 3$,
  - E is satisfiable $\Rightarrow$ E' is satisfiable: Assume x_m is assigned TRUE. Then, to satisfy all clauses, assign

$$z_t = \begin{cases} \text{TRUE}, \text{if } t \leq m-2 \\ \text{FALSE}, \text{if } t \geq m-1 \end{cases}.$$

  - E' is satisfiable $\Rightarrow$ E is satisfiable: If $x_1, \cdots, x_j$ are all FALSE, then $z_1, \cdots, z_{j-3}$ are all TRUE. But, the last clause in C_i' is FALSE. (Proof by Contraposition.)

Therefore, 3-SAT $\leq$ SAT.

More Examples in NP-Complete Class

- **Vertex Cover:** Given a graph $G = (V, E)$ and an integer k, return YES if there is a subset $S \subseteq V$ of size k or less such that every edge og G is incident to S (or, return NO otherwise).

  Proof: Vertex Cover $\in$ NP, Vertex cover $\leq$ 3-SAT.

- **Hamiltonian Circuit:** Given a directed graph $G = (V, E)$, return YES if there is a path in G that visits every vertex in V exactly once.

  Proof: Hamiltonian Circuit $\in$ NP, Hamiltonian Circuit $\leq$ Vertex cover.

Traveling Salesman Problem (TSP)

Given a directed graph G with positive weights, report YES if there is a tour (a cycle that passes through every vertex exactly once) within a budget B (or, report NO)?

- For more information, visit http://www.math.uwaterloo.ca/tsp/index.html

Proof: TSP $\in$ NP, TSP $\leq$ Hamiltonian Circuit.

In summary, the sequence of reductions needed are

TSP $\leq$ Hamiltonian Circuit $\leq$ Vertex cover $\leq$ 3-SAT $\leq$ SAT

One final note...

Why is 0-1 Knapsack NP-Complete, and not P?

Remember, dynamic programming algorithm has run-time $O(nW)$.

But, in most problems, we have $n \approx \log W$ (reasonable size for knapsack).

Therefore, such algorithms are called **pseudo-polynomial** algorithms.