CS 253: Algorithms

Chapter 34

NP-Completeness
Optimization & Decision Problems

- **Decision problems**
  - Given an input and a question regarding a problem, determine if the answer is yes or no

- **Optimization problems**
  - Find a solution with the “best” or “optimum” value

- Optimization problems can be cast as decision problems that are easier to study
  - *e.g.:* Shortest path: $G = \text{unweighted directed graph}$
    - Find a path between $u$ and $v$ that uses the fewest edges (optimization)
    - Does a path exist from $u$ to $v$ consisting of at most $k$ edges? (decision)
Hamiltonian Cycle

- **Optimization problem:**
  Given a directed graph $G = (V, E)$, determine a cycle that contains each and every vertex in $V$ only once.

- **Decision problem:**
  Given a directed graph $G = (V, E)$, is there a cycle that contains each and every vertex in $V$ only once.
Class of “P” Problems

- **Class P** consists of (decision) problems that are solvable in polynomial time

- Polynomial-time algorithms
  - Worst-case running time is $O(n^k)$, for some constant $k$
  - Examples: $O(n^2), O(n^3), O(1), O(n \log n)$

- Examples of non-polynomial time: $O(2^n), O(n^n), O(n!)$
Tractable/Intractable Problems

• Problems in P are called **tractable**

• Problems **not** in P are **intractable or unsolvable**
  - Can be solved in reasonable time only for small input size
  - Or, can not be solved at all

• Are non-polynomial algorithms always worst than polynomial ones?
  - $n^{500}$ is *technically* tractable, but almost impossible to solve
  - $n^{\log \log n}$ is *technically* intractable, but possible to solve for reasonable size $n$
Examples of Intractable Problems

**Hamiltonian Path:** Given a graph $G = (V, E)$, determine a path that contains each and every vertex in $V$ only once.

**Traveling Salesman:** Find a minimum weight Hamiltonian Path.

The most popular unsolvable problem

- Alan Turing discovered in the 1930’s that there are problems which are **unsolvable** by any algorithm.
- The most famous unsolvable problem is **Halting Problem:**
  
  Given an arbitrary algorithm and its input, will that algorithm eventually halt, or will it continue forever in an “infinite loop?”
**Nondeterministic and NP Algorithms**

**Nondeterministic algorithm** = two stage procedure:

1) Nondeterministic (“guessing”) stage:
   
   generate randomly an arbitrary string that can be thought of as a candidate solution (“certificate”)

2) Deterministic (“verification”) stage:
   
   take the certificate and the instance to the problem and return YES if the certificate represents a solution

**NP algorithms (Nondeterministic polynomial)**

   verification stage is polynomial

**Warning!**: NP does **not** mean “non-polynomial”

**Class NP** consists of problems that could be solved by NP algorithms (i.e., verifiable in polynomial time)
Is P = NP?

- Any problem in P is also in NP:
  \[ P \subseteq NP \]

- The big (and open question) is whether \( NP \subseteq P \) or \( P = NP \)
  - i.e., if it is always easy to check a solution, should it also be easy to find a solution?

- Most computer scientists believe that this is false but we do not have a proof …
NP-Completeness

- **NP-complete** problems are defined as the hardest problems in NP.

- Most practical problems turn out to be either P or NP-complete.

- It is important to study NP-complete problems …
Reductions

- Reduction is a way of saying that one problem is “easier” than another.

- We say that problem A is easier than problem B, (i.e., we write “A ≤ B”) if we can solve A using the algorithm that solves B.

- **Idea:** transform the inputs of A to inputs of B
Polynomial Reductions

Given two problems A, B, we say that A is polynomially reducible to B (A ≤_p B) if:

1. There exists a function f that converts the input of A to inputs of B in polynomial time

2. A(i) = YES ⇔ B(f(i)) = YES
A problem B is **NP-complete** if:

1. \( B \in \text{NP} \)
2. \( A \leq_p B \) for all \( A \in \text{NP} \)

If B satisfies only property (2) we say that B is **NP-hard**.

No polynomial time algorithm has been found for an **NP-Complete** problem.

No one has ever proven that “no polynomial time algorithm can exist for any **NP-Complete** problem”.
Implications of Reduction

If $A \leq_p B$ and $B \in P$, then $A \in P$

If $A \leq_p B$ and $A \notin P$, then $B \notin P$
Proving NP-Completeness In Practice

- Prove that the problem $B$ is in NP
  - A randomly generated string can be checked in polynomial time to determine if it represents a solution

- Show that one known NP-Complete problem can be transformed to $B$ in polynomial time
  - No need to check that all NP-Complete problems are reducible to $B$
Revisit “Is P = NP?”

**Theorem:**

If any NP-Complete problem can be solved in polynomial time then \( P = NP \)
P & NP-Complete Problems

- **Shortest simple path**
  - Given a graph $G = (V, E)$ find a **shortest** path from a source to all other vertices
  - **Polynomial solution**: $O(VE)$

- **Longest simple path**
  - Given a graph $G = (V, E)$ find a **longest** path from a source to all other vertices
  - **NP-complete**
P & NP-Complete Problems

- **Euler tour**
  - G = (V, E) a connected, directed graph find a cycle that traverses each edge of G exactly once (may visit a vertex multiple times)
  - Polynomial solution O(E)

- **Hamiltonian cycle**
  - G = (V, E) a connected, directed graph find a cycle that visits each vertex of G exactly once
  - NP-complete
Satisfiability Problem (SAT)

- **Satisfiability problem**: given a logical expression $\Phi$, find an assignment of values (T/F) to variables $x_i$ that causes $\Phi$ to evaluate to $T$(True)

  $$\Phi = x_1 \lor \neg x_2 \land x_3 \lor \neg x_4$$

- SAT was the first problem shown to be NP-complete!
CFN Satisfiability (CNF-SAT)

- CNF is a special case of SAT
- $\Phi$ is in “Conjuctive Normal Form” (CNF)
  - “AND” of expressions (i.e., clauses)
  - Each clause contains only “OR”s of the variables and their complements

E.g.: $\Phi = (x_1 \lor x_2) \land (x_1 \lor \neg x_2) \land (\neg x_1 \lor \neg x_2)$

clauses
3-CNF Satisfiability  (3-CNF-SAT)

A subcase of CNF problem:
- each clause is limited to at most three literals

\[ \Phi = (x_1 \lor \lnot x_2 \lor \lnot x_3) \land (x_2 \lor x_3 \lor x_4) \land (\lnot x_1 \lor \lnot x_3 \lor \lnot x_4) \land (x_1 \lor x_2 \lor x_4) \]

- 3-CNF is NP-Complete
- Interestingly enough, 2-CNF is in P!
Clique

Clique Problem:
- Undirected graph \( G = (V, E) \)
- **Clique**: a subset of vertices in \( V \) all connected to each other by edges in \( E \) (i.e., forming a complete graph)
- **Size of a clique**: number of vertices it contains

Optimization problem:
- Find a clique of maximum size

Decision problem:
- Does \( G \) have a clique of size \( k \)?

\[
\begin{align*}
\text{Clique}(G, 2) &= \text{YES} \\
\text{Clique}(G, 3) &= \text{NO} \\
\text{Clique}(G, 3) &= \text{YES} \\
\text{Clique}(G, 4) &= \text{NO}
\end{align*}
\]
Clique Verifier

- **Given:** an undirected graph $G = (V, E)$

- **Problem:** Does $G$ have a clique of size $k$?

- **Certificate:**
  - A set of $k$ nodes

- **Verifier:**
  - Verify that for all pairs of vertices in this set there exists an edge in $E$
CNF-SAT $\leq_p$ Clique

- **Idea:**
  - Construct a graph $G$ such that $\Phi$ is satisfiable only if $G$ has a clique of size $k=$ no. of clauses.

- Given instance of CNF-SAT, create a person for each literal in each clause.
- Two people know each other (has a connection) except if:
  - they come from the same clause
  - they represent a literal and its negation

$$(x' + y + z) (x + y' + z) (y + z')$$

$C = 3$ clauses

- Clique of size $C \Rightarrow$ satisfiable assignment.
- Satisfiable assignment $\Rightarrow$ clique of size $C$.

- $(x, y, z) = (true, true, false)$
- choose one true literal from each clause