The Theory of Computation

Bernard M.E. Moret

Addison-Wesley-Longman, 1998

Solution Manual
Chapter 7, Problem 7.34

There are really two versions of this problem, depending on whether the pair $(M, p())$ is fixed for the problem or is part of each instance. If the pair is fixed for the problem, then each pair $(M, p())$ defines a different problem; and different problems have different complexity. For instance, a checker for *SAT* that expects a truth assignment specified by a binary string defines a problem where every possible string is a valid certificate for some "yes" instance of *SAT*. In this case, the complexity of the problem is constant—the solution algorithm is simply `print "yes"`. The relevant question for this version of the problem is simply "how difficult can any of these problems be?" Checking that the string is in the proper syntactic form (that it follows the encoding expected by the checker) is easily done in polynomial time—indeed, any reasonable encoding scheme must be such that the decoding can be done in time polynomial in the length of the code. So, given a syntactically correct string, must this string be a valid certificate? and if not, can we decide whether it is valid or not in polynomial time? The answer to the first question remains unknown, but the answer to the second is yes, as can be deduced from our reasoning about the more general problem below.

The other version, in which the checker and its bound are part of the instance, defines a single problem; it was the intended problem and is more interesting. Since we do not know for what particular NP problem the checker is intended, we cannot easily reconstruct an instance for what we may believe to be a valid certificate, nor can we determine even as much as the syntactic correctness of a string—at least not without analyzing the checker program and determining its properties, something that is in general undecidable. We appear, therefore, to be reduced to using non-determinism to guess an instance x and to feeding the pair (x, y) to the checker and waiting for the answer. Even this step is not obvious: the checker takes at most $p(|x|)$ time to carry out its computation, but our input is y, not x: can we translate the polynomial bound in $|x|$ into some bound in $|y|$? Unfortunately, while we know that $|y|$ is bounded by a polynomial function of $|x|$, we have no such guarantee in the reverse direction.

However, we can make some useful observations. First, if there exist several instances x for which the same y is a valid certificate, then we can always consider the shortest of these instances (For instance, given any graph with vertex cover of size K, we can always add any number of independent vertices to the graph without changing the solution and thus without changing the certificate, at least in some encodings; given the certificate, though, we can reconstruct a graph for it with at most K edges and thus concise in the size of the instance. Similarly, given a satisfiable Boolean formula, we can always add any number of redundant variables by padding the clauses as seen in the transformation for *3SAT*; while a concise certificate need not include the truth values of these redundant variables and thus can appear increasingly small, we can always reconstruct from such a certificate of length n an instance of *Satisfiability* with n 1-literal clauses.) Secondly, given an NP-complete problem, non-trivial instances in which all "fluff" (such as unconnected vertices in most graph problems) has been eliminated cannot really have very short certificates; if their certificates had length $O(\log n)$, for instance, we could try them all in deterministic polynomial time and thus show $\mathrm{P} = \mathrm{NP}$. We conclude that we must have $|y| = \omega(\log |x|)$, or $|x| = o(c^{|y|})$, so that our problem is at least in NEXP—not a particularly encouraging result.

To tighten our conclusion, we must revisit the examination of certificates. We want to show that, for each valid certificate, there exists at least one instance that is concise in the length of the certificate. For that, we can appeal to Exercise 6.19 about honest polynomial functions, in which we used a well-behaved pairing function to produce the instances of an NP problem—along with their certificates. In that pairing, the certificate which serves to validate the instance is of size comparable to the instance, at least when the output is produced honestly. Thus we can conclude that, for any problem in NP, valid certificates can be used to certify "yes" instances of polynomially related sizes. This observation allows us to place our problem in NP: guess an instance of polynomially related size and feed it along with the certificate to the checker, which will then run in time polynomial in both arguments. It does not appear that we can do better, even if all syntactically correct strings are valid certificates, since we cannot check for syntactic correctness without running the checker and since the checker must have both an instance and a putative certificate in order to run.