Non-Deterministic Polynomial Algorithms

Module 4

AoA Even 2021-22

Introduction

- Almost all the algorithms studied thus far have been polynomial-time algorithms, on inputs of size n,
- Their worst-case running time is $O(n^k)$ for some constant k.
- Not all problems can be solved in polynomial time. There are also problems that can be solved, but not in time O(nk) for any constant k.
- There is a class of problems, called "NP-complete" problems whose status is unknown.
- No polynomial-time algorithm has yet been discovered for an NP-complete problem, nor has anyone yet been able to prove that no polynomial-time algorithm can exist for any one of them.
- This so-called P ≠ NP question has been one of the deepest, most perplexing open research problems in theoretical computer science since it was first posed in 1971.

Introduction

Introduction to P and NP Classes:

- P, NP, NP-Hard and NP-Complete are classes that any problem would fall under or would be classified as.

P(Polynomial) problems:

- P problems refer to problems where an algorithm would take a polynomial amount of time to solve.
- If an algorithm is polynomial, we can formally define its time complexity as: $T(n) = O(C * n^k)$ where, $C > 0$ and $k > 0$ where C and k are constants and is input size.
- In general, for polynomial-time algorithms k is expected to be less than n.
- Many algorithms complete in polynomial time:
  - Linear Search (n)
  - Binary Search (logn)
  - Insertion Sort (n^2)
  - Merge Sort (nlogn)
  - Matrix Multiplication (n^3)

Introduction to P and NP Classes

NP (Non-deterministic Polynomial) Problems:

- NP class problems don't have a polynomial run-time to solve but have a polynomial run-time to verify solutions.
- These algorithms have an exponential complexity, which we'll define as:

  $T(n) = O(C_1 * k^{nC_2})$ where $C_1 > 0$, $C_2 > 0$ and $k > 0$ where C_1, C_2, k are constants and n is the input size.
- There are several algorithms that fit this description.
  - 0/1 knapsack problem (2^n)
  - Traveling salesperson problem
  - Sum of Subsets problem
  - Graph coloring problem
  - Hamiltonian cycles problem

Introduction to P and NP Classes

NP (Non-deterministic Polynomial) Problems:

- The problems that are solved in polynomial time are called tractable problems and the problems that require super polynomial time are called non-tractable problems.
- All deterministic polynomial time algorithms are tractable and the non-deterministic polynomials are intractable.

Introduction to P and NP Classes

Non-deterministic Algorithms:

- When the result of every operation is uniquely defined then it is called deterministic algorithm.
- When the outcome is not uniquely defined but is limited to a specific set of possibilities, we call it non-deterministic algorithm.
- New statements to specify such algorithms.
  - choice(S): arbitrarily choose one of the elements of set S
  - failure: signals an unsuccessful completion
  - success: signals a successful completion
- The assignment X:= choice(1:n) could result in X being assigned any value from the integer range[1..n]. There is no rule specifying how this value is chosen.
- The nondeterministic algorithms terminates unsuccessfully if and only if there is no set of choices which leads to the successful signal.

Introduction to P and NP Classes

Non-deterministic Algorithms:

- Example:

Searching an element x in a given set of elements A(1:n). We are required to determine an index j such that a(j) = x or j = 0 if x is not present.

```
Algorithm NSearch(A, n, key)
  {
   j = choice();
   if(key = a[j]) then
   {
   write(j); Success();
   }
   write(0);
   Failure();
  }
```

Introduction to P and NP Classes

Non-deterministic Algorithms:

- Any problem for which the answer is either zero or one (yes or no) is called a **decision problem**. An algorithm for a decision problem is termed a decision algorithm.
- Any problem that involves the identification of an optimal (either minimum or maximum) value of a given cost function is known as an optimization problem. An optimization algorithm is used to solve an optimization problem
- Many problems will have decision and optimization versions.

  e.g. Traveling salesperson problem.
  - optimization: find Hamiltonian cycle of minimum weight
  - decision: is there a Hamiltonian cycle of weight $\leq k$.

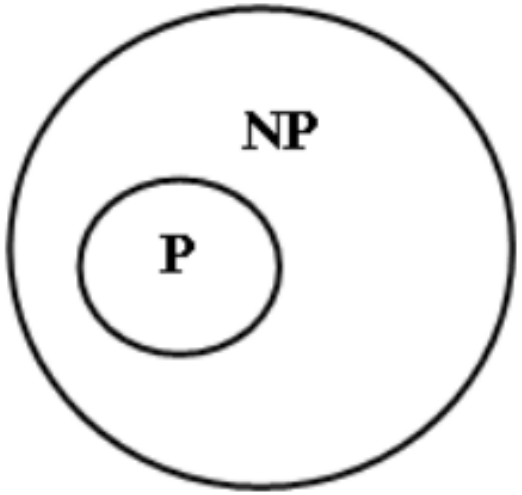
Introduction to P and NP Classes

Non-deterministic Algorithms:

Definition:

- P is a set of all decision problems solvable by a deterministic algorithm in polynomial time.
- NP is the set of all decision problems solvable by a non-deterministic algorithm in polynomial time.
  - **$\Rightarrow P \subseteq NP$**

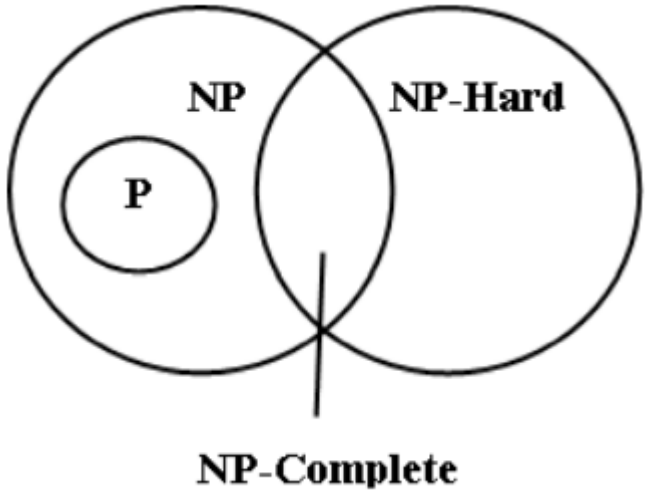
- Let L1 and L2 be problems. L1 reduces to L2 (L1$\propto$ L2) if and only if there is a way to solve L1 by deterministic polynomial time algorithm that solve L2 in polynomial time.



Introduction to NP-Hard and NP-Complete

Non-deterministic Algorithms:

- A problem is NP-Hard if all problems in NP are polynomial time reducible to it.
- A problem is NP-complete if the problem is both
  - NP-hard, and
  - NP
- All NP-Complete problems are NP-Hard, but not all NP-Hard problems are NP-Complete.
- NP-Complete problems are subclass of NP-Hard.



Introduction to NP-Hard and NP-Complete

Non-deterministic Algorithms:

- NP-Hard: A problem L is NP-Hard iff satisfiability reduces to L i.e., any nondeterministic polynomial time problem is satisfiable and reducible then the problem is said to be NP-Hard.
- Example: Halting Problem, Flow shop scheduling problem.

- NP-Complete: A problem L is NP-Complete iff L is NP-Hard and L belongs to NP (nondeterministic polynomial).
- A problem that is NP-Complete has the property that it can be solved in polynomial time iff all other NP-Complete problems can also be solved in polynomial time. (NP=P).
- If an NP-hard problem can be solved in polynomial time, then all NP- complete problems can be solved in polynomial time.

- All NP-Complete problems are NP-hard, but some NP-hard problems are not known to be NP- Complete.
- Normally the decision problems are NP-complete but the optimization problems are NP-Hard.
- Example: Knapsack decision problem can be reduced to knapsack optimization problem. There are some NP-hard problems that are not NP-Complete.

Introduction to NP-Hard and NP-Complete

Reducibility:

- The class NP-complete (NPC) problems consist of a set of decision problems (a subset of class NP) that no one knows how to solve efficiently. But if there were a polynomial solution for even a single NP-complete problem, then every problem in NPC will be solvable in polynomial time. For this, we need the concept of reductions.

- A problem Q1 can be reduced to Q2 if any instance of Q1 can be easily rephrased as an instance of Q2. If the solution to the problem Q2 provides a solution to the problem Q1, then these are said to be reducible problems.

Introduction to NP-Hard and NP-Complete

Criteria to come either in NP-hard or NP-complete:

1. The output is already given, and you can verify the output/solution within the polynomial time but can't produce an output/solution in polynomial time.
2. Here we need the concept of reduction because when you can't produce an output of the problem according to the given input then in case you have to use an emphasis on the concept of reduction in which you can convert one problem into another problem.

- *If you satisfy both points then your problem comes into the category of NP-complete class.*
- *If you satisfy the only 2nd points then your problem comes into the category of NP-hard class*

Introduction to NP-Hard and NP-Complete

Satisfiability Problem:

- The satisfiability is a Boolean formula that can be constructed using the following literals and operations.
  1. A literal is either a variable or its negation of the variable.
  2. The literals are connected with operators $\vee, \wedge, \Rightarrow, \Leftrightarrow$
  3. Parenthesis
- The satisfiability problem is to determine whether a Boolean formula is true for some assignment of truth values to the variables. In general, formulas are expressed in Conjunctive Normal Form (CNF).
- A Boolean formula is in conjunctive normal form iff it is represented by $(x_i \vee x_j \vee x_k') \wedge (x_i \vee x_j \vee x_k')$.

Introduction to NP-Hard and NP-Complete

Satisfiability Problem:

```
1    Algorithm Eval(E, n)
2    // Determine whether the propositional formula E is
3    // satisfiable. The variables are x_1, x_2, ..., x_n.
4    {
5        for i := 1 to n do  // Choose a truth value assignment.
6            x_i := Choice(false, true);
7        if E(x_1, ..., x_n) then Success();
8        else Failure();
9    }
```

Nondeterministic Satisfiability

Introduction to NP-Complete
The Clique Problem:

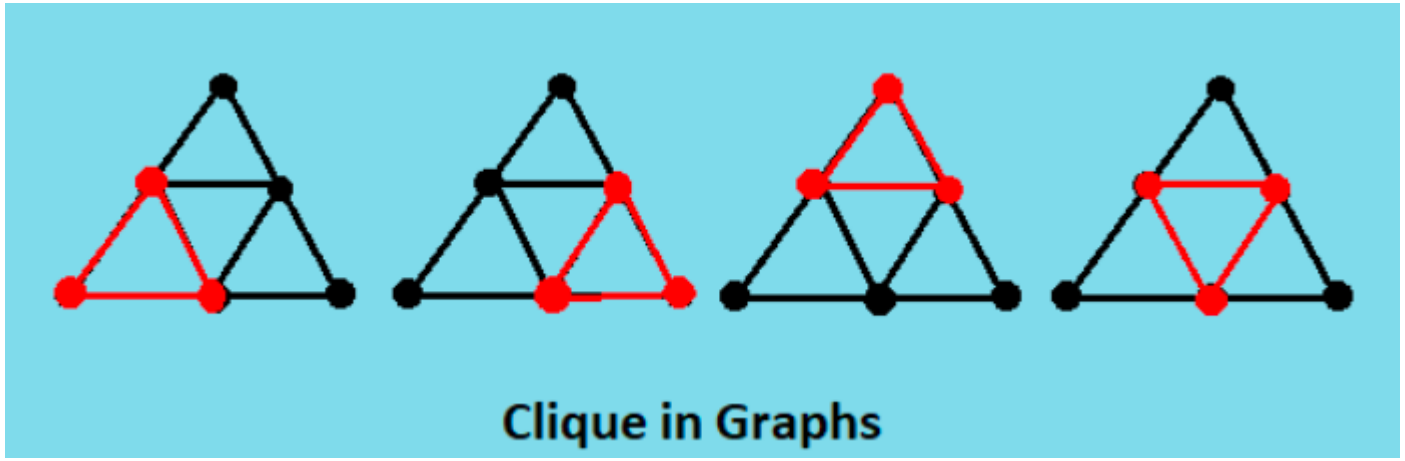
- NP-complete problems arise in diverse domains: Boolean logic, graphs, arithmetic, network design, sets and partitions, storage and retrieval, sequencing and scheduling, mathematical programming, algebra and number theory, games and puzzles, automata and language theory, program optimization, biology, chemistry, physics, and more.

- A **clique** is a subset of vertices of an undirected graph **G** such that every two distinct vertices in the clique are adjacent; that is, its induced subgraph is complete.
- The task of finding whether there is a clique of a given size in a graph (**the clique problem**) is NP-complete.

Introduction to NP-Complete

The Clique Problem:

- A **maximal clique** is a clique that cannot be extended by including one more adjacent vertex, that is, a clique which does not exist exclusively within the vertex set of a larger clique.
- A **maximum clique** of a graph, G, is a clique, such that there is no clique with more vertices. Moreover, the clique number ω(G) of a graph G is the number of vertices in a maximum clique in G.



Introduction to NP-Complete

What is the Clique Problem?

- The **clique problem** is the computational problem of finding cliques in a graph.
- It has several different formulations depending on which cliques, and what information about the cliques, should be found.
- Common formulations of the clique problem include finding a maximum clique, finding a maximum weight clique in a weighted graph, listing all maximal cliques, and solving the decision problem of testing whether a graph contains a clique larger than a given size.
- Most versions of the clique problem are hard. The clique decision problem is NP-complete.
- The problem of finding the maximum clique is both fixed-parameter intractable and hard to approximate. And listing all maximal cliques may require exponential time as there exist graphs with exponentially many maximal cliques.

Introduction to NP-Complete

- A clique is an undirected graph G = (V, E) is a subset V' $\subseteq$ V of vertices, each pair of which is connected by an edge 'E'. That is clique is a complete sub-graph of 'G'.
- The size of clique is the no. of vertices it contains. The clique problem is the optimization problem of finding a clique of maximum size in a graph.
- As a decision problem whether a clique of a given size 'k', exist in the graph. The formula definition of clique is:

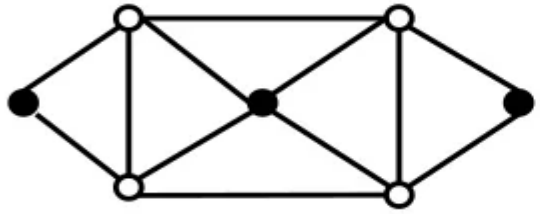
  CLIQUE = {<G, K> : 'G' is a graph with a clique of size k}

  E.g.: Clique problem is NP complete, we have to show that, i) Clique ϵ NP and ii) Clique ϵ NP-Hard.

Introduction to NP-Complete

- **Problem:**

1. Clique problem takes a graph ‘G’ and an integer ‘k’ as input and asks whether there exists a clique in a ‘G’ of size at least ‘k’ or not.
2. For proving clique to be in NP, we have to take a graph and find its vertex cover. The remaining vertices form an independent set that means there exist no edge between any two pairs of vertices in the independent set.



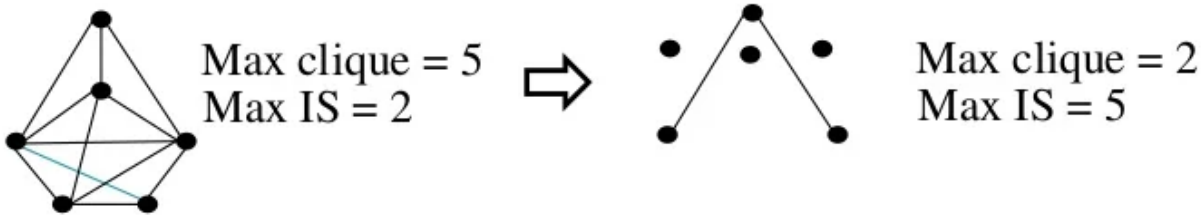
○ : Vertex in cover

● : Vertex in independent set

3. It can be easily seen that the smallest vertex cover gives the biggest independent set. For obtaining the order we have to delete the subset of vertices. Thus computing, the maximum independent set must be NP-complete.

Introduction to NP-Complete

4. It is to be noted in an independent set, no pair of vertices have edges. But in clique, there is an edge between two vertices. If we complete the graph, then an independent set becomes clique and vice-versa.
5. For instance, given a graph 'G' is present below. Maximum clique is equal to five and maximum independent set is equal to 2.



Thus to obtain the Maximum clique is NP-complete.

Note:

- If 'V_c' refers to a vertex-cover in 'G' then $V - V_c$ is a clique in G'. If c_q is a clique in 'G' then $V - c_q$ is a vertex-cover in G'.

Introduction to NP-Complete

The Vertex Cover Problem:

- Vertex cover takes a graph 'G' and integer 'k' as input and asks whether there exists a vertex cover for 'G', which contains at most 'k' vertex or not. It is not already notices that vertex cover is in NP. We have to show that it is NP-Hard.
- For this we have to reduce 3-SAT problem to it in polynomial time which is accomplished in 2 steps. That is:
  i. It represents an example in which a logic problem is reduced to graph problem.
  ii. It describes an application of the component design proof technique.
- Let B_{fg} is a given instance of 3-SAT problem that is a CNF Boolean formula, where each clause has exactly three literals.

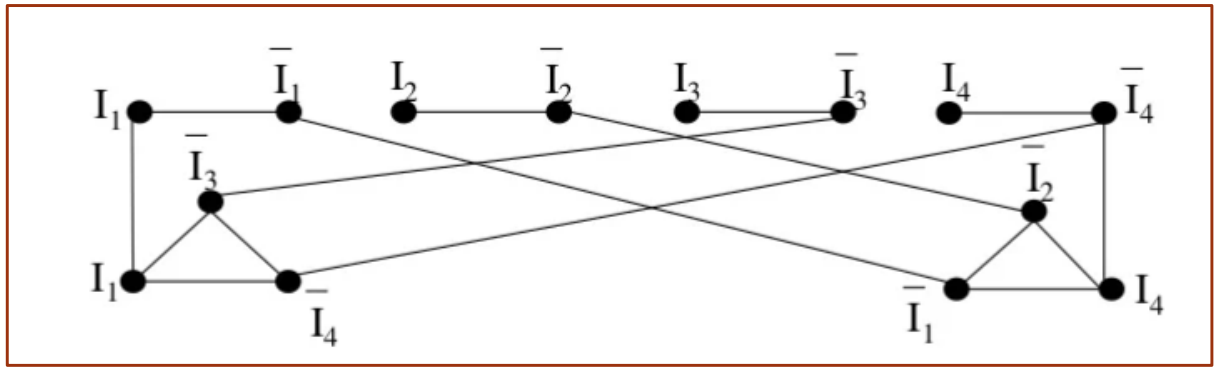
Introduction to NP-Complete

The Vertex Cover Problem:

- Now we create a graph 'G' and an integer 'k', such that 'G' has a vertex-cover of size at most 'k' iff 'Bfg' is satisfiable.
- For this, the following points are considered:
  i. For each input operand 'F_i' in Boolean formula 'B_{fg}', we add two vertices in 'G', one of which is labeled as 'I_i' and other as '$\overline{I}_i$'. After this we add the edge (I_i, $\overline{I}_i$).
  ii. For each clause Ci = (m + n + z) in B_{fg}, we form a triangle consisting of three vertices and three edges.
- At least two vertices per triangle must be in the cover for the edges in the triangle, for a total of at least 2c vertices.
- Lastly, we create a flat structure where each literal is connected to the corresponding vertices the triangle which shares the same literal given in the figure.

Introduction to NP-Complete

The Vertex Cover Problem:



- The graph will have a vertex cover of size n + 2c, iff the expression is satisfiable. It is noticeable that every cover must have at least n + 2c vertices. For showing that our reduction is correct, we have to show the following.
- **For every satisfying truth assignment there exist a cover:**
  - For this, select the 'n' vertices that correspond to the true literals to be in the cover. As it is a satisfying truth assignment, at least one of the tree cross edges associated with each clause must already be covered. Now select the other two vertices to complete the cover.

Introduction to NP-Complete

The Vertex Cover Problem:

- **There exist a satisfying truth assignment for every vertex cover:**
  - Every vertex cover must contain 'n' first level vertices and 2c second level vertices. Let the truth assignment be defined by the first level vertices. To give the cover at least one cross-edge must be covered, so that the truth assignment satisfies.
- **Approximation of vertex cover:**
  - A vertex cover of an undirected graph G = (V, E) is a subset $V_c \subseteq V$, such that if (u, v) ϵ E, then u ϵ Vc or v ϵ V_c or both. i.e.: Each vertex, covers its incident edges and a vertex cover for 'G' is a set of vertices, that all the edges in 'E'.
  - The size of a vertex is the no. of vertices in it. The vertex cover problem is to find a vertex cover of minimum size in a given graph.

Introduction to NP-Complete

The Vertex Cover Problem:

- The decision problem for 'v' to determine whether a graph has a vertex cover of a given size 'k'. The formal language is:

  Vertex cover = {<G, k>: graph 'G' has a vertex cover of a size 'k'}.

  E.g.: the vertex cover problem is NP complete, we have to show that,

  i. Vertex cover ϵ NP
  ii. Vertex cover ϵ NP-Hard

- The minimum vertex cover optimization problem is to obtain the vertex cover of minimum card inability S*. It is noticeable that the following greedy algorithm finds a vertex cover whose size is guaranteed to be not more than twice, the size of an optimal vertex cover. That is: $S < 2S^*$ or $\varrho = 2$.

Introduction to NP-Complete

The Vertex Cover Problem:

- The algorithm for obtaining the vertex-cover is given below:

```
Function Approx_vertex_cover(G)
  Step1: set V_c ← ϕ (Initialize an empty vertex cover)
  Step 2: for all (u, v) ϵ E
            If (u ϵ V_c and v ϵ V_c)
              then set V_c ← V_c U (u, v)
  Step 3: return V_c.
```

- By using Boolean vector of size |V|, we can represent the set V_c, we need O(V) time to initialize the representation of V_c and for every edge constant amount of time is required. So running time is: O(|E| + |V|)