Chapter G3: NP-Completeness

While humanity cannot determine whether P = NP or not, we can, however, identify problems that are the hardest in NP. These are called the NP-complete problems. They have the property that if there is a polynomial-time algorithm for any one of them, then there is a polynomial-time algorithm for every problem in NP.

G3.1 Reductions

For decision problems A and B, A is said to be polynomial-time reducible to B (written $A \leq_p B$) if there is a polynomial-time computable function $f$ such that

\[ q \text{ is a Yes-instance of } A \iff f(q) \text{ is a Yes-instance of } B \]

That is, $f$ translates questions about $A$ into questions about $B$ while preserving the answer to the question.

The key result:

**Lemma.**

a) If $A \leq_p B$ and $B$ in P, then $A$ in P.
b) If $A \leq_p B$ and $A$ not in P, then $B$ not in P.

**Proof.** Suppose the reduction from $A$ to $B$ is given by the function $f$ which is computable in $O(n^k)$ time. And suppose we can decide questions about $B$ in $O(n^\ell)$ time. Then we build a polynomial-time decider for $A$ as follows. It takes the input $q$, computes $f(q)$ and then sees whether $f(q)$ is a Yes-instance of $B$ or not. Does the program run in polynomial-time? Yes. If $q$ has length $n$ then the length of $f(q)$ is at most $O(n^k)$ (since a program can only write one symbol each step). Then the test about $B$ takes $O(n^{k\ell})$ time. And that’s polynomial. \(\diamondsuit\)

G3.2 NP-completeness

We need a definition:

A decision problem $S$ is defined to be NP-complete if

a) It is in NP; and
b) For all $A$ in NP it holds that $A \leq_p S$.

Note that this means that:

- If $S$ in NP-complete and $S$ in P, then P=NP.
- If $S$ is NP-complete and $T$ in NP and $S \leq_p T$, then $T$ is NP-complete.
G3.3  Examples

There are tomes of NP-complete problems. The standard method to proving NP-completeness is to take a problem that is known to be NP-complete and reduce it to your problem. What started the whole process going was Cook’s original result:

**Theorem.** SAT is NP-complete.

We omit the proof. Some more examples:

- The 3SAT problem is NP-complete:

  **3SAT**
  
  Input: $\phi$ a boolean formula in conjunctive form with three literals per clause
  
  Question: Is there a satisfying assignment?

- HAMPATH is NP-complete.

- Domination. A set of nodes in a graph is a dominating set if every other node is adjacent to at least one of these nodes. For example, in the graph on page 6, $\{A, D\}$ is a dominating set.

  The DOMINATION problem is NP-complete:

  **DOMINATION**
  
  Input: Graph $G$ and integer $k$
  
  Question: Does there exist a dominating set of $G$ of at most $k$ nodes?

To show a new problem is NP-complete, one shows that the problem is in NP, and then provides a reduction from a known NP-complete problem. There have been probably more than a million such proofs made.

**Exercises**

1. Show that the independence number of a graph with maximum degree 2 can be computed in polynomial time.

2. Show that if P=NP then there is a polynomial-time algorithm which on input a graph finds a hamiltonian path if one exists. (Note: this is not immediate.)

3. Show that if P=NP then there is a polynomial-time algorithm which on input $\phi$ finds a satisfying assignment if one exists.