ADSA – Unit 6
Advanced Topics

Dr. Sreeja S R

Topics to be covered

- Network Flows
- Randomized Algorithms
- Computational Complexity
  - NP-completeness
  - Polytime reductions

P = NP

- In computer science, there exist several famous unresolved problems, and $\mathcal{P}=\mathcal{NP}$ is one of the most studied ones.
- Until now, the answer to this problem is mainly “no”.
- And, this is accepted by the majority of the academic world.
- We probably wonder why this problem is still not resolved. And in the end, hopefully, we would have a better understanding of why $\mathcal{P}=\mathcal{NP}$ is still an open problem.

Classification

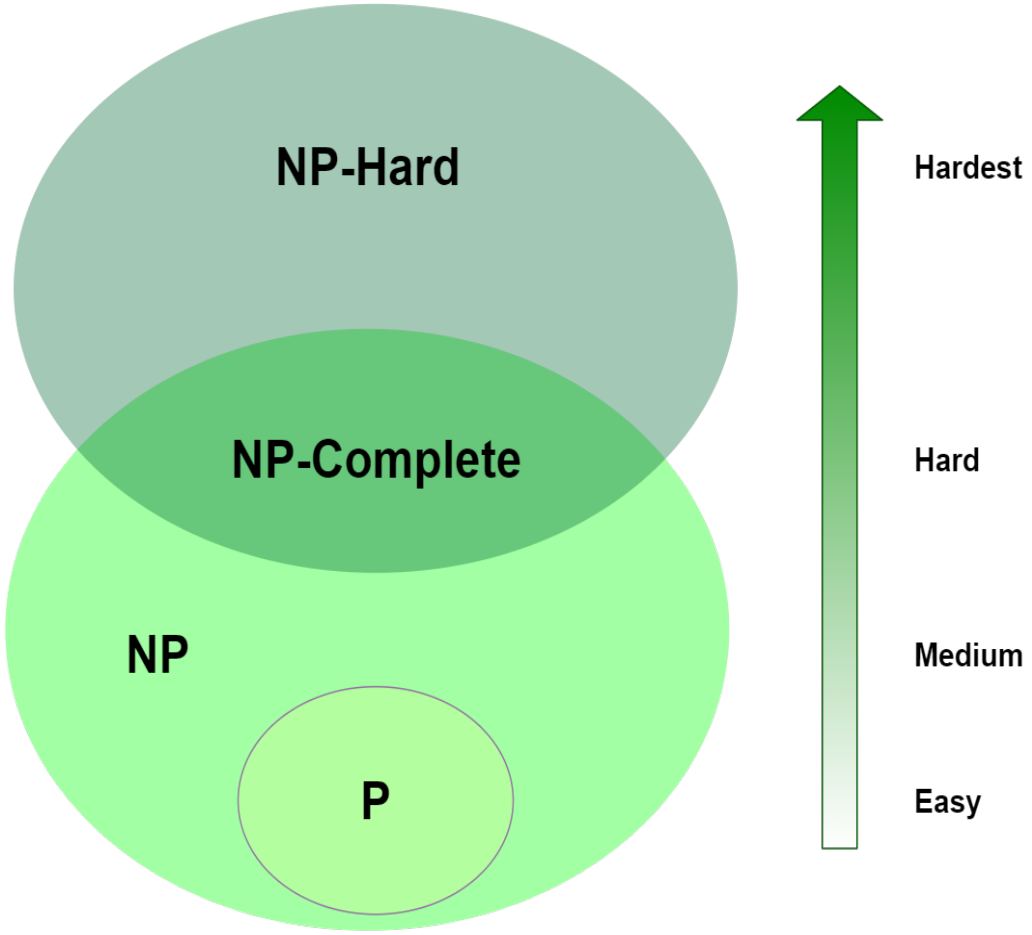
- To explain $\mathcal{P}$, $\mathcal{NP}$ and others, let's use the same mindset that we use to classify problems in real life. Let's classify it in Easy-to-Hard scale.
- In theoretical computer science, the classification and complexity of common problem definitions have two major sets; $\mathcal{P}$ which is Polynomial time and $\mathcal{NP}$ which is Non-deterministic Polynomial time.
- There are also $\mathcal{NP}$-*hard* and $\mathcal{NP}$ -*complete* sets, which we use to express more sophisticated problems.

Classification

- Easy → $\mathcal{P}$
- Medium → $\mathcal{NP}$
- Hard → $\mathcal{NP}$ *-complete*
- Hardest → $\mathcal{NP}$*-hard*

- Using the diagram, we assume that $\mathcal{P}$ and $\mathcal{NP}$ are not the same set, or, in other words, we assume that $\mathcal{P} \neq \mathcal{NP}$. This is apparently-true, but yet-unproven assertion.
- Of course, another interesting aspect of this diagram is that we've got some overlap between $\mathcal{NP}$ and $\mathcal{NP}$*-hard*.
- We call $\mathcal{NP}$ *-complete* when the problem belongs to both of these sets.

And we can visualize their relationship, too:



Algorithmic complexity

- $O(1)$ - *constant-time*
- $O(\log n)$ – *logarithmic-time*
- $O(n)$ – *linear-time*
- $O(n^2)$ – *quadratic-time*
- $O(n^k)$ – *polynomial-time*
- $O(k^n)$ – *exponential-time*
- $O(n!)$ – *factorial-time*

Where k is a constant and n is the input size. The size of n also depends on the problem definition. For example, using a number set with a size of n, the search problem has an average complexity between linear-time and logarithmic-time depending on the data structure in use.

Polynomial Algorithms

- The first set of problems are polynomial algorithms that we can solve in polynomial time, like logarithmic, linear or quadratic time.
- 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 n is input size.
- In general, for polynomial-time algorithms k is expected to be less than n.

Many algorithms complete in polynomial time:

- All basic mathematical operations; addition, subtraction, division, multiplication
- Testing for primacy
- Hashtable lookup, string operations, sorting problems
- Shortest Path Algorithms; Dijkstra, Bellman-Ford, Floyd-Warshall
- Linear and Binary Search Algorithms for a given set of numbers

Polynomial Algorithms

- All of these have a complexity of $O(n^k)$ for some k.
- we don't always have just one input, n. But, so long as each input is a polynomial, multiplying them will still be a polynomial.
- For example, in graphs, we use E for edges and V for vertices, which gives $O(E * V)$ for Bellman-Ford's shortest path algorithm. Even if the size of the edge set is $E = V^2$, the time complexity is still a polynomial, $O(V^3)$, still in $\mathcal{P}$.

Examples

Example 1: Game of checkers

- What is the complexity of determining the optimal move on a given turn?
- If the size of the board is constrained to $8 * 8$, then this is believed to be a polynomial-time problem, placing it in P.
- If the size of the board is constrained to $N * N$, it's no longer in P.

Example 2: Stable roommate problem

- To match a roommate without a tie, takes polynomial-time.
- But when ties are not allowed it's not in P.

If the input size is going to be near k, then the algorithm is going to behave more like an exponential. These variants are actually *NP-Complete*.

NP Algorithms

- The second set of problems cannot be solved in polynomial time. However, they can be verified (or certified) in polynomial time.
- If an algorithm to have an exponential complexity, can be formally define its time complexity as: $T(n) = O(C_1 * k^{C_2 * n})$ where $C_1 > 0, C_2 > 0$ and $k > 0$ where C_1, C_2 and k are constants and n is the input size.
- $T(n)$ is a function of exponential-time when at least $C_1 = 1, C_2 = 1$, we will get $O(k^n)$.
- For example, complexities like $O(n^n), O(2^n), O(2^{0.00001 * n})$.

There are several algorithms that fit this description.

- Integer Factorization and
- Graph Isomorphism

NP Algorithms

- Both of these have two important characteristics: Their complexity is $O(k^n)$ for some k and their results can be verified in polynomial time.
- Those two facts place them all in $\mathcal{NP}$ which is Non-deterministic Polynomial algorithms.
- These problems must be decision problems – have a yes or no answer – all function problems can be transformed into decision problems. This distinction helps to nail down what we mean by verified.
- These type of algorithms are in NP, if it can't be solved in polynomial time and the set of solutions to any decision problem can be verified in polynomial time by a Deterministic Turing Machine.

NP-Complete Algorithms

- This is similar to the previous set, but they are hard problems.
- There are more than 3000 of these problems, and the theoretical computer science community populates the list quickly.
- What makes them different from other NP problems is a useful distinction called completeness.
- For any NP problem that's complete, there exists a polynomial-time algorithm that can transform the problem into any other NP-Complete problem. This transformation requirement is also called **polytime reduction**.

NP-Complete Algorithms

As stated already, there are numerous NP problems proven to be complete. Among them are:

- Traveling Salesman
- Knapsack (How we reduced the time from exponential to polynomial?)
- Graph Coloring

Curiously, what they have in common, aside from being in NP, is that each can be reduced into the other in polynomial time. These facts together place them in NP-Complete.

NP-Hard Algorithms

- The last set of problems contains the hardest, most complex problems in computer science.
- They are not only hard to solve but are hard to verify as well.
- In fact, some of these problems aren't even decidable.

Among the hardest computer science problems are:

- K-means Clustering
- Traveling Salesman Problem, and
- Graph Coloring

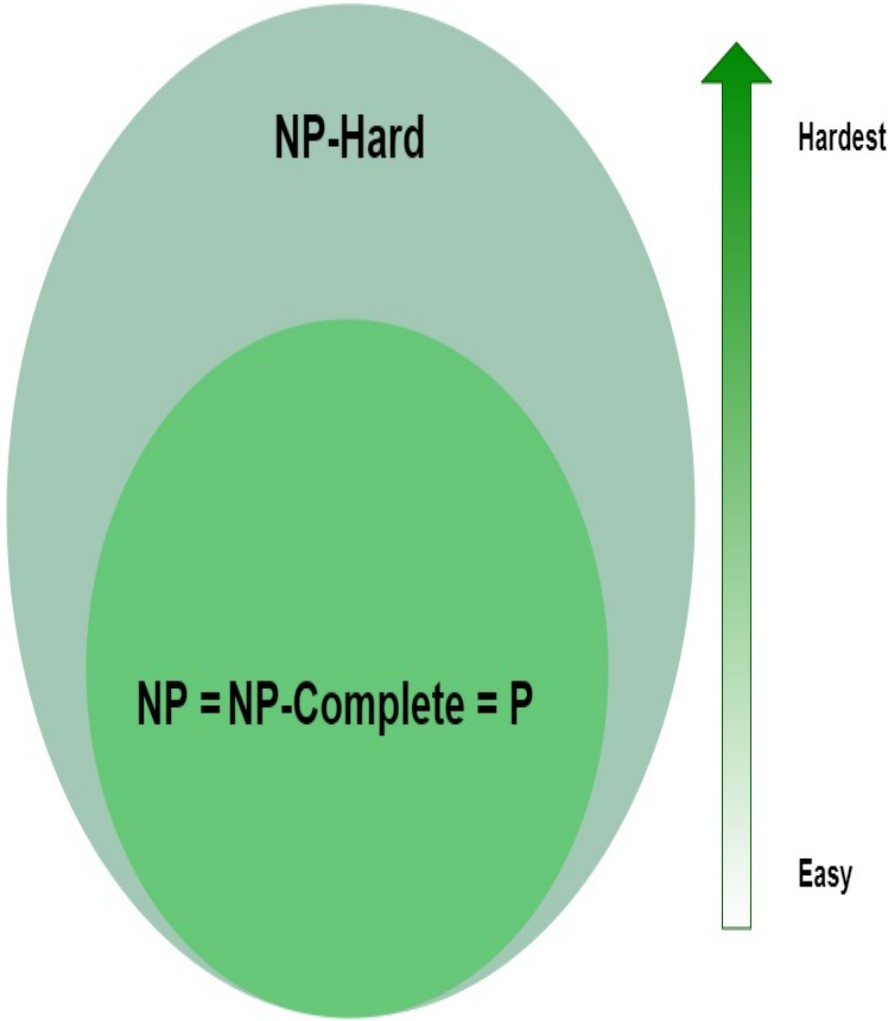
NP-Hard Algorithms

- These algorithms have a property similar to ones in NP-Complete - they can all be reduced to any problem in NP.
- Because of that, these are in NP-Hard and are at least as hard as any other problem in NP. A problem can be both in NP and NP-Hard, which is another aspect of being NP-Complete.

This characteristic has led to a debate about whether or not Traveling Salesman is indeed NP-Complete. Since NP and NP-Complete problems can be verified in polynomial time, proving that an algorithm cannot be verified in polynomial time is also sufficient for placing the algorithm in NP-Hard.

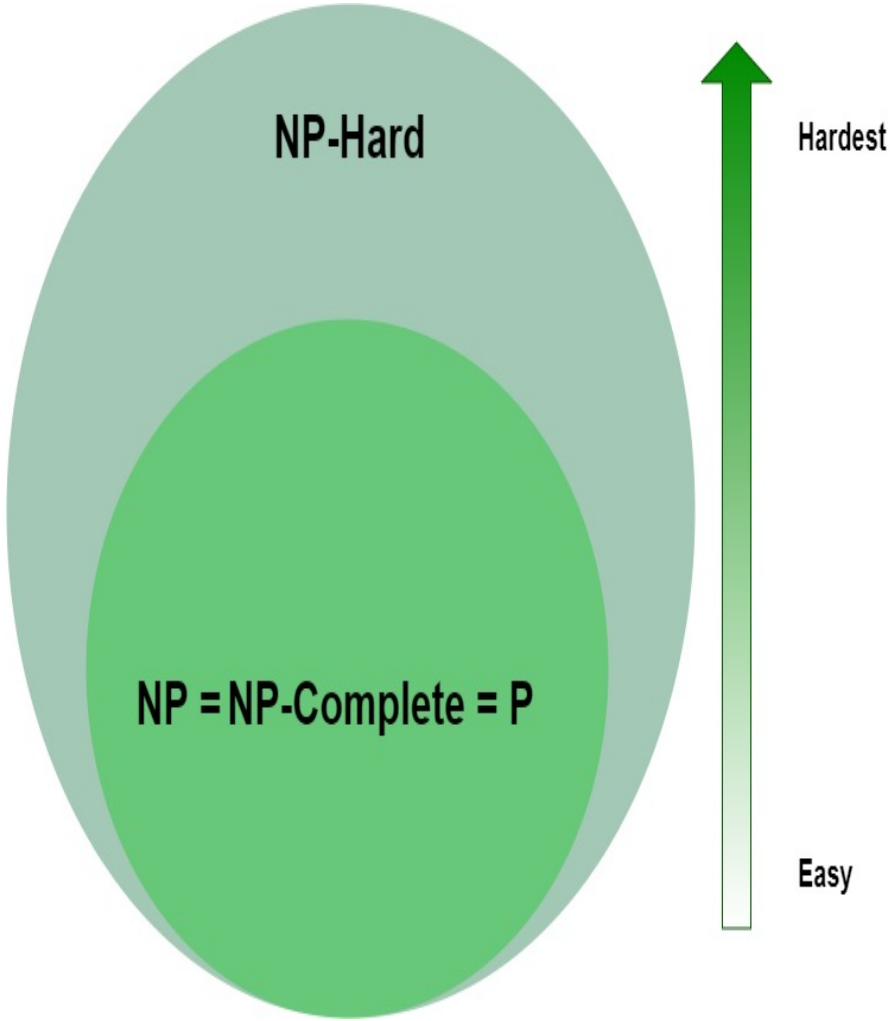
Does P=NP?

- The assumption is P!= NP, however, P=NP may be possible.
- If it were so, aside from NP or NP-Complete problems being solvable in polynomial time, certain algorithms in NP-Hard would also dramatically simplify.
- For example, if their verifier is NP or NP-Complete, then it follows that they must also be solvable in polynomial time, moving them into P = NP = NP-Complete as well.



Does P=NP?

- If P=NP means a radical change in computer science and even in the real-world scenarios.
- Currently, some security algorithms have the basis of being a requirement of too long calculation time.
- Many encryption schemes and algorithms in cryptography are based on the number factorization which the best-known algorithm with exponential complexity. If we find a polynomial-time algorithm, these algorithms become vulnerable to attacks.



Conclusion

- P problems are quick to solve.
- NP problems are quick to verify but slow to solve.
- NP-Complete problems are also quick to verify, slow to solve and can be reduced to any other NP problem.
- NP-Hard problems are slow to verify, slow to solve and can be reduced to any other NP problem.

THANK YOU