Brief Introduction to...

P NP Class

Problem?

- In Computer Science, many problems are solved where the objective is to maximize or minimize some values, whereas in other problems we try to find whether there is a solution or not. Hence, the problems can be categorized as follows −
- Optimization Problem

Optimization problems are those for which the objective is to maximize or minimize some values.

Continued..

- Finding the minimum number of colors needed to color a given graph.
- Finding the shortest path between two vertices in a graph.
- Decision Problem
- There are many problems for which the answer is a Yes or a No. These types of problems are known as **decision problems**. For example,

Continued...

- Whether a given graph can be colored by only 4-colors.
- Finding Hamiltonian cycle in a graph is not a decision problem, whereas checking a graph is Hamiltonian or not is a decision problem.

P - Class

- The class P consists of those problems that are solvable in polynomial time, i.e. these problems can be solved in time $O(n^k)$ in worst-case, where **k** is constant.
- These problems are called **tractable**, while others are called **intractable.**

Continued…

- Formally, an algorithm is polynomial time algorithm, if there exists a polynomial $p(n)$ such that the algorithm can solve any instance of size **n** in a time $O(p(n))$.
- Problem requiring $\Omega(n^{50})$ time to solve are essentially intractable for large n. Most known polynomial time algorithm run in time $O(n^k)$ for fairly low value of k.

NP-Class

- The class NP consists of those problems that are verifiable in polynomial time. NP is the class of decision problems for which it is easy to check the correctness of a claimed answer, with the aid of a little extra information. Hence, we aren't asking for a way to find a solution, but only to verify that an alleged solution really is correct.
- Every problem in this class can be solved in exponential time using exhaustive search.

NP-Completeness

- Some problems are *intractable*:
  as they grow large, we are unable to solve them in reasonable time
- What constitutes reasonable time? Standard working definition: *polynomial time*
  - On an input of size n the worst-case running time is $O(n^k)$ for some constant k
  - Polynomial time: $O(n^2)$, $O(n^3)$, $O(1)$, $O(n \lg n)$
  - Not in polynomial time: $O(2^n)$, $O(n^n)$, $O(n!)$

NP-Complete Problems

- The *NP-Complete* problems are an interesting class of problems whose status is unknown
  - No polynomial-time algorithm has been discovered for an NP-Complete problem
  - No suprapolynomial lower bound has been proved for any NP-Complete problem, either
- We call this the $P = NP$ *question*
  - The biggest open problem in CS

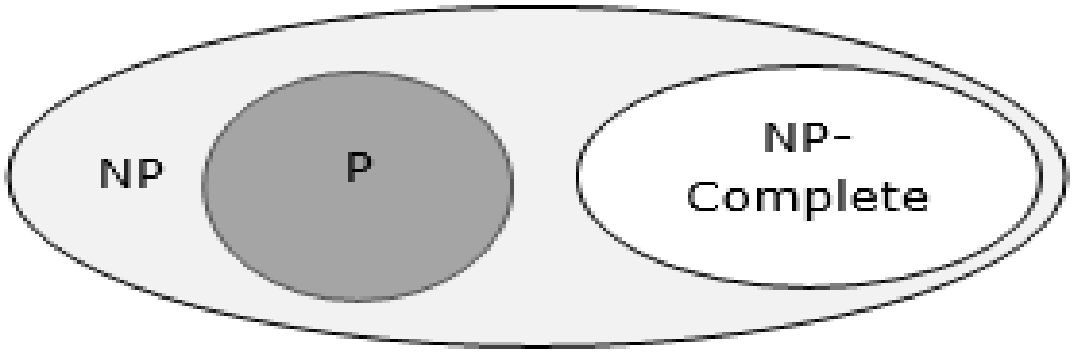
P versus NP

- Every decision problem that is solvable by a deterministic polynomial time algorithm is also solvable by a polynomial time non-deterministic algorithm.
- All problems in P can be solved with polynomial time algorithms, whereas all problems in *NP - P* are intractable.
- It is not known whether $\boldsymbol{P = NP}$. However, many problems are known in NP with the property that if they belong to P, then it can be proved that P = NP.
- If $\boldsymbol{P \neq NP}$, there are problems in NP that are neither in P nor in NP-Complete.
- The problem belongs to class **P** if it's easy to find a solution for the problem. The problem belongs to **NP**, if it's easy to check a solution that may have been very tedious to find.

NP Hard and NP-Complete Classes

- A problem is in the class NPC if it is in NP and is as **hard** as any problem in NP. A problem is **NP-hard** if all problems in NP are polynomial time reducible to it, even though it may not be in NP itself.
- 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**. The phenomenon of NP-completeness is important for both theoretical and practical reasons.

Continued....



Example:

NP-Complete Problems

- Following are some NP-Complete problems, for which no polynomial time algorithm is known.
- Determining whether a graph has a Hamiltonian cycle
- Determining whether a Boolean formula is satisfiable, etc.

NP-Hard Problems

- The following problems are NP-Hard
- The circuit-satisfiability problem
- Set Cover
- Vertex Cover
- Travelling Salesman Problem

Focus of the Cook's Theorem

- First, he emphasized the significance of polynomial time reducibility. It means that if we have a polynomial time reduction from one problem to another, this ensures that any polynomial time algorithm from the second problem can be converted into a corresponding polynomial time algorithm for the first problem.
- Second, he focused attention on the class NP of decision problems that can be solved in polynomial time by a non-deterministic computer. Most of the intractable problems belong to this class, NP.

Continued...

- Third, he proved that one particular problem in NP has the property that every other problem in NP can be polynomially reduced to it. If the satisfiability problem can be solved with a polynomial time algorithm, then every problem in NP can also be solved in polynomial time. If any problem in NP is intractable, then satisfiability problem must be intractable. Thus, satisfiability problem is the hardest problem in NP.
- Fourth, Cook suggested that other problems in NP might share with the satisfiability problem this property of being the hardest member of NP.

Reduction

- The crux of NP-Completeness is *reducibility*
  - Informally, a problem P can be reduced to another problem Q if *any* instance of P can be "easily rephrased" as an instance of Q, the solution to which provides a solution to the instance of P
    - *What do you suppose "easily" means?*
    - This rephrasing is called *transformation*
  - Intuitively: If P reduces to Q, P is "no harder to solve" than Q

Reducibility

- An example:
  - P: Given a set of Booleans, is at least one TRUE?
  - Q: Given a set of integers, is their sum positive?
  - Transformation: $(x_1, x_2, \ldots, x_n) = (y_1, y_2, \ldots, y_n)$ where $y_i = 1$ if $x_i =$ TRUE, $y_i = 0$ if $x_i =$ FALSE
- Another example:
  - Solving linear equations is reducible to solving quadratic equations
    - *How can we easily use a quadratic-equation solver to solve linear equations?*

Using Reductions

- If P is *polynomial-time reducible* to Q, we denote this $P \leq_p Q$
- Definition of NP-Complete:
  - If P is NP-Complete, $P \in \mathbf{NP}$ and all problems R are reducible to P
  - Formally: $R \leq_p P \; \forall \; R \in \mathbf{NP}$
- If $P \leq_p Q$ and P is NP-Complete, Q is also NP-Complete
  - This is the *key idea* you should take away today

The SAT Problem

- One of the first problems to be proved NP-Complete was *satisfiability* (SAT):
  - Given a Boolean expression on n variables, can we assign values such that the expression is TRUE?
  - Ex: $((x_1 \rightarrow x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$
  - **Cook's Theorem:** The satisfiability problem is NP-Complete
    - **Note: Argue from first principles, not reduction**
    - **Proof: not here**

Conjunctive Normal Form

- Even if the form of the Boolean expression is simplified, the problem may be NP-Complete
  - *Literal*: an occurrence of a Boolean or its negation
  - A Boolean formula is in *conjunctive normal form*, or *CNF*, if it is an AND of clauses, each of which is an OR of literals
    - Ex: $(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3 \vee x_4) \wedge (\neg x_5)$
  - *3-CNF*: each clause has exactly 3 distinct literals
    - Ex: $(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_3 \vee x_4) \wedge (\neg x_5 \vee x_3 \vee x_4)$
    - Notice: true if at least one literal in each clause is true

The 3-CNF Problem

- Thm 36.10: Satisfiability of Boolean formulas in 3-CNF form (the *3-CNF Problem*) is NP-Complete
  - Proof: Nope
- The reason we care about the 3-CNF problem is that it is relatively easy to reduce to others
  - Thus by proving 3-CNF NP-Complete we can prove many seemingly unrelated problems
    NP-Complete

3-CNF → Clique

- *What is a clique of a graph G?*
- A: a subset of vertices fully connected to each other, i.e. a complete subgraph of G
- The *clique problem*: how large is the maximum-size clique in a graph?
- *Can we turn this into a decision problem?*
- A: Yes, we call this the *k-clique problem*
- *Is the k-clique problem within **NP**?*

3-CNF → Clique

- *What should the reduction do?*
- A: Transform a 3-CNF formula to a graph, for which a k-clique will exist (for some k) iff the 3-CNF formula is satisfiable

3-CNF → Clique

- The reduction:
  - Let $B = C_1 \wedge C_2 \wedge \ldots \wedge C_k$ be a 3-CNF formula with *k* clauses, each of which has 3 distinct literals
  - For each clause put a triple of vertices in the graph, one for each literal
  - Put an edge between two vertices if they are in different triples and their literals are *consistent*, meaning not each other's negation
  - Run an example:
    $B = (x \vee \neg y \vee \neg z) \wedge (\neg x \vee y \vee z) \wedge (x \vee y \vee z)$

3-CNF → Clique

- Prove the reduction works:
  - If B has a satisfying assignment, then each clause has at least one literal (vertex) that evaluates to 1
  - Picking one such "true" literal from each clause gives a set V' of *k* vertices. V' is a clique (*Why?*)
  - If G has a clique V' of size k, it must contain one vertex in each triple (clause) (*Why?*)
  - We can assign 1 to each literal corresponding with a vertex in V', without fear of contradiction

Clique → Vertex Cover

- A *vertex cover* for a graph G is a set of vertices incident to every edge in G
- The *vertex cover problem*: what is the minimum size vertex cover in G?
- Restated as a decision problem: does a vertex cover of size *k* exist in G?
- Thm 36.12: vertex cover is NP-Complete

Clique → Vertex Cover

- First, show vertex cover in **NP** (*How?*)
- Next, reduce *k*-clique to vertex cover
  - The *complement* G_C of a graph G contains exactly those edges not in G
  - Compute G_C in polynomial time
  - G has a clique of size *k* iff G_C has a vertex cover of size $|V| - k$

Clique → Vertex Cover

- Claim: If G has a clique of size k, G_C has a vertex cover of size $|V| - k$
  - Let V' be the k-clique
  - Then V - V' is a vertex cover in G_C
    - Let (u, v) be any edge in G_C
    - Then u and v cannot both be in V' (*Why?*)
    - Thus at least one of u or v is in V-V' (*why?*), so edge (u, v) is covered by V-V'
    - Since true for *any* edge in G_C, V-V' is a vertex cover

Clique → Vertex Cover

- Claim: If G_C has a vertex cover $V' \subseteq V$, with $|V'| = |V| - k$, then G has a clique of size k
  - For all $u, v \in V$, if $(u,v) \in G_C$ then $u \in V'$ or $v \in V'$ or both (*Why?*)
  - Contrapositive: if $u \notin V'$ and $v \notin V'$, then $(u,v) \in E$
  - In other words, all vertices in V-V' are connected by an edge, thus V-V' is a clique
  - Since $|V| - |V'| = k$, the size of the clique is k

General Comments

- Literally hundreds of problems have been shown to be NP-Complete
- Some reductions are profound, some are comparatively easy, many are easy once the key insight is given
- You can expect a simple NP-Completeness proof on the final

Other NP-Complete Problems

- *Subset-sum*: Given a set of integers, does there exist a subset that adds up to some target *T*?
- *0-1 knapsack*: when weights not just integers
- *Hamiltonian path*: Obvious
- *Graph coloring*: can a given graph be colored with *k* colors such that no adjacent vertices are the same color?
- Etc…