NP-Complete and NP-Hard

In this chapter we further the discussion on complexity theory with NP-complete and NP-hard complexity classes.

NP Complete

Without getting into the mathematical details and jargon, we'll stick to informally defining the complexity class NP-complete. Problems in the NP-complete class are **those problems in NP that are *as hard as* any other problems in NP**. This may sound confusing but, before we get a chance to define *NP-hard* problems, one can safely assume that NP-complete problems are the hardest problems to solve in the complexity class NP.

By definition of NP, solutions for NP-complete problems can be verified in polynomial time. The wonderful property about problems in NP-complete class is that if we find a polynomial solution for any one of them, we will have found a solution for all of the NP-complete problems - thus implying P=NP. Take a minute to think through what we just stated. It is a very powerful assertion, claiming that the solution for one problem can unlock solutions to all other problems in NP-complete!

Conversely, if we can prove that no polynomial solution exists for any one of the problems in NP-complete complexity class, then we can say that no polynomial time solution exists for all of the problems in NP-complete - thus implying P≠NP.

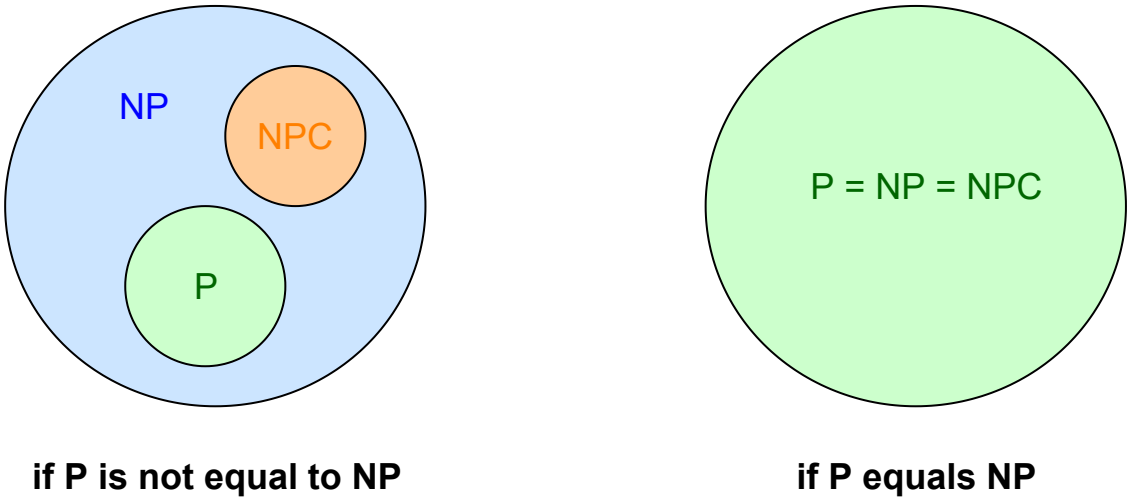
NP-complete problems exhibit two properties:

1. They are in NP, i.e. given a solution one can quickly (in polynomial time) verify that the solution is correct.
2. Every problem in NP can be converted or reduced to a problem in NP-complete. Note that since all NP-complete problems are also in NP, by definition NP-complete problems can be reduced to each other also.

The second property allows us to say that if we can solve a problem ***A*** quickly, and another problem ***B*** can be transformed into problem ***A***, then we can also solve ***B*** quickly. The caveat here is that the transformation itself should take polynomial time. If the transformation takes superpolynomial time then it defeats the purpose of transforming in the first place.

Both the graph coloring and subset sum problems are NP-complete, but string-matching isn't. Other notable NP-complete problems include:

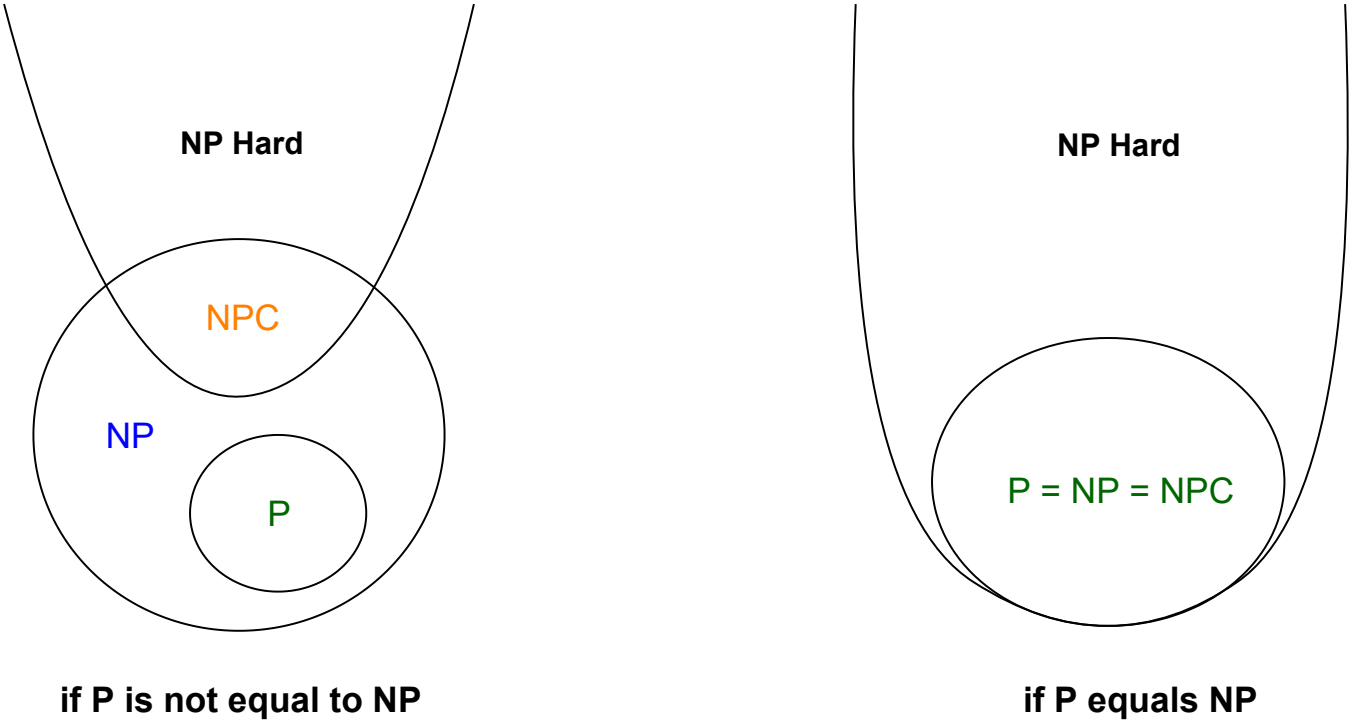
- Clique problem: A clique consists of a subset of those vertices in a graph, where each vertex is connected to every other vertex within the set. The goal is to find the maximum clique in a graph.
- Hamiltonian path problem: Given an undirected graph is there a path in the graph that visits each vertex exactly once?
- Sudoku: Solving a sudoku puzzle isn't polynomial time activity, whereas if given a solution to a Sudoku puzzle, the correctness can be verified in polynomial time.



P vs NP

NP hard

NP-hard problems are those that may not necessarily have a solution, verified for correctness in polynomial time. However, all problems in NP can be reduced or transformed into problems that are NP-hard. **NP-complete problems are in fact NP-hard and NP**. *Not all NP-hard problems are NP-complete though.* The pictorial representation appears below:



P vs NP

All problems in NP are decision problems with a yes or no outcome. Problems that are NP-hard but not NP-complete are those whose solutions can't be verified in polynomial time, let alone get solved in polynomial time. If you remember, we cast optimization problems into their decision versions so that their solutions can be verified in polynomial time. Optimization problems such as below are examples of NP-hard problems that are not NP-complete.

- Find all the max cliques in a given graph. Even if I give you a set of cliques and claim they represent all the max cliques in the graph, you can't verify in polynomial time if my claim is true or not. To verify, you need to yourself find all the max cliques and compare it with the ones I provide.
- Think about chess. If I tell you that a given move is the best or winning move, you have no way to verify my claim in polynomial time. You'll need to go through all the game trees and verify that what I claim is indeed true. The number of possible game trees is exponential.
- Minimum Vertex Cover: Find the minimum set of vertices in a graph such that every edge in the graph has at least one vertice from the set as an endpoint. This is an optimization problem. Even if provided a solution,

you may verify if it provides a cover but can't verify in polynomial time if it provides the minimum cover.