NP-Completeness

NP-Completeness is an attempt to identify “inherently difficult problems”. We show the equivalence of a particular problem to a class of problems that have no known polynomial time algorithms. It then makes sense to focus on heuristic methods to solve this problem.

The theory of NP-Completeness applies to a particular specific type of problem – decision / recognition / yes/no answerable problems. In the following section, we look at a particular optimization and show how it can be recast in the framework of recognition problem.

**TSP Optimization Problem:** given a directed graph $G(N,A)$, integer arc lengths, and costs $c_{ij}$ associated with each arc, determine a tour $W$ (i.e. a directed cycle which visits each node exactly once) with the smallest possible tour length $\sum_{(i,j)\in W} c_{ij}$.

**TSP Recognition-I:** Given a directed graph $G(N,A)$, integer arc lengths, costs $c_{ij}$ associated with each arc and a value $B$, does the network contain a tour $W$ such that $\sum_{(i,j)\in W} c_{ij} \leq B$?

Observations:
1. Clearly, a polynomial time algorithm for TSP Optimization implies a polynomial time algorithm for TSP Recognition-I.
2. A polynomial time algorithm for TSP Recognition-I implies a polynomial time algorithm for TSP Optimization.

Sketch Proof:
(1) Perform a binary search for the optimal tour length $W^*$. This requires $O(\log(nC))$ solutions of the TSP Recognition-I algorithm.
(2) For each arc, temporarily remove the arc and apply TSP Recognition-I with $W^*$. If yes, permanently delete the arc (the arc is not on the optimal path) If no, restore the arc.

After all arcs have been considered, the remaining graph is the optimal TSP tour. Thus, this method requires $O(n^2)$ TSP recognitions to find the tour.

We can then say that the TSP Optimization and TSP Recognition-I algorithm are polynomially equivalent (i.e. polynomial for TSP Optimization $\Leftrightarrow$ polynomial for TSP Recognition-I). This was an example of problem reduction: problem $P_1$ reduces to $P_2$ if we can solve $P_1$ using an algorithm where $P_2$ as a subroutine.

TSP Recognition-II: Given $G(N,A), c_{ij}, B$, does the network contain a tour $W$ such that $\sum_{(i,j)\in W} c_{ij} \geq B$?

Note that the exact formulation of the Recognition problem is important for proving polynomial equivalence. In the preceding problem, it is no longer clear that polynomial for TSP Optimization $\Leftrightarrow$ polynomial for TSP Recognition-II.

Polynomial Reduction
Problem $P_1$ polynomially reduces to $P_2$ if we can solve $P_1$ using a polynomial algorithm that uses an algorithm for $P_2$ at unit cost (in another words, the algorithm for $P_2$ is $O(1)$)

Property: If problem $P_1$ polynomially reduces to $P_2$ and some polynomial algorithm solves $P_2$, then there exists a polynomial algorithm which solves $P_1$.

Polynomial Transformation
A problem $P_1$ polynomially transforms to $P_2$ if for every instance $I_1$ of $P_1$, we can construct in polynomial time an instance of $I_2$ of $P_2$ so that $I_1$ is a ‘YES’ instance of $P_1$ if and only if $I_2$ is a ‘YES’ instance of $P_2$.

Observe: if $P_1$ polynomially reduces to $P_2$, $P_2$ is at least as hard as $P_1$. ($P_1$ might, however, be easier than $P_2$.)
Class P
A recognition problem belongs to class P if some polynomial algorithm solves it.

Class NP (Nondeterministic polynomial)
A recognition problem belongs to NP if, given a ‘YES’ answer, it can be verified in polynomial time (or more formally, there exists a certificate which can be verified in polynomial time).

Example: TSP Recognition-I
Given a tour, we can verify in polynomial time (actually in $O(n)$) if it passes through each node once and has a total length less than or equal to $B$.

Note 1: The reverse does not hold: we cannot verify a ‘NO’ answer to the TSP Recognition-I problem in polynomial time, e.g. whether there are no tours less than or equal to $B$. Thus this certificate cannot be verified in polynomial time and is thus not in NP.

Note 2: Every problem which is P is in NP, but the reverse does not necessarily hold.

Class NP-Complete
A recognition problem $P$ is NP-Complete (NPC) if:
(1) $P \in NP$
(2) All other problems in NP polynomially transform to P.

Thus, if we have an efficient algorithm for one problem in NPC, we have an efficient algorithm for all problems in NP. (So P=NP.)
Proving NP-Completeness
- Required to show both (1) and (2) in the definition
- Done by Cook (1972) for the satisfiability problem (see below)
- We can show (2) by showing that some NP-Complete problem polynomially transforms to P

Satisfiability Problem
Given a set of boolean variables (literals) \( \{X_1, X_2, \ldots, X_n\} \) whose complements are \( \overline{X_i} \), assign a label to each literal so that:
- \( X_i \) is true if \( \overline{X_i} \) is false
- \( \overline{X_i} \) is true if \( X_i \) is false

Clause: a collection of literals that is true if at least one of the literals is true (i.e. or, +)
Expression: a collection of clauses that is true if all of the clauses are true (i.e. \( \cdot \), and)

Given a set of literals and a collection of clauses defined on these literals, is there an assignment of labels to literals for which the expression is true (i.e. each and every clause is satisfied)?

e.g. \((X_1 + X_2 + X_4) \cdot (\overline{X_1} + X_2 + \overline{X_2}) \cdot (\overline{X_2} + \overline{X_4})\) is satisfied (for example if \( X_2 \) is true and all others false)