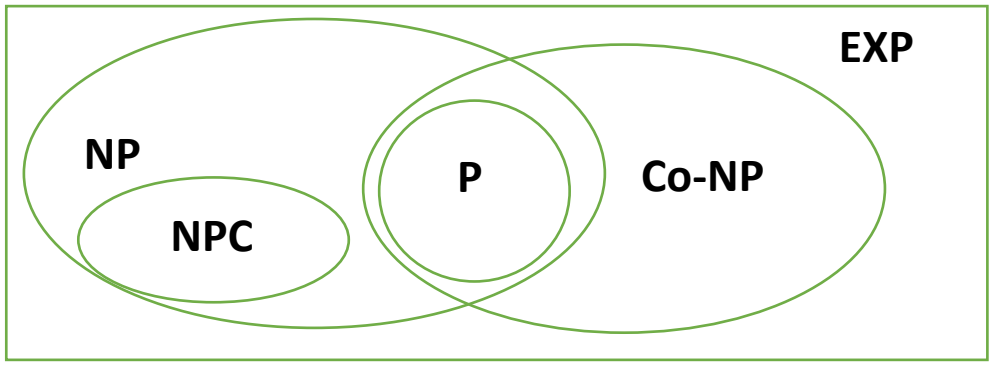
Foundations of Algorithm Design and Machine Learning

Name: Kshitish Raj Date : 06-02-2025
Roll No. : 24BM6JP25 Lecture Slot: 8:00 hrs to 9:00 hrs

Revision



Basic Complexity Class Relations

- **P ⊂ NP ∩ Co-NP** → Problems solvable in polynomial time are a subset of both NP and Co-NP.
- **NPC ⊂ NP** → NP-Complete (NPC) problems are a subset of NP.
- **If NP ⊆ P (Unknown Case)** → Two possibilities:
    1. **P = NP** → Polynomial-time solvability implies verifiability.
    2. **P ≠ NP** → Some problems in NP cannot be solved in polynomial time.

Definitions

- **P = Q** → Problem Q can be solved in **O(n^k)** time.
- **NP = Q** → Problem Q's solution can be **verified in O(n^k)** time.
- **Co-NP = Q** → Complement of Q belongs to NP.

NP-Completeness (NPC)

A problem is **NP-Complete** if:

1. **It belongs to NP.**
2. **All problems in NP can be polynomial-time reducible to it.**

Reduction Concept

- **Reduction (≤p)**: If a known **NP-Complete (NPC) problem** can be polynomially reduced to a **new problem**, then the new problem is at least as hard as the known NPC problem.
- **Notation**: **Known NPC Problem ≤p New Problem**

Problems in P (Solvable in polynomial time): **Shortest Path**
Problems in NP (Solution can be verified in polynomial time): **Longest Path**

Reduction Chain of NP-Complete Problems

1. **SAT** ∈ NPC
2. **SAT ≤p 3-SAT** (Reduction from SAT to 3-SAT)
3. **3-SAT ≤p K-Clique** (Reduction from 3-SAT to K-Clique)
4. **3-SAT ≤p Hamiltonian Cycle** (Reduction from 3-SAT to Hamiltonian Cycle)

1. Reduction from SAT to 3-SAT:

To reduce the SAT problem to the 3-SAT problem, we will transform a given Boolean formula in conjunctive normal form (CNF) into an equivalent 3-SAT formula, ensuring that each clause contains exactly three literals.

Construction of the 3-SAT Formula:

1. Handling Clauses with Three or Fewer Literals:

- If a clause already contains exactly three literals, it remains unchanged.
- If a clause contains fewer than three literals, we introduce new dummy variables to extend it to three literals.

  Examples:

  - A clause with one literal, e.g., (x1), is transformed into (x1 ∨ y ∨ ¬y), where y is a new variable.
  - A clause with one literal, e.g., (x1 ∨ x2), is transformed into (x1 ∨ y ∨ ¬y), where y is a new variable.
  - A clause with two literals, e.g., (x1∨x2), is transformed into (x1 ∨ x2 ∨ y), where y is a new variable.

2. Handling Clauses with More Than Three Literals:

- If a clause contains more than three literals, we break it into multiple clauses, each containing exactly three literals, by introducing new auxiliary variables.

  Example:

- Consider the clause (x1 ∨ x2 ∨ x3 ∨ x4 ∨ x5),
- We introduce new variables y1 and y2 and split the clause as follows:
  - (x1 ∨ x2 ∨ y1)
  - (¬y1 ∨ x3 ∨ y2)
  - (¬y2 ∨ x4 ∨ x5)

This transformation ensures that each new clause has exactly three literals while preserving the logical equivalence of the original formula.

3. Why This Works:

- The transformation preserves satisfiability:
  - If the original SAT formula is satisfiable, there exists an assignment of truth values that satisfies each clause. The introduced variables do not affect the truth assignment.
  - If the transformed 3-SAT formula is satisfiable, then the original formula is also satisfiable, as the newly introduced variables ensure logical consistency.
- Since this transformation is polynomial in size, it establishes that SAT reduces to 3-SAT in polynomial time.

Thus, this reduction proves that 3-SAT is at least as hard as SAT, demonstrating the equivalence of the two problems in terms of computational complexity.

2. Reduction from 3-SAT to K-Clique

Goal:

Transform a **3-SAT formula** into a **graph** such that finding a **K-Clique** in the graph corresponds to solving the **3-SAT** problem.

Graph Construction

1. **Vertices (Variables and Literals)**
   - Each **literal** (e.g., xi or ¬xi) in the **3-SAT formula** is represented as a **vertex** in the graph.
2. **Clause Representation (Triangles)**
   - For each **clause** (l1 ∨ l2 ∨ l3), create a **triangle** (fully connected set of three vertices) in the graph.
   - Each **triangle** contains the **three literals** in that clause.
3. **Edges (Connecting Compatible Literals)**
   - **Edges are added** between vertices that belong to **different clauses** if they are **compatible** (i.e., they do not contradict each other).
   - **Incompatibility Condition:** If two vertices represent **complementary literals** (e.g., xi and ¬xi), they are **not connected**.

K-Clique Formation

- The **clique size (K)** is equal to the **number of clauses** in the **3-SAT formula**.
- A **K-Clique** in this graph represents a **truth assignment** that satisfies all clauses.

Why This Works

- If a **K-Clique exists**, it must contain **one vertex from each clause**, meaning that each clause has at least **one true literal**.
- If a **literal is in the clique**, its **complementary literal cannot be included**, ensuring a valid truth assignment.

Reduction Chain

- **3-SAT ≤p K-Clique** → Finding a **K-Clique** in the constructed graph is **equivalent** to solving the **3-SAT problem**.

Reduction from 3-SAT to Hamiltonian Cycle

The goal is to **transform a 3-SAT formula into a directed graph** such that finding a **Hamiltonian cycle** in this graph corresponds to solving the **3-SAT** problem.

Graph Construction

1. **Vertices (Literals and Clauses Representation)**
    - Each **literal** (xi or ¬xi) in the **3-SAT formula** is represented as a **vertex** in the graph.
    - Each **clause** (l1 ∨ l2 ∨ l3) is represented as a **group of three vertices** corresponding to its literals.
    - A **start vertex (S)** and an **end vertex (T)** are introduced to guide traversal.
2. **Edges (Ensuring Logical Flow and Compatibility)**
    - **Within Each Clause:**
        - The three literals of a clause are arranged in a **cycle** (ensuring at least one literal per clause is selected).
    - **Between Clauses:**
        - Directed edges connect **compatible literals** (i.e., those that do not contradict each other) across different clauses.
    - **Avoiding Contradictions:**
        - If a literal xi is chosen, then its negation ¬xi should not be reachable.
    - **Start and End Connections:**
        - The **start vertex (S)** is connected to **one literal from the first clause**.
        - The **end vertex (T)** is connected to **one literal from the last clause**.

Hamiltonian Cycle Formation

- A **Hamiltonian cycle** in this graph must pass through **one literal from each clause**, ensuring that at least **one literal per clause is selected**.

- The cycle must return to the **start vertex** without contradictions, implying a **valid truth assignment**.

Why This Works

- If a **Hamiltonian cycle exists**, it must include **one vertex from each clause** and **avoid contradictions**.
- This ensures that there is a **truth assignment** satisfying the **3-SAT formula**.

Reduction Chain

- **3-SAT ≤p Hamiltonian Cycle** → Finding a **Hamiltonian cycle** in the constructed graph is **equivalent** to solving the **3-SAT problem**.