Chapter 6

Complexity Theory and Revision

6.1 Complexity Hierarchy

Here is a diagram of the complexity hierarchy:

- **NP**: problems which can be checked in polynomial time
- **P**: problems which can be *solved* in polynomial time.
- **NP-Hard**: problems into which any NP problem can be transformed within poly time.

An NP-Hard problem *is also* NP-Complete if it itself happens to be NP. Some NP-Hard problems are *too hard* to be in NP: solutions require more than poly time to check.

Note that the diagram is somewhat speculative. In particular, I put all of NP-Complete within EXP, whereas, if P=NP, then it all belongs in P.
6.1.1 Checking versus Solving

What is the difference between checking and solving? SAT makes for a clear example:

- In order to check a proposed solution to SAT (assuming CNF), all we need to do is take a given instantiation of the Boolean variables and try out each clause until either we run out of clauses or we find one which fails. In the first case, the instantiation is a solution; in the latter case it is not. This can be done in polynomial time.

- In order to solve a SAT problem, we also need to search the space of possible instantiations and apply our check routine to all such possible solutions we visit in our search. The solution space has the size of the number of bit strings of length \( n \), where \( n \) is the number of Boolean variables. This is, of course, \( 2^n \). Since this problem is NP-Hard, the best known solutions are exponential.

6.2 Analysis of Algorithms

A trouble maker in dealing with analysis of algorithms is the neglect of input, or output, size.

The distinction between input and input size needs to be kept clear. The analysis of algorithms methods, formulae, properties, etc., require the \( n \) terms to be understood as input sizes and not as the inputs themselves. The input size for a problem is, at bottom, the number of characters required to describe its input; hence, if the input is an integer \( N \), then its input size is (approximately) \( n = \log_2 N \), because that's the number of bits required to represent it.

Sometimes the output size is relevant. For example, the argument that SAT is in NP depends upon this. That is,

- It's known that conversion of arbitrary Boolean formulas into CNF is polynomial time in \( n \), the number of Boolean variables.
- Given that, the output number of clauses cannot be greater than a polynomial function of \( n \). Why? Because in a Turing machine one step is required for each output symbol; so, if there were more symbols than any polynomial function of the input size, the computation itself would have to be unbounded by any polynomial function.
- Checking clauses is linear in the size of the clause.
- So, checking a Boolean formula involves a polynomially bounded function, composed with a linearly bounded function. Hence, checking CNF is polynomially bounded.
- Whence it follows that SAT is NP, by definition.

6.3 Exercises

6.3.1 Vocabulary

Before you can begin the exercises, you will need to make sure you understand the vocabulary for this part of the course. Write down a brief definition for each of these phrases, and discuss them in your tutorial.
6.3.2 Complexity theory

(1) Always true/false? Sometimes true/false?

(a) Problems in \( \text{NP} \) are harder than all problems in \( \text{P} \).
(b) NP-Complete problems have no known polynomial-time solutions.
(c) NP-Hard is a subset of \( \text{NP} \).
(d) NP-Complete is a subset of \( \text{NP} \).
(e) No NP-Hard problem is in \( \text{NP} \).
(f) Sorting is an example of a problem in \( \text{NP} \).
(g) SAT is an example of a problem in \( \text{NP} \).
(h) Sorting is an example of a NP-Complete problem.
(i) NP-Complete is a subset of NP-Hard.
(j) NP-Complete is a subset of \( \text{EXP} \).
(k) The Traveling Salesman Problem is NP-Hard.
(l) For all \( Q \in \text{P} \) and \( R \in \text{NP} \), \( Q \) is polynomial time reducible to \( R \).
(m) If \( Q \) is reducible to \( R \), then \( Q \) is easier than \( R \).
(n) 3-SAT has a polynomial-time solution.
(o) NP-Hard problems are computationally infeasible.

(2) You are discussing the material you’ve seen in CSE3305 lectures with a friend who is studying second-year Computer Science. Your friend says, “I don’t see why they make us do this complexity stuff. We’re never going to have to worry about it after we graduate.” Do you agree or disagree with your friend? Come up with some good arguments for or against studying complexity in undergraduate Computer Science courses. If there’s time, hold a class debate.

(3) (a) Write a pseudo-code algorithm for testing whether a Boolean formula is satisfiable.
(b) Analyse the worst-case time complexity of your algorithm using \( O \) notation.

(4) This year you are given a new computer as a birthday present; we can call it the Birthday Machine. It has a machine architecture based upon a functional programming language (recursion); furthermore it has a very a special property: each recursion takes half the time it took on the previous recursion. This property resets whenever a new recursive function is initiated. You can interpret this as: when a function is called, if it is the same function as was last invoked, then the execution speed of the machine is doubled; otherwise, the execution speed is reset to the original speed. (Wouldn’t you like this story to be real?) Determine the \( O \)-notation worst-case time complexity of the following algorithms on the Birthday Machine:
(a) Binary search
(b) Your algorithm for testing satisfiability from the problem above.

(5) Prove that \( P \subset EXP \).

6.4 Revision Exercises

(1) For each of the following problems, write a C data structure that can store problem state:
- FreeCell
- peg solitaire
- N-Queens
- Rubik’s Cube

For each of these problems:
(a) describe the search space
(b) compute its size
(c) define a useful neighbourhood
(d) write a perturb() function, in pseudocode if you prefer, that moves to a random point in the neighbourhood
(e) describe a possible heuristic evaluation function

(2) You are writing a program that simulates the traffic flow in the Campus Centre. Choose good probability distributions to model the following events:
(a) the times that customers arrive at the Information Desk
(b) the speed at which each pedestrian walks through the corridors  
(c) the length of time between customers arriving at the noodle bar  
(d) the number of items a customer wishes to buy at the Bookshop  
(e) the length of time needed to serve a customer at the sandwich bar  

(3) This is also a good time to go over any problems you didn’t cover on previous tutes.

Acknowledgements

Robyn A. McNamara for preparation of tutorial materials.