The Class NP

- It is often not possible to find brute force algorithms to solve some problems.

- The complexity of many problems are linked.

Hamiltonian Path

- A Hamiltonian path moves through a directed graph by passing through every node exactly once.
  
  \[ \text{HAMPATH} = \{<G, s, t> | G \text{ is a directed graph with a Hamiltonian path from } s \text{ to } t.\} \]
Hamiltonian Path

- The Hamiltonian Path problem has a property called polynomial verifiability.

  - Verifying the existence of a Hamiltonian path can be easier than determining its existence.

Verifier

- A **verifier** for a language $A$ is an algorithm $V$ where $A = \{w \mid V \text{ accepts } <w, c> \text{ for some string } c\}$.

  - The time of a verifier is measured in terms of the length of $w$, so a polynomial time verifier runs in polynomial time with the length of $w$. A language $A$ is polynomial verifiable if it has a polynomial time verifier.

  - A **certificate** (or proof) is the additional information represented by $c$ and represents membership in $A$.

    - A polynomial verifier has a polynomial length (in the length of $w$) certificate because that is all the verifier can access in its time bound.
Verifier/Certificate

- The certificate for a string \( <G, s, t> \in \text{HAMPATH} \) is the Hamiltonian path from \( s \) to \( t \).

Verifier

- Note, some problems are not polynomial verifiable.
  - HAMPATH’ – The compliment of HAMPATH is not polynomial verifiable.
NP

- NP (nondeterministic polynomial time) is the class of languages that have polynomial time verifiers.
  - Theorem: HAMPATH is a member of NP.

Hamiltonian Path

- A nondeterministic TM can be built to decide HAMPATH in NP.
  - \( N = \) “On input \(<G, s, t>\), where \( G \) is a directed graph with nodes \( s \) and \( t \):
    1. Write a list of \( m \) numbers, \( p_1, \ldots, p_m \), where \( m \) is the number of nodes in \( G \). Each number in the list is nondeterministically selected (guessed at) to be between 1 and \( m \).
    2. Check for repetitions in the list. If any are found, reject.
    3. Check whether \( s = p_1 \) and \( t = p_m \). If either fail, reject.
    4. For each \( i \) between 1 and \( m-1 \), check whether \((p_i, p_{i+1})\) is an edge of \( G \). If any are not, reject. Otherwise, all tests have been passed, so accept.”
NP

- Theorem: A language is in NP iff it is decided by some nondeterministic polynomial time TM.
  - Proof idea: Convert the polynomial time verifier to an equivalent polynomial time NTM and vice versa.

NTIME

- The nondeterministic time complexity class is represented by NTIME(t(n)) = \{L \mid L is a language decided by a O(t(n)) time nondeterministic Turing machine\}.

- \( NP = \bigcup_k NTIME(n^k) \)
NP Problem Examples

- **Clique**: a clique in an undirected graph is a subgraph in which every two nodes are connected by an edge.
  - **$k$-clique**: a $k$-clique is a clique that contains $k$ nodes.
  - The **Clique problem** is one that attempts to determine whether a graph contains a clique of a specific size.
    - $\text{CLIQUE} = \{<G, k> | G \text{ is an undirected graph with a } k\text{-clique}\}$
    - Theorem: CLIQUE is in NP.

NP Problem Examples

- **Proof of CLIQUE**
  - The clique is the verifier.
  - A verifier $V$ for CLIQUE is:
    - $V = \text{“On input } <G, k>, c>:\$
      1. Test whether $c$ is a set of $k$ nodes in $G$.
      2. Test whether $G$ contains all edges connective nodes in $c$.
      3. If both pass, accept; otherwise reject.”
NP Problem Examples

- **SUBSET-SUM**: a collection of numbers exists \((x_1, \ldots, x_k)\) and a target number \(t\). The problem is to determine whether the collection contains a subcollection that adds up to \(t\).
  - SUBSET-SUM = \(\{<S, t> | S = \{x_1, \ldots, x_k\} \text{ and for some } \{y_1, \ldots, y_k\} \subseteq \{x_1, \ldots, x_k\} \text{ we have } \sum y_i = t\}\)
  - Theorem: SUBSET-SUM is in NP.

NP vs. P

- NP is the class of languages that are solvable in polynomial time on a nondeterministic TM
  - The class of languages whereby membership in the language can be verified in polynomial time.
  - NP is the class of languages for which membership can be verified quickly.
- P is the class of languages where membership can be tested in polynomial time.
  - P is the class of languages for which membership can be decided quickly.
- It is possible that NP = P, but we are unable to prove that there exists a single language in NP that is not in P.
NP-Completeness

- Certain NP problems have individual complexity related to the entire class. If a polynomial time algorithm exists for any of these problems, all problems in NP would be polynomial time solvable.
  - These problems are called NP-Complete.
  - Why do we care?

Polynomial Time Reducibility

- Previously we discussed reducing problem $A$ to problem $B$, a solution to problem $B$ can be used to solve $A$.

- Now we define a version of reducibility that accounts for the efficiency of computation.
  - When problem $A$ is efficiently reducible to problem $B$, an efficient solution to $B$ can be used to solve $A$ efficiently.
Polynomial Time Reducibility

- A polynomial time computable function is a function $f: \Sigma^* \rightarrow \Sigma^*$, if some polynomial time TM $M$ exists that halts with just $f(w)$ on its tape, when started on any input $w$.

- Language $A$ is polynomial time mapping reducible (many-one reducible), or simply polynomial time reducible, to language $B$, written $A \leq_P B$, if a polynomial time computable function $f: \Sigma^* \rightarrow \Sigma^*$ exists, where for every $w$, $w \in A \iff f(w) \in B$.
  - The function $f$ is called the polynomial time reduction of $A$ to $B$.

Theorem: If $A \leq_P B$ and $B \in P$, then $A \in P$.
- We will use this theorem to determine if a language (e.g. a problem) is NP-complete.
Satisfiability

- Boolean Formula: \( \phi = (x \land y) \lor (x \land \overline{z}) \)

- A Boolean formula is **satisfiable** if some assignment of 0s and 1s to the variables makes the formula evaluate to 1.
  - The satisfiability problem tests whether a Boolean formula is satisfiable.
    - SAT = \( \{ \phi \mid \phi \text{ is a satisfiable Boolean formula} \} \)

Satisfiability

- Cook-Levin Theorem: SAT \( \in \) P iff P = NP.
3SAT

- 3SAT is a special case of the satisfiability problem in which all formulas are in 3-conjunctive normal form.
  - What is conjunctive normal form?
  - What is a 3cnf-formula?

\[ CNF : (x_1 \lor \bar{x}_2 \lor x_3) \land (x_4 \lor x_5 \lor \bar{x}_6) \land (x_7 \lor \bar{x}_8) \]
\[ 3CNF : (x_1 \lor \bar{x}_2 \lor x_3) \land (x_4 \lor x_5 \lor x_6) \land (x_7 \lor \bar{x}_8 \lor x_9) \land (x_4 \lor \bar{x}_5 \lor \bar{x}_8) \]

3SAT

- 3SAT = \{ \phi \mid \phi \text{ is a satisfiable 3cnf-formula} \}
  - Each clause must contain at least one literal that is assigned to 1.
  - Theorem: 3SAT is polynomial time reducible to CLUQUE.
NP-Completeness

- A language $B$ is **NP-complete** if it satisfies two conditions:
  - $B$ is in NP
  - Every $A$ in NP is polynomial time reducible to $B$.

- Theorem: If $B$ is NP-complete and $B \in P$, then $P = NP$.
- Theorem: If $B$ is NP-complete and $B \leq_p C$ for $C$ in NP, then $C$ is NP-complete.

Cook-Levine Theorem

- Theorem: SAT is NP-Complete.
NP-Completeness of other languages

- We can prove NP-completeness of languages using the polynomial time reduction from a language known to be NP-complete.
  - Can frequently use SAT, but…

- Theorem: 3SAT is NP-Complete.

NP-Complete Problems

- Theorem: CLIQUE is in NP.
  - Corollary: CLIQUE is NP-complete.

- Theorem: HAMPATH is a member of NP.
  - Theorem: HAMPATH is NP-Complete.
NP-Complete Problems

- Theorem: VERTEX-COVER is NP-complete.
  - 3SAT is polynomial time reducible to VERTEX-COVER.

- Theorem: UHAMPATH (undirected HAMPATH) is NP-complete.
  - HAMPATH is polynomial time reducible to UHAMPATH.

- Theorem: SUBSET-SUM is NP-complete.
  - Reduce 3SAT to SUBSET-SUM.

- Theorem: The k-colorability problem is NP-Complete.
  - Reduce 3SAT to k-colorability.
Example

- The following solitaire game has an $m \times m$ board. One each of its $n^2$ positions lies either a blue stone, a red stone, or nothing at all. You play by removing stones from the board so that each column contains only stones of a single color and each row contains at least one stone. You win if you achieve this objective. Winning may or may not be possible, depending upon the initial configuration.
  - Let SOLITAIRE = \{\langle G \rangle \mid G \text{ is a winnable game configuration}\}.

Prove that Solitaire is NP-complete.

Example

- SOLITAIRE ∈ NP since it can be verified that a solution works in polynomial time.
- Show that 3SAT $\leq_p$ SOLITAIRE. Given $\phi$ with $m$ variables $x_1, \ldots, x_m$ and $k$ clauses $c_1, \ldots, c_k$, construct the following $k \times m$ game $G$.
  - Assume that $\phi$ has no clauses that contain both $x_j$ and $\overline{x}_j$, because such clauses may be removed without affecting satisfiability.
Example

- If $x_i$ is in $c_j$ put a blue stone in row $c_j$, column $x_i$. If $\bar{x}_i$ is in clause $c_j$ put a red stone in row $c_j$, column $x_i$. The board can be made square by repeating a row or adding a blank column as necessary without affecting solvability. Now show that $\phi$ is satisfiable iff $G$ has a solution.
  - Have to show both directions.

Example

- $(\rightarrow)$ Take a satisfying assignment.
  - If $x_i$ is true (false), remove the red (blue) stones from the corresponding column. Stones corresponding to true literals remains. Since every clause has a true literal, every row has a stone.
- $(\leftarrow)$ Take a game solution.
  - If the red (blue) stones were removed from a column, set the corresponding variable true (false). Every row has a stone remaining, so every clause has a true literal. Therefore $\phi$ is satisfied.
Structure of NP-Complete proofs

SAT

3SAT

CLIQUE

SUBSET-SUM

VERTEX-COVER

HAMPATH

TSP

UHAMPATH