2.3. NP-COMPLETENESS

**Definition 2.7.a.** Decision problem $Q_1$ *reduces polynomially* to decision problem $Q_2$, if the following condition holds:

If there exists a polynomial algorithm $A_2$ for problem $Q_2$, then there exists a polynomial algorithm $A_1$ for problem $Q_1$.

This is called a *Turing reduction*, denoted by $Q_1 \preceq_T Q_2$.

**Definition 2.7.b.** Decision problem $Q_1$ *transforms polynomially* to decision problem $Q_2$, if there exists an algorithm that transforms an instance $y_1$ of problem $Q_1$ to an instance $y_2$ of problem $Q_2$ such that

1) the transformation algorithm is polynomial in the size of $y_1$
2) $y_1$ is a *yes* instance of $Q_1$ if and only if $y_2$ is a *yes* instance of $Q_2$

This is called a *Karp reduction* or *polynomial transformation*, denoted by $Q_1 \preceq Q_2$.

Open question: Are the definitions 2.7.a and 2.7.b equivalent? Obviously the latter condition 2.7.b. implies the former 2.7.a.

**Definition 2.8.** Decision problem $Q$ is *NP-hard*, if every problem of the class NP reduces polynomially to it.

**Definition 2.9.** Decision problem $Q$ is *NP-complete*, if $Q \in$ NP and $Q$ is NP-hard. The class of NP-complete problems is denoted by NPC.

In other words, problem $Q$ is NP-complete, if

1) $Q \in$ NP i.e. its solution can be verified in polynomial time
2) every problem in the class NP reduces polynomially to it.

NPC is an equivalence class: every NP-complete problem reduces polynomially to every other NP-complete problem.

**Theorem 1.1.** PROPERTIES OF NP-COMPLETE PROBLEMS:

(1) If there exists a polynomial algorithm to one NP-complete problem, then there is a polynomial algorithm to every NP-complete problem, that is,

EITHER all NP-complete problems are polynomially solvable OR none of them is.

(2) If decision problem $Q$ is NP-complete, then $Q \in$ P if and only if $NP = P$.

(3) Assume $Q$ is NP-complete. If $R \in$ NP and $Q \preceq R$, then $R$ is NP-complete.

No polynomial algorithm has been found so far to any NP-complete problem. Neither has it been proved that they didn't exist. "Is $P = NP$?" is an open question.

The first provably NP-complete problem was the *Satisfiability Problem* (SAT):
SAT
Input: Boolean expression, that consists of literals from a set of boolean (logical) variables $x_1, \ldots, x_n$ and their complements $\overline{x}_i$, connectives $\land$ (AND) and $\lor$ (OR) and parentheses ($,$).
Question: Is the expression satisfiable i.e. is there a truth value assignment of the variables such that the expression is TRUE?

S.A. Cook (1971) proved that every decision problem in NP reduces polynomially to the satisfiability problem SAT. Problem SAT belongs to NP because if a correct truth value assignment for a satisfiable expression is given, the truth value can be calculated with a number of steps that is polynomial in the length of the expression. Thus, SAT is NP-complete.

Example of a satisfiable boolean expression:
$$(x_1 \lor x_2 \lor x_3) \land (\overline{x}_1 \lor x_3 \lor x_4) \land (x_1 \lor x_2 \lor \overline{x}_4) \land (\overline{x}_1 \lor \overline{x}_2 \lor \overline{x}_4)$$
One truth value assignment that makes the expression TRUE ($=1$):
$x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 1$.

It can be proved that every boolean expression $C$ can be formulated in a conjunctive normal form
$$C = c_1 \land c_2 \land \ldots \land c_m,$$
in which the clauses $c_i$ are disjunctions of the literals i.e. of the form $y_1 \lor y_2 \lor \ldots \lor y_m$, $y_j \in \{x_1, x_2, \ldots, x_n, \overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n\}$.

Other NP-complete decision problems:
- Traveling Salesman Problem (TSP)
- Hamiltonian Cycle Problem
- Chinese Postman Problem in mixed graphs
- Knapsack Problem
- Bin Packing Problem
- Set Covering
- Clique Problem
- Vertex Cover Problem
- Vertex Coloring Problem
- Partition Problem
- Integer Linear Programming (ILP), Binary Integer Programming (BIP)

The above problems have been proved NP-complete by reductions (polynomial transformation from a known NPC problem) and making use of transitivity property of reducibility: If $P \preceq Q$ and $Q \preceq R$, then $P \preceq R$. In addition, membership in NP must be assured.

Starting from the Satisfiability Problem, reduction chains $SAT \preceq Q_1 \preceq Q_2 \ldots$ have been derived. Because every NP-problem reduces polynomially to SAT it makes all the problems in the chain $Q_1, Q_2, \ldots$ NP-complete (when they have been shown to be members of NP).

Example 2.2. Prove that the following decision problem CLIQUE is NP-complete:

CLIQUE
Input: Graph $G=(V,E)$ and integer $k$.
Question: Does there exist clique of size $k$ in the graph i.e. a completely connected subgraph of $k$ vertices?

Proof:
1) CLIQUE $\in$ NP, because if a subset of vertices is given as a guess, it can be checked in a polynomial time that the number of vertices is $k$ and they are all pairwise connected.
2) Next we show that SAT $\propto$ CLIQUE by showing that any instance of a satisfiability problem (in a conjunctive normal form) can be polynomially transformed to an instance of the CLIQUE problem so that the SAT-instance is satisfiable if and only if a clique of specified size exists in the constructed graph.

Let a set of boolean variables be $X = \{x_1, x_2, ..., x_n\}$ and the given expression in a conjunctive normal form

$$C = c_1 \land c_2 \land ... \land c_k$$

Construct a graph $G = (V,E)$, with

vertex set $V = \{(\alpha,i) \mid \alpha \text{ is a literal in clause } c_i\}$,

edge set $E = \{[(\alpha,i), (\beta,j)] \mid i \neq j \text{ and } \alpha \neq \beta\}$.

The number of vertices $|V|$ is of the same order as length of the expression $C$ and the number of edges $|E|$ is of order $|V|^2$, so the transformation can be executed in polynomial time with respect to the size of the instance of SAT.

The question for the CLIQUE problem will be:
Is there a clique of size $k$ (the number of clauses) in our graph?

"SAT yes $\rightarrow$ CLIQUE yes":
Assume $C$ is satisfiable with a given truth value assignment. Then every expression $c_i$ is TRUE, so each $c_i$ must have at least one literal with value TRUE. Let $\alpha_{j(i)}$ be a true literal of clause $c_i$. Then vertices $(\alpha_{j(1)},1), (\alpha_{j(2)},2), ..., (\alpha_{j(k),k})$ constitute a clique of size $k$.

Explanation: When $i \neq m$, there is an edge between vertices $(\alpha_{j(i)},i)$ and $(\alpha_{j(m)},m)$, because if both literals are true, $\alpha_{j(i)} \neq \alpha_{j(m)}$.

"CLIQUE yes $\rightarrow$ SAT yes":
Assume a $k$-clique exist in $G$. Then every vertex in the clique corresponds to a unique clause $c_i$ with its literal $\alpha_{j(i)}$. Define a truth value assignment by setting the value TRUE to these literals corresponding to the members of the clique, FALSE to their complements and arbitrary values to the nonexistent variables. Then every clause $c_i$ is TRUE and there are no conflicting clauses because $\alpha_{j(i)} \neq \alpha_{j(m)}$ for all pairs of vertices in the clique.

We have proved that the boolean expression $C = c_1 \land c_2 \land ... \land c_k$ is satisfiable if and only if the corresponding graph $G(V,E)$ has a $k$-clique.

Because SAT is NP-complete, so is CLIQUE, by Theorem 2.1.(3). □

COMPLEXITY OF OPTIMIZATION PROBLEMS

NP-completeness was originally defined for decision problems. Many of the decision problems originate from an optimization problem: the decision problem is a threshold value analog of the optimization problem.

As mentioned in the Chapter 2.2, p.13, if the optimization problem is solved the solution to the decision problem is straightforward. For instance, if an optimization algorithm gives the optimal TSP tour, we can compute the length of the tour and compare it to the given threshold value $L$ in polynomial time.
That is, if the optimization problem could be solved by a polynomial algorithm, then we could solve the decision analog in polynomial time. So all decision problems in NP reduce polynomially to an optimization version of an NP-complete problem. By the definition of Turing reduction, we can say that an optimization version of an NP-complete decision problem is NP-hard.

If a polynomial solution algorithm to one of these NP-hard optimization problems would exist, then all decision problems in the class NP would be solved in polynomial time.

Optimization problems are at least as hard as their decision counterparts. Now consider the opposite question: does an optimization problem reduce polynomially to the corresponding decision problem? Here we use the Turing reduction because we cannot use polynomial transformation.

The evaluation version is first reduced to the decision problem by so called binary search: Suppose the optimum value \( f^* \) is an integer between \([0, 2^{2p(n)}]\), where \( p(n) \) is a polynome of the instance size \( n \). The value \( f^* \) can be solved by a binary search, using \( p(n) \) calls of the decision algorithm.

**Example 2.3.** Suppose an upper bound for a TSP tour is \( 32768 = 2^{15} \) (a large enough bound can be easily determined). A given algorithm \( A_Q \) will answer the question "Does the graph contain a cycle through all the vertices with total length \( \leq L \)?"

Set \( L := 32768 / 2 = 16384 \).
If the answer is yes, set \( L := 16384 / 2 = 8192 \).
If the answer is no, set \( L := 16384 + (16384 / 2) = 24576 \).

Continuing the process, \( f^* \) will be found in \( \log_2(32768) = 15 \) calls of the algorithm.

There is no general procedure for reducing the optimization problem to the evaluation problem, the method depends on the problem. In some cases a technique like dynamic programming can be used to determine the solution \( x^* \) by means of the optimum value \( f^* \).

**Example 2.4.** The Knapsack problem: choose a subset of given \( n \) object with weights \( w_1, \ldots, w_n \) and values \( v_1, \ldots, v_n \) such that the total weight does not exceed the capacity \( M \) of the knapsack and the total value is maximized.

Decision variables:
\[
x_i = \begin{cases} 
1 & \text{if object } i \text{ is chosen} \\
0 & \text{if not}
\end{cases}
\]

A binary linear optimization model:
\[
\begin{align*}
\text{max} \quad f &= v_1x_1 + v_2x_2 + \ldots + v_nx_n \\
\text{s.t.} \quad w_1x_1 + w_2x_2 + \ldots + w_nx_n &\leq M \\
x_i &= 0 \text{ or } 1
\end{align*}
\]

Suppose an evaluation algorithm \( A_f \) solves the maximum \( f \) value \( f^* \) of the knapsack.

Set the value of each variable in turn to \( x_i = 0 \) and \( x_i = 1 \) and solve the knapsack values for the remaining \( n-1 \) variables using capacities \( M \) and \( M-w_i \), respectively. Let the objective function values be \( f_1 \) (for \( x_i = 0 \)) and \( f_2 \) (for \( x_i = 1 \)). If \( f_1 = f^* \), then \( x_i^* = 0 \). If \( f_2 + v_i = f^* \), then \( x_i^* = 1 \). The optimum solution vector \( x^* \) is determined by \( 2n \) executions of the evaluation algorithm \( A_f \).

Alternative: After fixing \( x_i \), modify the problem for the remaining variables and solve for \( x_i = 0 \) and \( x_i = 1 \), etc. In each subsequent problem set \( M \) as the remaining capacity after the fixed variables.
If the evaluation algorithm $A_t$ were be polynomial, then the optimization algorithm would be polynomial. We have shown that the optimization problem Knapsack reduces polynomially to the evaluation problem. And because of transitivity of reduction, the optimization problem reduces polynomially to the decision problem version of Knapsack.

Most of the NP-complete optimization problems can be polynomially reduced to the corresponding decision problems. In the following example TSP optimization problem is reduced to the corresponding decision problem.

**Example 2.5.** TSP with $n$ vertices, distances given in $n \times n$ integer matrix $C$, a threshold value $L$. Suppose that an algorithm $\text{TSPDECISION}(n,C,L)$ solves the decision problem: "Does there exist a TSP tour with length $\leq L$?"

The following algorithm solves the optimal TSP tour using subroutine $\text{TSPDECISION}$. The first part solves the optimum tour length by a binary search. The other part tests for each edge if that edge belongs to the optimum tour. In the resulting matrix $C$, the edges that keep their original values $c_{ij}$ belong to the optimal tour. For other edges, $c_{ij} = n \max \{c_{ij}\} + 1$.

Algorithm $\text{TSPTOUR}(n,C)$

1. $\text{low} := 0$
2. $\text{high} := n \max \{c_{ij}\}$
3. while $\text{low} \leq \text{high}$ do
   1. $L := \lfloor (\text{low} + \text{high})/2 \rfloor$
   2. if $\text{TSPDECISION}(n,C,L) = \text{'yes'}$
      then $\text{high} := L$
      else $\text{low} := L + 1$
   4. $\text{opt} := \text{high}$
   5. for $i := 1$ to $n$ do
      1. for $j := 1$ to $n$ do
         1. $\text{remember} := c_{ij}$
         2. $c_{ij} := n \max \{c_{ij}\} + 1$
         3. if $\text{TSPDECISION}(n,C,\text{opt}) = \text{'no'}$
            then $c_{ij} := \text{remember}$

The subroutine $\text{TSPDECISION}$ is called $\lceil \log_2 (nc_{\text{max}}) \rceil + n^2$ times which is polynomial in the size of the input.

We have seen that the optimization problems associated with certain NP-complete decision problems are Turing reducible to their decision counterparts, and vice versa. We use the term NP-complete also for these optimization problems.

**HOW TO SOLVE AN NP-COMPLETE PROBLEM: GENERAL PRINCIPLES**

When confronted with a known NP-complete problem, large or small, a solution is needed anyway. For instance, a vehicle routing plan or a flight crew schedule must be available for the decision makers.

1) **Exact exponential algorithms**
   - Worth trying if an implemented algorithm is readily available.
   - May be exponential in the worst case, but reasonable in the average (e.g. several branch and bound methods).
   - If a so called pseudopolynomial algorithm exists, it is preferable although not polynomial.
2) **Efficient approximation algorithms**
   • Polynomial time methods for which a bound for the objective value can be given.
   • With good luck, optimal solution may be found.

3) **Heuristics**
   • Based on reasonable principles and rules of thumb but cannot guarantee optimum.
   • Should be fast and easy to implement.
   • If the solution is not satisfactory, use it as a starting point for a higher level method.
   • A heuristic belongs to the previous category if a bound for the solution value can be given.

4) **Tractable special cases**
   • Some of the NP-complete problems have special cases that are polynomially solvable. Does your problem have special characteristics or structure that make it easier? E.g. the clique problem for planar graphs can be decided efficiently, because a planar graph cannot have a clique of size $\geq 5$.

5) **Tractable relaxations**
   • In these modified problems, some of the restrictions of the original problem have been dropped to make the problem tractable.
   • The solution set includes the solutions of the original problem.
   • The solution to the relaxed problem can be used as a starting point.