Part 1 Introduction (Combinatorial Optimization, P, NP, NP-Complete, NP-Hard)

(**Definition 1**) **Combinatorics**. Study of **finite discrete** structures, e.g., graphs (finite set of nodes and edges).

Notes: 'Finite & Discrete' mean you can count the finite set of elements. In contrast, minimization/maximization of a function in some domain, e.g., $x^2+3xy+y^2$, is regarding continuous math (e.g., calculus), which isn't studied in combinatorics.

(**Definition 2**) **Combinatorial Optimization**. Deals with finding an optimal or near-optimal solution among a finite collection of possibilities.

(**Example 1**) **Shortest Path Problem.** For a graph, where each edge has an associated non-negative weight, find the **shortest path** between 2 given vertexes.

Notes: There is a finite set of paths between two given vertexes u & v. The number of possibilities is **exponential** in the **input size** (i.e., n vertexes). The number of possibilities is $\Omega(n!)$, i.e., permutation of all the possible cases.

(**Example 2**) **Minimum Spanning Tree** (**MST**). For a given graph, find the **spanning tree** (connected acyclic subgraph) with minimum cost.

Note: We can use the **Kruskal's Algorithm** or **Prim's Algorithm** to find the **MST** of a graph.

(**Example 3**) **Travelling Salesman Problem** (**TSP**). For a given graph, find the **shortest closed path** (i.e., cycle) that visit every vertex exactly once, i.e., given a **complete weighted graph** (fully-connected graph), where the number of cycles is exponential in the number of vertexes, find the cheapest cycle.

Notes: TSP is an **NP-Complete problem**, i.e., **no polynomial-time algorithm is currently known**. The belief among most experts is that no polynomial-time algorithm exists for TSP, but no proof is known. It's one of the biggest open problems in all of math and computer science.

Features of **Combinatorial Optimization**

(1) Each **instance** (i.e., input) of the problem is associated with a finite set of **feasible solutions**. For example, any path between the 2 given vertexes of the graph is a feasible solution for the shortest path problem. Any spanning tree is a feasible solution for the MST problem.

(2) Each feasible solution is associated with a 'number' called its '**objective function value**'.

(3) Typically, the feasible solutions are described in some concise manner rather than being exactly listed.

(4) Our goal is to develop an algorithm that finds the feasible solution that minimize/maximize the objective function value.

(**Definition 3**) **Decision Problem**. A problem whose output is either 'yes' or 'no'.

(**Example 4**) **MST Decision Problem**. Given a weighted graph G and an integer K, does G has a spanning tree of cost at most K (i.e., $\leq K$)?

Notes: The **MST decision problem** can be solved in **polynomial time**. Concertely, run the algorithm for **MST optimization problem** (e.g., Kruskal's Algorithm), which is in polynomial time, and get the minimum spanning tree with the minimum cost C. If $C \leq K$, then return 'yes' for the MST decision problem and return 'no', otherwise.

Notes: We can also use the algorithm for **MST decision problem** (which is in **polynomial time**) to solve the **MST optimization problem**. Let W be the total weight of all the edges. Set K=1, 2, …, W, respectively for the MST decision problem. The total running time $W \times T(n)$, where $T(n)$ is the complexity of the **polynomial algorithm of MST decision problem**. By using **binary search**, the total running time can be reduced to $\log(W) \times T(n)$, which is also in **polynomial time**.

Notes: For most problems, the associated **optimization problem** and **decision problem** are usually with the **same complexity**. Namely, if the **decision problem** can be solved in polynomial time, then its corresponding **optimization problem** can also be solved in polynomial time.

(**Example 5**) **TSP Decision Problem**. Given a weighted graph G and an integer, does G have a TSP cycle with cost at most K (i.e., $\leq K$)?

(**Definition 4**) **Class** P. Class of **decision problems** that can be solved in **polynomial time** (w.r.t. the input size of every input in the worst case).

Notes: Class P is the set of **problems**, not algorithms.

Notes: MST decision problem is in Class P.

Notes: Whether the **TSP decision problem** is in Class P is still unknown. Experts believes that it's not in Class P.

(**Definition 5**) **Class** NP. Class of **decision problem** for which there is a **Yes-Certificate**. More precisely, for every **yes-instance** (i.e., **input** for which the correct answer is 'yes'), there should exist **a short proof/certificate** that the answer is 'yes'.

Notes: Concretely, for MST decision problem, a **yes-instance** is a **graph**, in which there exists a spanning tree with cost at most K. A **no-instance** is a graph, in which there is no snapping tree with cost at most K. Every instance is either a yes-instance or a no-instance.

Notes: Especially, '**short proof / certificate**' means that the **size of the proof** is polynomial in the input size, and it can be **verified** in polynomial time.

Notes: **TSP** is in Class NP. Concretely, for TSP, to verify an **instance** (graph G & integer K) is indeed a yes-instance, one is first given a set of tours, and then checks each tour, where the time to check a tour is in polynomial time (e.g., check whether the tour is in the input graph & whether cost of the tour is at most K). Note that we don't care about the time to find a tour with cost at most K, but only consider the time to verify a tour.

Notes: MST is in Class NP. Especially, no more certification is needed for MST decision problem, because there exist the polynomial-time algorithms (e.g., Prim's algorithm & Kruskal's algorithm) for MST optimization problem. It indicates **Theorem 1**.

(**Theorem 1**) $P \subseteq NP$.

(**Definition 6**) ***NP*-Complete**. A **decision problem** Π is said to be ***NP*-complet**e if (i) it belongs to NP and (ii) there is **a polynomial time reduction** from any problem in the class NP to Π.

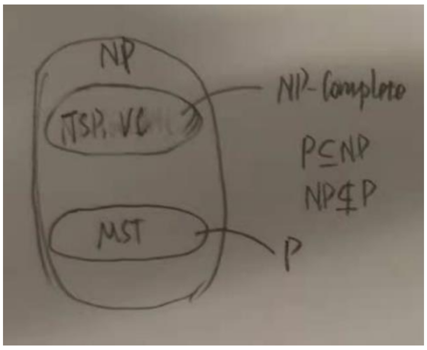
Notes: It means that if Π can be solved in polynomial time, then all the **decision problems in**

Class *NP* should be solved in polynomial time.

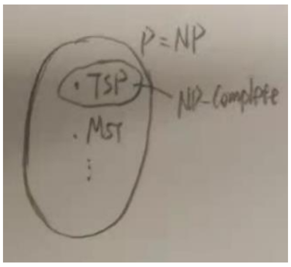
Notes: Informally, think of *NP*-complete problems as the hardest problem in Class *NP*.

(**Theorem 2**) TSP is *NP*-Complete.

According to **Theorem 1**, we have $P \subseteq NP$. There are two possibilities regarding the relationship between Class P and Class NP. First, if $NP \not\subset P$, then we have $P \neq NP$, i.e.,



Second, if $NP \subseteq P$, then we have $P = NP$, i.e.,



(**Definition 7**) **Asymmetry of NP**. What if we're given a no-instance of TSP. Can we offer a No-Certificate of this fact, i.e., can you convince someone quickly that it's a no-instance?

Fact: No one so far knows the answer to this question. The general belief is that TSP (or any other *NP*-Complete decision problems) doesn't have a No-Certificate.

Notes: For the TSP decision problem, a **yes-instance** is a graph *G* with an integer *K*, where there exists a tour with cost at most *K*. Note that TSP is in Class *NP*, i.e., one can verify a given yes-instance is indeed a yes-instance in polynomial time (a.k.a., the **Yes-Certificate**).

Notes: For the TSP decision problem, a **no-instance** is also a graph *G* with an integer *K*, where there is **no such a tour** with cost at most *K*. Given a no-instance (i.e., graph *G* and integer *K*), no one has found a method to prove that this instance is indeed a no-instance in polynomial time. The general belief is that TSP doesn't have a **no-certificate**, i.e., one cannot verify an instance is a no-instance in polynomial time.

(**Definition 8**). ***NP*-Hard**. A **decision problem** Π is said to be *NP*-hard if there is a polynomial time reduction from any problem in Class *NP* to Π.

Notes: Compared with *NP*-Complete, an *NP*-Hard problem is not required to be in Class *NP*, but an *NP*-Complete problem must be in Class *NP*.

Notes: *NP*-Hard can also be applied to **optimization problems**, but not only **decision problems**. For **optimization problem**, we don't say a problem is *NP*-Complete, but say it's **NP-hard**.