1 NP-Complete Problems and General Strategy

In the last lecture, we defined two classes of problems, $P$ and $NP$. While $P \subseteq NP$, it is still an open question whether $NP \subseteq P$. We recognized a special class of problems inside $NP$, which are called $NP$-complete problems. They are the fundamental problems to tackle in order to solve $P$ vs $NP$. We gave three examples of $NP$-complete problems (proof omitted): SAT, Partition, and 3-Partition. Our goal in this lecture is to recognize other $NP$-complete problems based on Partition and SAT problems.

There is a general strategy to show that a problem $B$ is $NP$-complete. The first step is to prove that $B$ is in $NP$ (which is usually easy) and the second step is to prove there is an $NP$-complete problem $A$ such that it has a polynomial reduction to problem $B$. In general, the difficulties lie in the second step.

2 Knapsack

Let us recall the decision version of the Knapsack problem: given $n$ items with size $s_1, s_2, ..., s_n$, value $v_1, v_2, ..., v_n$, capacity $B$ and value $V$, is there a subset $S \subseteq \{1, 2, ..., n\}$ such that $\sum_{i \in S} s_i \leq B$ and $\sum_{i \in S} v_i \geq V$?

**Theorem 1** Knapsack is $NP$-complete.

**Proof:** First of all, Knapsack is $NP$. The proof is the set $S$ of items that are chosen and the verification process is to compute $\sum_{i \in S} s_i$ and $\sum_{i \in S} v_i$, which takes polynomial time in the size of input.

Second, we will show that there is a polynomial reduction from Partition problem to Knapsack. It suffices to show that there exists a polynomial time reduction $Q(\cdot)$ such that $Q(X)$ is a ‘Yes’ instance to Knapsack if $X$ is a ‘Yes’ instance to Partition. Suppose we are given $a_1, a_2, \ldots, a_n$ for the Partition problem, consider the following Knapsack problem: $s_i = a_i, v_i = a_i$ for $i = 1, \ldots, n$, $B = V = \frac{1}{2} \sum_{i=1}^{n} a_i$. $Q(\cdot)$ here is the process converting the Partition problem to Knapsack problem. It is clear that this process is polynomial in the input size.

If $X$ is a ‘Yes’ instance for the Partition problem, there exists $S$ and $T$ such that $\sum_{i \in S} a_i = \sum_{i \in T} a_i = \frac{1}{2} \sum_{i=1}^{n} a_i$. Let our Knapsack contain the items in $S$, and it follows that $\sum_{i \in S} s_i = \sum_{i \in S} a_i = B$ and $\sum_{i \in S} v_i = \sum_{i \in S} a_i = V$. Therefore, $Q(X)$ is a ‘Yes’ instance for the Knapsack problem.

Conversely, if $Q(X)$ is a ‘Yes’ instance for the Knapsack problem, with the chosen set $S$, let $T = \{1, 2, \ldots, n\} - S$. We have $\sum_{i \in S} s_i = \sum_{i \in S} a_i \leq B = \frac{1}{2} \sum_{i=1}^{n} a_i$, and $\sum_{i \in S} v_i = \sum_{i \in S} a_i \geq V = \frac{1}{2} \sum_{i=1}^{n} a_i$. This implies that $\sum_{i \in S} a_i = \frac{1}{2} \sum_{i=1}^{n} a_i$ and $\sum_{i \in T} a_i = \sum_{i=1}^{n} a_i - \frac{1}{2} \sum_{i=1}^{n} a_i = \frac{1}{2} \sum_{i=1}^{n} a_i$. 

25-1
Therefore, \( \{S, T\} \) is the desired partition, and \( X \) is a ‘Yes’ instance for the Partition problem. This establishes the \( \text{NP} \)-completeness of Knapsack problem.

\[\square\]

**Remark 1** In the previous lecture, we showed that Knapsack problem can be solved using dynamic programming with running time \( O(n^3B^2) \), where \( n \) is the number of items and \( B \) is the capacity. Since our input is binary, \( nB^2 \) is exponential in the input size \( (B = 2^k \log B) \), thus DP does not provide a polynomial running time algorithm.

On the other hand, if the given input to the Knapsack problem is unary rather than binary (that is, we encode a 5 as 11111), then DP provides a polynomial running time algorithm. We call such algorithms pseudo-polynomial time algorithms.

Hence, we see that Knapsack is not \( \text{NP} \)-complete if the given input is unary (assuming \( \text{P} \neq \text{NP} \)), but \( \text{NP} \)-complete when the given input is binary. Such problems are called **weakly \( \text{NP} \)-complete**. However, some problems (like 3-Partition) are \( \text{NP} \)-complete even if the given input is unary. We call such problems **strongly \( \text{NP} \)-complete**.

### 3 3-SAT

The SAT problem is the following: given \( n \) boolean variables \( x_1, x_2, ..., x_n \), \( m \) clauses (e.g. \( x_1 \lor \overline{x}_3 \lor x_7 \)), is there an assignment of true/false to the \( x_i \), such that all clauses are satisfied? 3-SAT problem is a special case of SAT problem in the sense that each clause contains at most 3 variables.

**Theorem 2** 3-SAT is \( \text{NP} \)-complete.

**Proof:** First of all, since 3-SAT problem is also a SAT problem, it is \( \text{NP} \). We now show that there is a polynomial reduction from SAT to 3-SAT.

Given \( m \) clauses in the SAT problem, we will modify each clause in the following recursive way: while there is a clause with more than 3 variables, replace it by two clauses with one new variable. The tree below is an example of this process, and we will use it for the demonstration of proof. The new 3-SAT problem contains all the clauses corresponding to the leaves, they are \( x_1 \lor \overline{x}_2 \lor z \lor \overline{x}_3 \lor x_4 \lor \overline{x}_5 \).

\[
\begin{array}{c}
\begin{array}{c}
\overline{x}_1 \lor \overline{x}_2 \lor \overline{x}_3 \lor x_4 \lor \overline{x}_5 \\
x_1 \lor \overline{x}_2 \lor z \\
\overline{z} \lor \overline{x}_3 \lor x_4 \lor \overline{x}_5 \\
\overline{z} \lor \overline{x}_3 \lor w \\
w \lor x_4 \lor \overline{x}_5
\end{array}
\end{array}
\]

We first observe that this reduction process is polynomial in the input size. For a clause consisting of \( k \) variables, we can build a tree recursively until each leaf is a clause consisting of exactly 3 variables. At \( i \)-th level of the tree, the clause corresponds to rightest node at each level contains one less variable than the previous layer. Hence, the tree has \( k - 3 \) layers in total. This implies that we will construct \( k - 2 \) new clauses that consists of exactly 3 variables for each clause.
that consists of $k$ variables in the SAT problem. Suppose the original SAT problem has $m$ clauses, with $k_1, \ldots, k_m$ variables respectively, we will construct a 3-SAT problem with $\sum_{i=1}^{m}(k_i - 2)$ clauses. And this procedure takes $O(2\sum_{i=1}^{m}(k_i - 2))$ steps, which is polynomial in the size of input.

For the final step, we claim that the original SAT problem is a ‘Yes’ instance iff the constructed 3-SAT problem is a ‘Yes’ instance. The key property here is that each step (during the tree construction) maintains satisfiability, i.e, the clauses at level $i$ can be satisfied iff the clauses at level $i + 1$ can be satisfied. For a demonstration, we will use the above tree. First suppose that $x_1 \lor \bar{x}_2 \lor \bar{x}_3 \lor x_4 \lor \bar{x}_5$ is true, then either $x_1 \lor \bar{x}_2$ is true or $\bar{x}_3 \lor x_4 \lor \bar{x}_5$ is true. In the previous case, set $z = False$, and in the latter case, set $z = True$. We see that with this assignment, both $x_1 \lor \bar{x}_2 \lor z$ and $\bar{z} \lor \bar{x}_3 \lor x_4 \lor \bar{x}_5$ are satisfied. Conversely, if both $x_1 \lor \bar{x}_2 \lor z$ and $\bar{z} \lor \bar{x}_3 \lor x_4 \lor \bar{x}_5$ are satisfied, then if $z = True$, we know that $\bar{x}_3 \lor x_4 \lor \bar{x}_5$ is true; if $z = False$, we know that $x_1 \lor \bar{x}_2$ is true. Both imply that the original clause is true.

This property allows us to prove the general claim. For $\Rightarrow$ direction, we start from the root of the tree and use the satisfiability property to deduce that all the clauses at the leaves can be satisfied. For $\Leftarrow$ direction, we start from the leaves and use the satisfiability property to show that the root clause can be satisfied as well. This completes the proof that 3-SAT problem is NP-complete. \qed

4 Independent Set Problem

Given a graph $G = (V, E)$, an independent set (IS) $S$ is a subset of $V$ such that for all $i, j \in S$, $(i, j) \notin E$. The maximum IS problem is to find an independent set of maximum size, and the decision version of this problem is the following: give $G = (V, E)$, is there an independent set of size at least $B$?

**Theorem 3** IS is NP-complete.

**Proof:** First of all, IS is NP with proof $S$. The verification process consists of checking all possible pairs in $S$ and checking $|S| = B$. It takes $\binom{B}{2} + 1$ steps, which is polynomial in the size of the input.

Secondly, we claim that there is a polynomial-time reduction from 3-SAT problem to IS problem. The construction is the following: give a 3-SAT problem with $m$ clauses, we draw $m$ triangles with nodes representing the literals appearing in the clause. Then we connect each node corresponding to a literal $x_i$ with each node corresponding to a literal $\bar{x}_i$, for all $i$. For example, consider a 3-SAT problem: $x_1 \lor \bar{x}_2 \lor x_3, x_1 \lor x_2 \lor \bar{x}_4, x_1 \lor \bar{x}_3 \lor x_5$, we will convert it to the following graph:

![Graph](image)

Note that the newly constructed graph $G$ consists of $3m$ nodes, and at most $3m + \binom{3m}{2}$ edges. Hence, the reduction process takes time polynomial in the input size. Moreover, we claim that a
3-SAT problem is a ‘Yes’ instance iff \((G, m)\) is a ‘Yes’ instance to the IS problem. Suppose that 3-SAT is satisfiable, then for each triangle, we choose one node such that the corresponding literal satisfies the clause (that is, \(x_i\) is set true or \(\overline{x}_i\) is set false). For any two nodes we choose, they are from two different triangles. If they are connected to each other, they are \(x_i\) and \(\overline{x}_i\) for some \(i\) by our construction. But it is not possible that \(x_i = True\) and \(\overline{x}_i = True\). Hence, the set of nodes we choose forms an independent set of size \(m\).

Conversely, if we have an independent set of size \(m\) for \(G\), then each node must come from different triangles since each triangle is connected. For each node we choose, if it corresponds to variable \(x_i\) for some \(i\), we set \(x_i = True\); if it corresponds to \(\overline{x}_i\) for some \(i\), then we set \(x_i = False\); the assignment is consistent because we cannot have both \(x_i\) and \(\overline{x}_i\) in the independent set because they are joined by an edge. This assignment satisfies the clause corresponding to each triangle. This shows that the 3-SAT instance is satisfiable. \(\square\)