

582206 Models of Computation (Autumn 2008)

Additional exercises

The problems are related to the material covered in the last lecture, for which there will be no regular exercise session. Solving these is recommended, as you are expected to know this material in the examination. Solutions will appear on the course web page on Friday, 5 December.

1. Find the definitions of the following NP-complete problems from the literature: (a) travelling salesman problem (TSP); (b) knapsack problem; and (c) set covering problem. You can find these problems in most textbooks of computational complexity or algorithm theory, but for the purpose of this exercise the Internet is a quite acceptable source, too.
2. An *independent set* in an undirected graph $G = (V, E)$ is any set of nodes $I \subseteq V$ such that no two nodes in I are connected by an edge (ie, $(I \times I) \cap E = \emptyset$). Define a formal language

$$\text{INDEPENDENTSET} = \{ \langle G, k \rangle \mid \text{graph } G \text{ has an independent set of size } k \}.$$

Show that $\text{INDEPENDENTSET} \in \text{P}$ if and only if $\text{CLIQUE} \in \text{P}$.

Hint: Assume some procedure p solves the clique problem. Solve the independent set problem in graph (V, E) by applying p in the complement graph $(V, \overline{E}) = (V, (V \times V) - E)$.

3. By definition, NP-complete problems are formal languages, or in practice *decision problems* (yes/no questions). For example the clique problem can be written as

Input: undirected graph G , natural number k

Output: “yes” if G has k -clique; otherwise “no”.

With the same basic setting we can also associate the *optimisation problem*

Input: undirected graph G

Output: the size of the largest clique in G

where the answer is a single number. Further, we have the *search problem*

Input: undirected graph G

Output: largest clique in G

where the answer is the actual set of nodes forming the largest clique.

Show the following relationships between the above problems:

- (a) If the decision problem can be solved in polynomial time, then so can the optimisation problem.
- (b) If the optimisation problem can be solved in polynomial time, then so can the search problem.

Hint: Solve the optimisation problem by calling the assumed algorithm for the decision problem several times, slightly varying the input parameters. Similarly solve the search problem using the solution for the optimisation problem as a subroutine.

4. (a) In the context of some application, a computational problem we would like to solve turns out to be undecidable. What does this mean in practice for implementing the application?
(b) As part (a), but instead of undecidable the problem is NP-complete.