The Theory of Computation

Bernard M.E. Moret

Addison-Wesley-Longman, 1998

Solution Manual

Chapter 7, Problem 7.10

- *Optimal Vertex Cover*. We prove this problem NP-hard by (Turing) reducing the standard *VC* to it. Given a graph, $G = (V, E)$ and a positive integer bound, $B < |V|$, we simply call the oracle for *Optimal Vertex Cover* repeatedly, starting with a bound of B and decreasing by 1 at each call, until we have either obtained an answer of "yes" (in which case we also answer "yes") or have reached a bound of 0 receiving only answers of "no" (in which case we answer "no").
  We prove this problem NP-easy by (Turing) reducing it to the standard *VC*. Given a graph $G = (V, E)$ and a positive integer bound, $B < |V|$, we simply call the oracle for *VC* twice, once with bound B and the second time with bound $B - 1$; if the first answer is "yes" and the second "no", we answer "yes", otherwise we answer "no". Thus the problem is NP-equivalent and also in D^p.
- *Minimal Unsatisfiability*. We prove this problem NP-hard with a many-one reduction from the coNP-complete problem *3UNSAT*. See Exercise 7.46 for details. This is a fairly elaborate reduction and one that does not make use of the power of Turing reduction, but it is surprisingly difficult to derive a simpler, Turing reduction.
  We prove this problem NP-easy by reducing it to the standard *SAT*. We begin with one call with the full instance, to establish that the instance is unsatisfiable. If the instance is indeed unsatisfiable, we proceed with a loop that will remove one clause from the instance and call the decision algorithm to establish that the resulting instance is now satisfiable; this loop proceeds as along as all answers remain positive and eventually ends up testing each clause through removal. If all of these tests are passed, then the instance is a "yes" instance of *Minimal Unsatisfiability*, otherwise it is a "no" instance. The total number of calls to the decision procedure is at most one more than the number of clauses in the instance—and thus is at most linear in the size of the instance.

- *Unique Satisfiability*. We prove this problem NP-hard by devising a many-one polynomial-time reduction from the known coNP-complete problem *UNSAT* to it—a rather tighter reduction than needed. The basic idea is to take the formula for *UNSAT*, call it $f(x_1, \ldots, x_n)$, and turn it into the new formula, $g(x_0, x_1, \ldots, x_n)$ given by

  $$g(x_0, x_1, \ldots, x_n) = (x_0 \wedge x_1 \wedge \cdots \wedge x_n) \vee (\overline{x_0} \wedge f(x_1, \ldots, x_n))$$

  It is easily verified that this new formula g, which is always satisfiable (in itself an interesting result, since it shows that deciding unique satisfiability is hard even when satisfiability is known), is uniquely satisfiable (by setting all $n + 1$ variables to true) if and only if the original formula f is unsatisfiable. Turning this formula, which is not in clause form, in a form suitable for *Unique Satisfiability* can be done in polynomial time, thereby completing our reduction and showing that *Unique Satisfiability* is coNP-hard and thus also NP-hard.
  We prove this problem NP-easy by reducing it to the standard *SAT*. We begin with a single call with the same instance to establish satisfiability. If the answer is "no," we answer "no" and stop. Otherwise, we establish uniqueness as follows. We proceed one variable at a time. At stage i, we have determined the truth values of the first $(i - 1)$ variables and also determined that only this particular combination of truth values can lead to a satisfying truth assignment. Working now with variable x_i, we first test whether there exists a satisfying truth assignment with x_i set to true (by adjusting the clauses to reflect this assignment and calling the oracle on the resulting collection of clauses); if not, we set x_i to false and proceed to the next variable. If there exists a truth assignment, we now test whether there exists a satisfying truth assignment with x_i set to false; if not, we set x_i to true and proceed to the next variable; otherwise, we answer "no" and stop. On establishing that only one truth value works for the last variable, we can answer "yes." The total number of calls to the oracle is at most $2n + 1$ (two per variable plus the initial call) and thus polynomial in the size of the input.
  Actually, two oracle calls to NP problems suffice. Note that the problem "does an instance of *SAT* have at least 2 satisfying assignments?" is NP-complete (it is in NP since the certificate need only give both assignments and it is complete because it is trivial to transform *SAT* in such a way that its number of satisfying assignments is exactly doubled, by introducing one redundant variable). Now we can solve

Unique Satisfiability by one call to *SAT* and one to "does an instance of *SAT* have at least two satisfying assignments?", expecting a "yes" answer to the first question and a "no" to the second. Thus the problem is in fact in D^p.

- *TSP Factor*. We prove this problem NP-hard by reducing the standard *HC* to it. Given a graph, $G = (V, E)$, we create two instances of *TSP Factor*; both have $|V|$ cities and intercity distances $d(i, j)$ of 1 if $i, j \in E$ and of 2 otherwise. In the first call, we ask if $|V|$ is a factor; in the second call, we ask if $2|V|$ is a factor. If the first answer is "yes" and the second "no", we answer "yes", otherwise we answer "no". (The TSP tour must have length at least $|V|$ and at most $2|V|$; it has length $|V|$ if and only if the *HC* instance admits a solution. Our second call eliminates tours of length $2|V|$ for graphs where $|E| = \emptyset$.)
  We prove this problem NP-easy by reducing it to the standard *TSP*. We can find the value of the optimal tour, l_{opt}, by binary search with at most $\log l_{\text{opt}}$, which is polynomial (in fact, linear) in the input size. Once l_{opt} is known, we can check whether or not the given number is a factor of it in time polynomial in $\log l_{\text{opt}}$, using any standard division algorithm. There seems to be no way of avoiding the linear number of calls; indeed, the problem can be proved to be complete for Δ_2^p and thus unlikely to be in D^p like the first three.