Fundamentals of Algorithm Design and Machine Learning

NP:
How do we know that a problem is NP (Nondeterministic Polynomial time)?

For a given problem, if a proposed solution can be verified in polynomial time **$O(n^k)$** (for some constant k), then the problem is in **NP**.

Example: Hamiltonian Cycle Problem

Given a graph G, does there exist a cycle that visits each vertex exactly once and returns to the starting point?

If a proposed cycle is given, we can check its validity in **$O(n^2)$** time. Thus, the **Hamiltonian Cycle problem** is in **NP**.

NP-complete (NPC):
How do we know that a problem is **NP-complete (NPC)**?

A problem is **NP-complete (NPC)** if it satisfies **both** of the following conditions:

1. **It is in NP** → A proposed solution can be verified in polynomial time.
2. **It is NP-hard** → Any problem in NP can be reduced to it in polynomial time.

Exist a problem $q \in NPC$, $q \leq_P P^{NEW}$

$q \leq_P P^{NEW}$ is used to show **NP-hardness**: If an NP-complete problem reduces to P^{NEW}, then P^{NEW} is at least as hard as any NP problem.

Example: SAT Problem

The **SAT problem** is a decision problem where we are given a Boolean formula, and we must determine if there is a way to assign truth values to the variables such that the formula evaluates to **True**. It is NP-complete, meaning it is computationally difficult to solve but easy to verify a solution. The formula is in **Conjunctive Normal Form (CNF)**, which is a conjunction (AND) of clauses, where each clause is a disjunction (OR) of literals (variables or their negations).

- **Variables**: These are the Boolean variables that can take values **True** or **False** (e.g., x1, x2, x3).
- **Literals**: A literal is either a variable or its negation (e.g., x1or ¬x1).
- **Clauses**: A clause is a disjunction (OR) of literals. For example, (x1∨¬x2∨x3) is a clause.
- **Formula**: A formula is a conjunction (AND) of clauses.

For example, $(x1 \lor \neg x2 \lor x3) \land (\neg x1 \lor x2) \land (x2 \lor \neg x3)$ is a Boolean formula.
For the formula above, one valid assignment could be:

- x1=TRUE
- x2=TRUE
- x3=FALSE

This assignment makes all three clauses true, so the formula is **satisfiable**.

K-SAT Problem:

In the **K-SAT** problem, each clause in the Boolean formula has exactly **K literals** (variables or their negations), where K can be any positive integer. The task is to determine if there is a way to assign truth values to the variables such that the entire formula is **satisfied** (i.e., evaluates to **True**).

Example of 2-SAT:

For the **2-SAT** problem, the formula would look like this:

$(x1 \lor \neg x2) \land (x2 \lor x3) \land (\neg x1 \lor x3)$

Each clause contains exactly two literals.

For **4-SAT**, a formula might look like this:

$(x1 \lor x2 \lor \neg x3 \lor x4) \land (x2 \lor \neg x4 \lor x3 \lor \neg x1)$

Each clause contains exactly four literals.

To establish that the **3-SAT** problem can be **solved using the K-clique problem**, we need to show a **reduction** from the **3-SAT** problem to the **K-clique problem**. Specifically, we need to demonstrate that any instance of the **3-SAT** problem can be transformed into an instance of the **K-clique** problem, and the solution to the **K-clique** problem will give us a solution to the **3-SAT** problem.

Step-by-Step Explanation:

1. The 3-SAT Problem:

We are given a **3-SAT** formula with a set of clauses, each containing exactly **3 literals** (variables or their negations). The goal is to determine if there is a truth assignment to the variables that makes the formula **true**.

The **3-SAT** formula is in **Conjunctive Normal Form (CNF)**, which means the formula is a conjunction (AND) of clauses, where each clause is a disjunction (OR) of **3 literals**.

For example, a **3-SAT** formula might look like this:

$(x1 \lor \neg x2 \lor x3) \land (\neg x1 \lor x2 \lor x3) \land (\neg x1 \lor \neg x2 \lor x3)$

- Every clause should be 1, that means at least one variable in a clause should be 1

2. The K-Clique Problem:

The **K-clique problem** is a graph-based problem where we are given a graph G and an integer K. The task is to determine whether there is a **clique** (a subset of vertices) of size K in the graph, i.e., a set of K vertices such that every pair of vertices in the set is connected by an edge.

3. Reduction from 3-SAT to K-Clique:

To reduce the **3-SAT** problem to the **K-clique** problem, we will transform the **3-SAT** formula into a graph such that finding a **K-clique** in this graph corresponds to solving the **3-SAT** problem.

Construction of the Graph:

1. **Variables and Literals**:
   - For each literal in the 3-SAT formula, we will create a corresponding **vertex** in the graph. Each literal xi or ¬xi in the formula will be represented by a vertex in the graph.
2. **Graph Construction**:
   - For each clause in the **3-SAT** formula, we will create a **triangle** (a set of 3 vertices fully connected) in the graph, one for each of the literals in that clause. These vertices will correspond to the **literals** in the clause.
   - We need to ensure that the literals in the same clause are connected to each other (forming a clique of size 3 for that clause).
   - **Incompatibility**: We need to ensure that incompatible literals (such as xi and ¬xi) are not included in the same clique. Therefore, we will not create edges between vertices that represent incompatible literals.
3. **Clique Size K**:
   - The size of the clique we are looking for is **equal to the number of clauses** in the **3-SAT** formula. Each clause will contribute one vertex to the clique, and the **clique** will consist of one vertex from each clause (representing a **true literal** in that clause).
4. **Edges**:
   - Connect the vertices in the graph such that:
     - Vertices representing **compatible literals** (from different clauses) are connected by an edge.
     - Vertices representing **incompatible literals** (such as xi and ¬xi) are not connected.

4. Why This Works:

- A **K-clique** in this graph corresponds to a truth assignment in the **3-SAT** formula:
  - If a vertex corresponding to a literal xi is included in the clique, then the literal xi is **True**.
  - If a vertex corresponding to a literal ¬xi is included, then xi is **False**.
  - For a clique to exist, every clause in the formula must be satisfied, which means at least one literal in each clause must be **True**.

Thus, finding a **K-clique** in this graph corresponds to finding a **satisfying assignment** for the **3-SAT** formula, which proves that **3-SAT** can be solved using the **K-clique problem**.