

## The NP-Completeness Column: An Ongoing Guide

DAVID S. JOHNSON

*AT&T Bell Laboratories, Murray Hill, New Jersey 07974*

This is the fifteenth edition of a quarterly column that covers new developments in the theory of NP-completeness. The presentation is modeled on that used by M. R. Garey and myself in our book "Computers and Intractability: A Guide to the Theory of NP-Completeness," W. H. Freeman & Co., New York, 1979 (hereinafter referred to as "[G&J]"; previous columns will be referred to by their dates). A background equivalent to that provided by [G&J] is assumed, and, when appropriate, cross-references will be given to that book and the list of problems (NP-complete and harder) presented there. Readers who have results they would like mentioned (NP-hardness, PSPACE-hardness, polynomial-time-solvability, etc.) or open problems they would like publicized, should send them to David S. Johnson, Room 2C-355, AT&T Bell Laboratories, Murray Hill, NJ 07974 (CSNET address: dsj.btl@csnet-relay). Please include details, or at least sketches, of any new proofs; full papers are preferred. If the results are unpublished, please state explicitly that you would like them to be mentioned in the column. Comments and corrections are also welcome. For more details on the nature of the column and the form of desired submissions, see the December 1981 issue of this Journal.

### 1. INTRODUCTION

Novices in the theory of NP-completeness often ask a question that goes something like this: "We know that it's hard to find Hamiltonian circuits, but that's because it's NP-complete even to tell whether such circuits exist. Suppose this latter difficulty is removed. Could there be an algorithm that, given a graph that contains a Hamiltonian circuit, is guaranteed to find one in polynomial time?"

The answer is, of course, the standard one: "Not unless  $P = NP$ ." If we had such an algorithm  $A$ , we could use it to tell in polynomial time whether an arbitrary graph  $G$  has a Hamiltonian circuit. Let  $p$  be the polynomial that bounds  $A$ 's running time on graphs with Hamiltonian circuits. Apply  $A$  to  $G$ . If  $G$  has a Hamiltonian circuit,  $A$  will find one in time  $p(|G|)$ . If  $G$  does not have such a circuit, then after  $p(|G|)$  steps  $A$  cannot have found one, and we will know that none exists.

Occasionally a persistent questioner makes a second attempt. "OK, suppose we restrict our goals further. Could there be an algorithm that, when given a

graph with *exactly one* Hamiltonian circuit (viewed as a set of edges), is guaranteed to find it in polynomial time?"

In its general form (does the fact that a problem instance has a unique solution make the problem easier to solve for that instance?), this second question turns out to be much more interesting than the first. Although some NP-complete problems are trivialized when restricted to instances with unique solutions, others appear to remain hard. Until quite recently, however, no one knew how such hardness might be proved. A new result of Valiant and Vazirani [31] has now provided us with the needed method, a method that depends heavily on the "randomized complexity" notions discussed in the [Sept. 1984] column. In this column I shall describe this new result, as well as other recent results about "uniqueness" and its impact on the theory and applications of NP-completeness (especially, as we shall see, the applications to cryptography).

Section 2 further motivates the problem posed above and sketches Valiant and Vazirani's answer to it. Section 3 then surveys results on a related question: How hard is it to determine *whether* a given problem instance has exactly one solution? In particular, the complexity classes for which such uniqueness questions might be "complete" are discussed, including a new class,  $D^p$ , recently introduced by Papadimitriou and Yannakakis [21]. Section 4 concludes by considering the "unambiguous Turing machine," a machine model devised by Valiant [28] to capture a slightly different notion of uniqueness.

## 2. A PROMISE OF UNIQUENESS

Readers of this column and of [G&J] should be familiar with the idea of investigating how the complexity of a problem is affected by restricting the problem's domain. By now such expressions as "remains NP-complete for planar graphs" and "solvable in polynomial time if the partial order is a tree" are the clichés of the field. For the most part, however, the study of restrictions has obeyed a fairly strong restriction itself. The restrictions considered have mostly been ones that could themselves be verified in polynomial time (like graph planarity), rather than ones (like solution uniqueness) for which no polynomial-time algorithms may be known. It was assumed that one would not want to design an algorithm for a special case of a problem unless one had an efficient way, given an instance, of telling whether the algorithm was applicable.

This need not be the case, however. For instance, we still know of no polynomial-time algorithm for identifying perfect graphs (see the [Jan. 1981] column), but this does not render useless the recently developed polynomial-time algorithms of [13] that solve CLIQUE and CHROMATIC NUMBER for perfect graphs. There are many important subclasses of the perfect graphs that *are* polynomial-time recognizable (chordal graphs, comparability graphs, permutation graphs, interval graphs, etc. [11]), and the algorithms are clearly applicable to these. More generally, if we know that all instances generated by a

particular application must have property “ $\Pi$ ” because of the nature of the application, then we can apply an algorithm that is known to work whenever property  $\Pi$  holds, whether or not property  $\Pi$  is in general hard to verify. An application for which uniqueness can be guaranteed in this way is cryptography. Most encryption schemes require that the recipient of an encoded message be able, with the aid of the appropriate key, to decode the original message precisely. Hence the encryption function should be one-one, and the problem “Given a string  $y$ , find an  $x$  such that  $y$  is the encoding of  $x$  (if one exists),” is guaranteed to have a unique solution if  $y$  is actually an encrypted message. Thus in a public key cryptosystem based on an NP-complete problem (like the knapsack systems of [18]), the would-be decoder is not faced with the general NP-complete problem, but rather with the restriction of that problem to instances with unique solutions.

In order to study restrictions of NP-hard problems to domains that are not known to be in P, a new formalism is needed. Even and Yacobi [9] have proposed a particularly natural one, the “promise problem.” In [G&J], a decision problem  $X$  is viewed as consisting of two parts, a polynomial-time recognizable instance domain  $I_X$  and a question  $Q_X$  that can be viewed as a function from  $I_X$  to the set {yes, no}. In a *promise problem*, these are joined by a *promise*  $P_X$ , which can be viewed as a predicate defined on  $I_X$ . The function  $Q_X$  now need only be defined for those members of  $I_X$  that satisfy the promise  $P_X$ . An algorithm  $A$  solves a promise problem  $X$  if it halts with the correct answer whenever its input is an instance in  $I_X$  satisfying  $P_X$ . (It need not give the correct answer, or even halt, for instances that do not satisfy the promise.)

It is not difficult to show that our uniqueness problem from the introduction is polynomial-time equivalent (via Turing reductions) to the following promise problem.

#### **UNIQUELY PROMISED HAMILTONIAN CIRCUIT (UPHC)**

INSTANCE: Undirected graph  $G = (V, E)$ .

QUESTION Does  $G$  contain a Hamiltonian circuit?

PROMISE:  $G$  contains at most one Hamiltonian circuit.

We say that a promise problem is solvable in polynomial time if there is an algorithm that solves it and, for instances satisfying the promise, runs in time bounded by a polynomial. Let us denote the class of promise problems that can be solved in this way by “PROMISE-P.” Since no one knows of an algorithm that solves UPHC in polynomial time, it is tempting to conjecture that UPHC is not in PROMISE-P. Can we use complexity theory to provide more substantial evidence?

We already have one ready-made tool: the “parsimonious transformation” introduced by Simon in 1975 [25] and described in Chapter 7 of [G&J]. This

special kind of transformation applies to decision problems  $X$  whose question  $Q_X$  is of the form “Given instance  $x$  such that  $R_X(x,y)$  is true?” (Recall that, according to one of the definitions of NP,  $X$  is in NP if and only if it can be formulated in these terms with a polynomial-time verifiable relation  $R_X$  that obeys the additional constraint that whenever  $R_X(x,y)$  is true,  $|y|$  is bounded by a fixed polynomial in  $|x|$ .) For a problem  $X$  in this form and an instance  $x \in I_X$ , let  $\#X(x) = |\{y: R_X(x,y)\}|$ . A *parsimonious transformation* from a problem  $X_1$  to a problem  $X_2$  is a polynomial transformation  $f$  such that for all instances  $x \in I_X$ ,  $\#X_1(x) = \#X_2(f(x))$ .

In the past, parsimonious transformations have been used mainly for proving results about the complexity of computing  $\#X(x)$ . (See [29,30] or Chapter 7 of [G&J] for a discussion of such “#P-completeness” results.) Note, however, the obvious applicability of parsimonious transformations to promise problems like UPHC. If  $X$  is a decision problem, let  $UPX$  be the promise problem obtained by adjoining the promise  $\#X(x) \leq 1$ . If  $X_1$  is parsimoniously transformable to  $X_2$  and the promise problem  $UPX_2$  is in PROMISE-P, then so is  $UPX_1$ . Thus to show that UPHC is hard, all we need do is find some problem  $X$  that (a) is parsimoniously transformable to HAMILTONIAN CIRCUIT and (b) is such that  $UPX$  is not in PROMISE-P (or at least is not in PROMISE-P unless some widely believed conjecture is false).

Due to the work of Simon [25,26] and Valiant [29,30], there are many candidates that satisfy (a). Until the new result of Valiant and Vazirani, however, none of these candidates was known to satisfy (b). The candidate they chose was (what else?) SATISFIABILITY, or “SAT” for short. Here, given a logical expression  $E$  in conjunctive normal form,  $\#SAT(E)$  is the number of satisfying truth assignments for  $E$ .

The task of proving that the promise problem UPHC is hard is thus reduced to that of proving that the promise problem UPSAT is hard. However, we still have not confronted the basic question of how one proves that promise problems are hard. To do this we must turn questions about PROMISE-P into ones about P. This is done as follows.

Let us call an ordinary decision problem  $Y$  a *completion* of a promise problem  $X$  if  $I_Y = I_X$  and, for all  $x \in I_X$  that satisfy  $P_X$ ,  $Q_Y(x) = Q_X(x)$ . (Even et al. [8] call such a  $Y$  a *solution*, which is perhaps giving it more credit than it deserves.) In terms of this definition, the ordinary HAMILTONIAN CIRCUIT problem is one completion of UPHC, but there are infinitely many others. The key observation is that a promise problem  $X$  is precisely as hard as its easiest completion. An algorithm for a completion of  $X$  clearly solves  $X$ , and an algorithm  $A$  that solves  $X$  can be turned into an algorithm with approximately the same running time that solves a completion of  $X$ : Simply modify  $A$  by having it halt and output “no” whenever its time bound for instances obeying the promise is exceeded. The answers output by the modified algorithm define a completion of  $X$ , and  $A$  by definition solves that completion. (This argument of course

assumes that  $A$ 's time bound is itself easy to compute.)

Thus to prove that UPSAT is hard, we must prove that, assuming some widely-believed conjecture, none of its completions is in P. The conjecture chosen by Valiant and Vazirani is that  $NP \neq RP$ , the class of those problems solvable in polynomial time by *randomized* algorithms (sometimes also called "R"). This conjecture is perhaps not as widely believed as  $NP \neq P$  (since it is known that  $P \subseteq RP \subseteq NP$ ), and hence its contradiction will provide slightly weaker evidence of intractability. On the other hand, the evidence, being weaker, is easier to obtain. One is not restricted to polynomial transformations, but can use more powerful "randomized" reduction techniques.

Readers may recall two such techniques, the "R-reduction" of Adleman and Manders [1] (discussed in the [Sept. 1984] column) and the "PR-reduction" of Vazirani and Vazirani [32] (mentioned in my [Dec. 1984] follow-up). Here we need yet another technique, one that I shall call an "RR-reduction" to distinguish it from its predecessors. It is easy to show (or would be easy, if I had room to provide the precise definition) that if problem  $X$  RR-reduces to problem  $Y$  and  $Y \in RP$ , then  $X \in RP$ . Recall that no NP-complete problem can be in RP unless  $NP = RP$ . Thus if we can RR-reduce a known NP-complete problem to  $Y$ , then  $Y$  is not in RP (or P) unless  $NP = RP$ .

Given a completion  $Y$  of UPSAT, the natural NP-complete problem to choose for reduction to  $Y$  is simply SATISFIABILITY itself. Omitting a few more key definitions, let me quickly indicate the heart of the construction, which is quite ingenious. Suppose we are given an instance  $E$  of SAT with variables  $v_1, \dots, v_m$ . Our goal is to construct a new instance  $E'$  that has a reasonable probability of having a unique solution when  $E$  is satisfiable, and has no solutions when  $E$  has none. To do this we generate a sequence of instances  $E_1, \dots, E_m$  such that  $E_i$  is satisfiable only if  $E$  was satisfiable *and* some additional conditions are satisfied. These conditions become more stringent as  $i$  increases, so that the number of satisfying truth assignments (if there are any) will tend to decline and there will be a chance that some  $E_i$  will have precisely one such assignment. For our output instance  $E'$ , we randomly pick one of the  $E_i$ .

The precise conditions on the  $E_i$  are obtained by generating a sequence  $u_1, \dots, u_m$  of random vectors in  $\{0, 1\}^m$ . Note that a truth assignment  $t$  for the variables  $v_j$  can also be viewed as a vector in  $\{0, 1\}^m$ , and so we can compute the inner product of  $t$  with any of the  $u_k$ 's. We construct  $E_i$  by adding to  $E$  a collection of clauses (in the  $v_j$  and some new variables) that force the inner products  $t \cdot u_j$  to be even when  $1 \leq j \leq i$  and  $t$  is the restriction of a satisfying truth assignment for  $E_i$  to the original variables  $v_i$ . If the clauses are designed correctly, there will be only a polynomial blow-up in the length of  $E_i$  and there will be a one-one correspondence between the satisfying truth assignments for  $E_i$  and those for  $E$  that obey the inner product constraints.

A snappy combinatorial lemma then shows that for any satisfiable  $E$ , the probability is at least 1/4 that one of the  $E_i$  will have exactly one satisfying truth

assignment. By randomly choosing one of the  $E_i$  as our output instance  $E'$ , we thus have a lower bound of  $1/4m$  on the probability that  $E'$  has a unique solution, and this is enough to make everything work. Thus no completion of UP-SAT can be in RP (and UPSAT cannot itself be in PROMISE-P) unless  $NP = RP$ , as claimed.

As pointed out earlier, this result immediately implies that UPHC is not in PROMISE-P unless  $NP = RP$ , nor is any other “UP” version of a decision problem reachable via parsimonious transformations from SAT. Among such decision problems is SUBSET SUM, the NP-complete problem upon which “knapsack” public key cryptosystems are based. Thus even when unique solutions are guaranteed, it is still hard to solve this problem. (This, unfortunately, is not enough to guarantee the security of knapsack cryptosystems, and in fact most such schemes have now been shown to be insecure for other reasons; see the [March 1983] and [June 1984] columns.) Additional problems that are reachable under parsimonious transformations and hence remain hard even when restricted to instances with unique solutions include CLIQUE, PARTITION, MAX CUT, 3SAT, 0-1 INTEGER PROGRAMMING and a host of others [26]. However, SAT cannot be parsimoniously transformed to *every* known NP-complete problem (despite an occasional claim to the contrary).

As an example of a problem to which SAT is *not* parsimoniously transformable, consider the special case “DOUBLE-SAT” of SAT, in which every instance contains a clause of the form  $\{x \vee \bar{x}\}$  where  $x$  is a variable that does not occur in any other clause. DOUBLE-SAT is still NP-complete, but for any instance  $E$ , the number of satisfying truth assignments, i.e.,  $\#DOUBLE-SAT(E)$ , must be even. (If  $x = T$  is part of a satisfying truth assignment  $S$ , then the assignment  $S'$  obtained by replacing  $x = T$  with  $x = F$ , must also be satisfying.) Since there are clearly instances of SAT with an odd number of accepting computations, a parsimonious transformation from SAT to this problem is impossible. Moreover, the UP version of this problem is in PROMISE-P: Since the number of satisfying truth assignments must be even, the promise that  $\#DOUBLE-SAT(f) \leq 1$  implies that there are *no* satisfying truth assignments, and so the answer is trivially “no.”

A less contrived example is provided by CHROMATIC INDEX: “Given a graph  $G = (V, E)$  and an integer  $k$ , can  $G$  be  $k$ -edge colored, i.e., is there a partition of  $E$  into  $k$  sets such that no set contains two edges sharing an endpoint?” This problem has been proved NP-complete for all fixed  $k \geq 3$  [15,17]. The UP version of this problem is trivial for all  $k \geq 4$ , however. A result of Thomason [27] (pointed out to me by Edwards and Welsh in [7]) shows that the only uniquely  $k$ -edge colorable graph for  $k \geq 4$  is the star  $K_{1,k}$ . Thus the UP version of CHROMATIC INDEX is in PROMISE-P for all fixed  $k \geq 4$  (and there can be a parsimonious transformation from SAT to CHROMATIC INDEX only if  $RP = NP$ ).

(Note that CHROMATIC INDEX might still be #P-complete for fixed  $k \geq 4$ ,

i.e., the problem of computing the number of  $k$ -colorings might still in general be hard. Strictly parsimonious transformations are not necessary in order to prove #P-hardness results. A transformation that, say, doubles the number of solutions, as did the transformation from SAT to DOUBLE-SAT above, will certainly still suffice. All one really needs for such results is (1) a polynomial transformation  $f$  such that  $\log(\#Y(f(x)))$  is bounded by a polynomial in  $|x|$  and (2) a polynomial-time algorithm  $A$  for computing  $\#X(x)$  given  $\#Y(f(x))$ .

Both the above examples have the property that it is easy to tell whether an instance has a unique solution. Could there be an NP-complete problem for which it is *hard* to tell whether an instance has a unique solution, and yet easy to generate a solution when it is unique? Currently no such examples are known, but there is a lot known about the complexity of the uniqueness question, “Given an instance  $x$  of problem  $X$ , does  $\#X(x) = 1$ ?” as we shall see in the next section.

### 3. THE QUESTION OF UNIQUENESS

For many a standard NP-complete problem  $X$ , the problem “UNIQUE- $X$ ” of whether a given instance  $x$  has a unique solution, i.e., whether  $\#X(x) = 1$ , is easily seen to be NP-hard. One simply shows how to modify an arbitrary instance  $x$  of  $X$  so that its number of solutions is increased by 1. Then there will be a unique solution for the modified  $x$  if and only if the answer for the original  $x$  is “no.” None of the randomization needed in the previous section is required.

For example, suppose  $E$  is an instance of SATISFIABILITY, which we may assume without loss of generality is not satisfied by the truth assignment that sets all variables true. To modify  $E$ , we add a new variable  $x$  to every clause of  $E$  and add the clause  $\{\bar{x} \vee y\}$  for every variable  $y$  that does occur in  $E$ . The modified instance  $E'$  is now satisfied by the truth assignment with all variables true, and there is a one-to-one correspondence between the remaining satisfying truth assignments and the assignments that satisfied  $E$ . Thus  $E$  is unsatisfiable if and only if  $\#SAT(E') = 1$ . See [20] for a similar construction involving HAMILTONIAN CIRCUIT. Such results can of course be propagated using (strictly) parsimonious transformations.

The NP-completeness of a problem  $X$  does not, however, *guarantee* the NP-hardness of UNIQUE- $X$  (assuming  $P \neq NP$ ). For instance, DOUBLE-SAT and CHROMATIC INDEX from the last section are exceptions. One might similarly ask if the polynomial-time solvability of a problem  $X$  guarantees the polynomial-time solvability of UNIQUE- $X$ . The answer is yes for many interesting problems, even for ones like PERFECT MATCHING where, despite the ease of telling whether  $\#X(x) > 0$ , the problem of computing  $\#X(x)$  is #P-complete [29]. (I leave as an exercise the task of showing how, in the case of PERFECT MATCHING, a polynomial-time algorithm for telling whether  $\#X(x) > 0$  can be used in the construction of one that tells whether  $\#X(x) = 1$ .) On the other hand, consider NONZERO-SAT, the restriction of

SATISFIABILITY to only those instances that can be generated by the construction in the previous paragraph. Since all instances of this restricted problem have answer “yes,” it is clearly in P, but it is also clear that UNIQUE-NONZERO-SAT is just as hard as UNIQUE-SAT and hence NP-hard.

It thus appears that we can only determine whether UNIQUE- $X$  is hard on a case-by-case basis. In the remainder of this Section I shall address the more interesting theoretical question, “Given that UNIQUE- $X$  is hard, precisely how hard is it?” This is “interesting” because there may be senses in which UNIQUE- $X$  is “harder” than  $X$ , even when  $X$  is NP-complete. It is “theoretical” because, even so, UNIQUE- $X$  will still be solvable in polynomial time if and only if  $X$  is. Readers interested in learning more about UNIQUE- $X$  for individual problems may wish to explore the extensive mathematical literature on such questions (e.g., see [14], Chapter 12). Mathematicians have been attempting to characterize problems with unique solutions since long before P and NP were invented, and many of their results, like Thomason’s for CHROMATIC INDEX, may have algorithmic significance.

In what follows, let us use UNIQUE-SAT as a standard example. It is often said that a problem  $X$  is “harder” than a problem  $Y$  if  $X$  lies higher in the Polynomial Hierarchy than  $Y$ . (For a discussion of the Polynomial Hierarchy, see [G&J], Chapter 7.) The possibility that UNIQUE-SAT might be harder than SAT in this sense arises because UNIQUE-SAT is not obviously in either NP or co-NP. The smallest class in the Hierarchy that clearly contains UNIQUE-SAT is  $\Delta_2^P$ , the class of all decision problems that can be solved in polynomial time if given an oracle for an NP-complete problem. Such an oracle can answer any question in NP, so since “Is #SAT( $x$ ) > 0?” and “Is #SAT( $x$ ) > 1?” are both in NP, we can determine whether #SAT( $x$ ) = 1 by just two questions to the oracle.

Now, on any complexity theorist’s list of widely-believed conjectures, right up there with  $P \neq NP$  and  $NP \neq RP$ , is the conjecture that  $\Delta_2^P$  *properly* contains  $NP \cup \text{co-NP}$  (containment itself is immediate from the definitions). Could it be that UNIQUE-SAT is one of the problems in  $\Delta_2^P - NP \cup \text{co-NP}$ ? One way to show this would be to prove that UNIQUE-SAT is complete for  $\Delta_2^P$ , and indeed this has been conjectured. It may, however, be too much to hope for. Somehow a problem that only needs to call on the oracle twice does not seem to be using the full power of a machine that is allowed a polynomial number of calls. Given this reservation, Papadimitriou and Yannakakis [21] have recently invented a class that seems much more likely to capture the complexity of problems of this type.

The class is called  $D^P$  (“ $D$ ” for “difference”). In language-theoretic terms,  $D^P$  consists of all languages that can be expressed as a difference  $L_1 - L_2$ , where  $L_1$  and  $L_2$  are languages in NP. In other words,  $D^P$  consists precisely of those problems in  $\Delta_2^P$  that require just two calls to the oracle, with the problem’s answer being “yes” if and only if the two oracular answers are “yes” and

“no,” in that order. Note that  $D^P$  contains both NP and co-NP, as can be seen by alternately setting  $L_2 = \emptyset$  and  $L_1 = \Sigma^*$ . In addition to questions of uniqueness,  $D^P$  also contains a variety of other types of questions that do not in general seem to be in either NP or co-NP.

One type is the *exact problem*, as exemplified by EXACT CLIQUE: Given a graph  $G$  and an integer  $k$ , does the largest clique in  $G$  contain exactly  $k$  vertices? This is in  $D^P$  since the property holds if and only if there is a clique of size  $k$  or more, but no clique of size  $k + 1$  or more. Another type is the *critical problem*, as exemplified by MAXIMAL NONHAMILTONIAN GRAPH: Given a graph  $G = (V, E)$ , is it the case that every graph obtained by adding an edge between two non-adjacent vertices of  $G$  has a Hamiltonian circuit, whereas  $G$  itself does not? (Here the yes-no nature of the problem is explicit, so membership in  $D^P$  is obvious.) A third type is the *facet problem*, as exemplified by TSP FACETS: Given a linear inequality over the variables  $x_{ij}$ ,  $1 \leq i < j \leq n$ , does it determine a facet of the  $n$ -city traveling salesman problem polytope? (Readers interested in deciphering this last definition are referred to [21], which also explains why the problem is in  $D^P$ .)

Given the variety of problems that are contained in  $D^P$  but not known to be in NP or co-NP, one can expect that the conjecture  $D^P \neq \text{NP} \cup \text{co-NP}$  will be coming soon to your local neighborhood list of widely-believed conjectures. (In fact, it's already there. Surprisingly, it turns out to be equivalent to the above conjecture that  $\Delta_2^P$  properly contains  $\text{NP} \cup \text{co-NP}$ , and both are equivalent to the much more famous  $\text{NP} \neq \text{co-NP}$ .) Thus a proof that UNIQUE-SAT is complete for  $D^P$  would provide strong evidence that it was in neither NP nor co-NP. Unfortunately, no such proof has been found, although a variety of other problems have been shown  $D^P$ -complete, via transformations from the “first”  $D^P$ -complete problem, SAT-UNSAT: “Given two logical expressions  $E_1$  and  $E_2$  in conjunctive normal form, is it the case that  $E_1$  is satisfiable and  $E_2$  is not?” These  $D^P$ -complete problems include EXACT CLIQUE [21] along with some variants of TSP FACETS (TSP SUPPORTING HYPERPLANES and a not-strictly-analogous CLIQUE FACETS problem [21]), but do not include TSP FACETS itself, or any critical or uniqueness problems.

Furthermore, a recent result of Blass and Gurevich [2] indicates that we may not be able to prove that UNIQUE-SAT is  $D^P$ -complete, even if it is. They show that there are oracle sets  $A$  and  $B$  such that UNIQUE-SAT is  $D^P$ -complete relative to  $A$  and is *not*  $D^P$ -complete relative to  $B$ , although for both  $A$  and  $B$ ,  $D^P$  properly contains  $\text{NP} \cup \text{co-NP}$ . Thus our current proof techniques, like simulation and diagonalization, whose results tend to be preserved under relativization, are unlikely to be potent enough to resolve the question of whether UNIQUE-SAT is  $D^P$ -complete.

A slightly different completeness result *is* possible, however, as a corollary of the result of Valiant and Vazirani [] mentioned in the previous section. UNIQUE-SAT can be proved complete for  $D^P$  if *randomized* reductions are

allowed (instead of polynomial transformations). This weaker notion of completeness may of course have weaker consequences. Although it implies that UNIQUE-SAT cannot be in NP unless  $\text{NP} = \text{co-NP}$ , it does not appear to say anything so direct about the possibility that UNIQUE-SAT is in co-NP.

Given this background, a recently published result of Papadimitriou [19] may come as a surprise. He shows that a slightly different kind of uniqueness problem is not only  $D^P$ -hard, but is complete for all of  $\Delta_2^P$ ! The problem is UNIQUELY OPTIMAL TRAVELLING SALESMAN TOUR: “Given an instance of the TSP, is there exactly one optimal tour?” The role of uniqueness in this result turns out to be less than it appears, however, as was pointed out to me by Mark Krentel [16].

The problem that is really  $\Delta_2^P$ -complete (or would be if  $\Delta_2^P$  were not restricted to decision problems) is that of *computing* the optimal tour length. Telling whether there is but one optimal tour can be done with a  $D^P$  algorithm (and hence just two calls) once the optimal tour length is known. Asking whether there is a unique optimal tour is thus just a way of forcing us to compute the optimal tour length. Another way is to ask, given an integer  $k$ , if the optimal tour length is divisible by  $k$ , a problem that is also  $\Delta_2^P$ -complete. (This can be proved by replacing the “non-uniqueness” gadget by a “non-divisible-by- $k$ ” gadget in Papadimitriou’s proof.)

The significance of uniqueness here is thus not that it makes the problem hard but rather that it doesn’t make the problem easier. This is more than can be said for the standard decision problem version of the TSP (“Given a TSP instance and an integer  $k$ , is the optimal tour of length  $k$  or less?”), which is “only” NP-complete. It is also more than can be said for the exact decision version (“Given a TSP instance and an integer  $k$ , is the optimal tour length exactly  $k$ ?”), which is only  $D^P$ -complete [21]. Providing a proposed tour length as a target to shoot for (or at) seems to be giving too much away.

Note that the  $\Delta_2^P$ -hardness of computing the optimal TSP tour length may not extend to other optimization problems, such as computing the maximum clique size. The optimal tour length can be exponential in the instance size, given that edge lengths are written in binary notation, and hence the allowed polynomial number of oracle calls may all be required if this length is to be computed using binary search. Since the maximum clique size is polynomially bounded in the instance size, only a logarithmic number of calls will be needed to compute it. Observe, however, that this is still more than the two calls needed for problems in  $D^P$ . If one were interested in making the kind of “theoretical” distinctions we have been discussing in this section, one might well ask how many “NP-complete” optimization problems, besides the TSP, are actually harder than NP-complete when viewed in their full generality (or when stated in a decision problem version that gives less away).

#### 4. COMPUTING WITH UNIQUENESS

The questions we discussed in the previous sections had to do with ordinary NDTM's. Uniqueness came into the picture because we were either asking how hard it was to tell if a particular instance had a unique accepting computation, or else restricting attention to those instances that did. In this section we look at a machine model with the requirement for uniqueness built in: an NDTM for which *no* input has more than one accepting computation, called by Valiant [28] an *unambiguous* Turing machine, or "UTM."

We already are familiar with one particular kind of UTM. The ordinary *deterministic* Turing machine has at most one computation of any form, accepting or rejecting. Thus, if we let "UP" denote the class of languages recognizable by polynomial-time UTM's, we have  $P \subseteq UP \subseteq NP$ . To illustrate the possibility that polynomial-time UTM's might be more powerful than polynomial-time DTM's (and hence that  $UP - P$  might be nonempty), consider the problem of computing "discrete logarithms." Suppose we are given a prime  $p$  and a primitive root  $a$  modulo  $p$ . For any  $b$ ,  $0 < b < p$ , the *discrete logarithm* of  $b$  with respect to  $p$  and  $a$  is that integer  $c$ ,  $0 \leq c < p$ , such that  $a^c = b \pmod{p}$ . Currently there is no known polynomial-time algorithm for computing discrete logs (assuming the inputs are written in binary). Indeed, public key cryptosystems have been proposed based on the assumption that this problem is hard [6]. Nevertheless, suppose we view a UTM as computing a partial function (defined for the inputs it accepts, and, when defined, equal to the contents of its work tape at the end of the single accepting computation). Then there is a UTM that computes discrete logs, assuming that the input includes not only  $p$  and  $a$ , but also a short proof that  $p$  is a prime and  $a$  is a primitive root modulo  $p$ . (Pratt, in [22], shows that short proofs exist for all such  $p$  and  $a$ .) The UTM starts by verifying that  $p$  is a prime and  $a$  is a primitive root, using the provided proof, and halts unless the proof is correct. Assuming correctness, there is guaranteed to be at most one (in fact, exactly one) satisfactory value for  $c$ ,  $1 \leq c < p$ , so the UTM can proceed simply by guessing a potential  $c$  and checking to see if  $a^c = b \pmod{p}$ , using the standard "repeated squaring" technique. There is a computation for each possible guess, and since at most one guess can be correct, there will be at most one accepting computation. This suggests that the following "DISCRETE LOG" decision problem might be in  $UP - P$ : "Given  $p$ ,  $a$ ,  $b$ , the appropriate proof, and a positive integer  $k$ , is it true that the proof works and the discrete log of  $b$  with respect to  $p$  and  $a$  is less than  $k$ ?"

The relationship between UTM's and cryptography is not limited to this one example. Consider, for example, that holy grail of public key cryptography, the "one-way function." Although there are other proposed definitions (e.g., see [3]), for our purposes let us follow [12] and say that a (partial) function  $f$  is *one-way* if (a)  $f$  is one-one, (b)  $f$  is "honest" (for some polynomial  $p$ ,  $|f(x)|$  is guaranteed not to be so small that  $|x| > p(|f(x)|)$ ), (c)  $f$  can be computed in

polynomial time, and (d)  $f^{-1}$  is *not* computable in polynomial time. In light of the above remarks about discrete logs, the function  $f(p, a, \text{proof}, c) = (p, a, \text{proof}, a^c \pmod{p})$  is a candidate for a one-way function ( $p$ ,  $a$ , and the proof are included in the output to ensure that  $f$  is one-one). If it *is* one-way, then  $\text{UP} \neq \text{P}$  (the above DISCRETE LOG problem will be in  $\text{UP} - \text{P}$ ). The converse isn't necessarily true (the discrete log could be easy even though  $\text{UP} \neq \text{P}$ ), but one can show that if there is a language  $L \in \text{UP} - \text{P}$ , there must exist *some* one-way function  $f$  (simply define  $f$  in terms of the computations of a UTM accepting  $L$ ). Furthermore, suppose we adopt the convention that a DTM computing a partial function must always return the special symbol  $\Omega$  when the function is undefined (and hence that the domain of a one-way function must be in  $\text{P}$ ). Then it is easy to see that the inverse of any one-way function can be computed by a polynomial-time UTM, and so there exists a one-way function if and only if  $\text{UP} \neq \text{P}$  [12].

Having exhibited a potential member of  $\text{UP} - \text{P}$  that has cryptographic significance, it seems only fair now to exhibit a potential member of  $\text{NP} - \text{UP}$  with similar significance (in this case a relation to the RSA public key cryptosystem of [24]). The problem is FACTORIZATION: "Given positive integers  $k < n$ , does  $n$  have a factor less than  $k$ ?" This problem is clearly in  $\text{NP}$ , and one might at first think that it would be in  $\text{UP}$ . This is because every integer has a unique decomposition as a product of primes, suggesting that we just guess the prime decomposition and accept if and only if one of the factors is less than  $k$ . Unfortunately, this is not enough. To be sure that we have the desired unique decomposition of  $n$  into factors  $p_1, p_2, \dots, p_k$ , it is not enough simply to multiply the  $p_i$  together and verify that their product is  $n$ , as one does when testing for compositeness. We must also verify that each  $p_i$  is prime. This can be done by guessing short proofs of primality, as described in [22], but [22] allows for the possibility of many equivalent proofs, depending on one's choices of primitive roots. There seems no obvious way of ensuring that exactly one is guessed, so there may well be more than one accepting computation, even though there is but one prime decomposition. If one could show that primality testing were in  $\text{P}$  (or even in  $\text{UP}$ ), we could conclude that FACTORIZATION was in  $\text{UP}$ , but until then it remains a valid candidate for  $\text{NP} - \text{UP}$ .

At this point, a reader might be justified in questioning whether  $\text{UP}$  is really the correct class for capturing the complexity issues surrounding public key cryptography. To get a version of the discrete log problem into  $\text{UP}$  we had to include a primality proof in the input. This extra information, not normally provided in practice, could conceivably make the computation of discrete logs easier (although there is as yet no evidence in support of this possibility). FACTORIZATION may not be in  $\text{UP}$  even though prime factorizations are unique. Although both these difficulties will go away if primality testing turns out itself to be in  $\text{UP}$ , these examples raise a fundamental question about the relevance of both  $\text{UP}$  and our definition of "one-way function." How much should we care

about what happens on inputs that are not in the domain of a proposed one-way function? The difficulties encountered above come from requiring that the domain be in  $P$  or  $UP$ , whereas it would seem that in many applications membership in  $NP$  would suffice.

Nevertheless, connections between  $UP$  and public key cryptography *have* been made. In [8], Even et al. propose a definition for public key cryptosystem that does not directly depend on the existence of one-way functions in our sense, but in [12] Grollman and Selman show that, assuming “reasonable” restrictions, such a system can have a polynomial-time solvable “cracking problem” only if  $UP = P$ . In [8], Even et al. provide a plausible conjecture about promise problems that would imply both that  $UP \neq NP$  and that no public key cryptosystem has an *NP-hard* cracking problem (although this would not rule out a cracking problem that was neither  $NP$ -hard nor solvable in polynomial time). A variety of other relations between cracking problems, polynomial-time UTM’s, and  $UP$  are covered in [8] and [12]. (The latter even contains some results relating  $UP$  to the “average” difficulty of cracking problems, which is of course the more crucial measure for cryptographic purposes.)

It thus appears that we might know a lot more about what types of cryptographic systems are possible if only we knew more about  $UP$ . As usual, however, there is a collection of oracle results waiting in the wings to frustrate any hopes of early progress. Relative to one oracle, we have  $P = UP \neq NP$  [23]; to a second we have  $P \neq UP = RP = NP = \text{co-NP}$  [23]; to a third we have all four sets  $P$ ,  $UP$ ,  $RP$ ,  $NP$  distinct [10]. Thus present proof techniques, whose results tend to survive relativization, cannot be expected to resolve much.

It is possible to look on the bright side, however. Since the results mentioned in this section are proved using present proof techniques, they themselves survive relativization. This means that one-way functions are guaranteed to exist in certain relativized worlds, thus opening the way for a brand new field of research: relativized cryptography! (See [4,5].)

## 5. ACKNOWLEDGMENT

It has been some time since I’ve had the opportunity (actually, the space) to thank in print the large crew of friends who regularly provide me with stylistic and technical comments on drafts of this column: Jon Bentley, Steve Fortune, Mike Garey, Jeff Lagarias, Steve Mahaney, Andrew Odlyzko, Herb Wilf, Chris Van Wyk, and Mihalis Yannakakis. In addition to these, I would particularly like to thank Mark Krentel, Christos Papadimitriou, Alan Selman, and Vijay Vazirani for illuminating discussions about the topics covered in this edition.

## REFERENCES

1. L. M. ADLEMAN AND K. MANDERS, Reducibility, randomness and intractability, in "Proceedings 9th Ann. ACM Symp. on Theory of Computing," pp. 151-163, Association for Computing Machinery, New York, 1977.
2. A. BLASS AND Y. GUREVICH, On the unique satisfiability problem, *Information and Control* **55** (1982), 80-88.
3. G. BRASSARD, A note on the complexity of cryptography, *IEEE Trans. Inform. Theory* **IT-25** (1979), 232-233.
4. G. BRASSARD, Relativized cryptography, in "Proceedings 20th Ann. Symp. on Foundations of Computer Science," pp. 383-391, IEEE Computer Society, Los Angeles, 1979.
5. G. BRASSARD, A time-luck tradeoff in relativized cryptography, *J. Comput. System Sci.* **22** (1981), 280-311.
6. W. DIFFIE AND M. E. HELLMAN, New directions in cryptography, *IEEE Trans. Inform. Theory* **IT-22** (1976), 644-654.
7. K. J. EDWARDS AND D. J. A. WELSH, On the complexity of uniqueness problems, manuscript (1983).
8. S. EVEN, A. L. SELMAN, AND Y. YACOBI, The complexity of promise problems with applications to public-key cryptography, *Information and Control* **61** (1984), 159-173.
9. S. EVEN AND Y. YACOBI, Cryptography and NP-completeness, in "Automata, Languages, and Programming," pp. 195-207, Lecture Notes in Computer Science, Vol. 85, Springer, Berlin, 1980.
10. J. GESKE AND J. GROLLMAN, Relativizations of unambiguous and random polynomial time classes, *SIAM J. Comput.*, to appear. Unseen paper, cited in [12].
11. M. C. GOLUMBIC, "Algorithmic Graph Theory and Perfect Graphs," Academic Press, New York, 1980.
12. J. GROLLMAN AND A. L. SELMAN, Complexity measures for public-key cryptosystems, in "Proceedings 25th Ann. Symp. on Foundations of Computer Science," pp. 495-503, IEEE Computer Society, Los Angeles, 1984.
13. M. GRÖTSCHEL, L. LOVÁSZ, AND A. SCHRIJVER, The ellipsoid method and its consequences in combinatorial optimization, *Combinatorica* **1** (1981), 169-198.
14. F. HARARY, "Graph Theory," Addison-Wesley, Reading, Mass., 1969.
15. I. HOLYER, The NP-completeness of edge coloring, *SIAM J. Comput.* **10** (1981), 718-720.
16. M. KRENTEL, private communication (1985).
17. D. LEVEN AND Z. GALIL, NP-completeness of finding the chromatic index of regular graphs, *J. Algorithms* **4** (1983), 35-44.
18. R. C. MERKLE AND M. E. HELLMAN, Hiding information and signatures in trapdoor knapsacks, *IEEE Trans. Inform. Theory* **IT-24** (1978), 525-530.
19. C. H. PAPADIMITRIOU, On the complexity of unique solutions, *J. Assoc. Comput. Mach.* **31** (1984), 392-400.
20. C. H. PAPADIMITRIOU AND K. STEIGLITZ, Some complexity results for the traveling salesman problem, in "Proceedings 8th Ann. ACM Symp. on Theory of Computing," pp. 1-9, Association for Computing Machinery, New York, 1976.
21. C. H. PAPADIMITRIOU AND M. YANNAKAKIS, The complexity of facets (and some facets of complexity), *J. Comput. System Sci.* **28** (1984), 244-259.
22. V. PRATT, Every prime has a succinct certificate, *SIAM J. Comput.* **4** (1975), 214-220.
23. C. RACKOFF, Relativized questions involving probabilistic computations, *J. Assoc. Comput. Mach.* **29** (1982), 261-268.
24. R. RIVEST, A. SHAMIR, AND L. ADLEMAN, A method for obtaining digital signatures and public-key cryptosystems, *Comm. ACM* **21** (1978), 120-126.

25. J. SIMON, "On Some Central Problems in Computational Complexity," Doctoral Thesis, Dept. of Computer Science, Cornell University, Ithaca, N.Y., 1975.
26. J. SIMON, On the difference between one and many, in "Automata, Languages, and Programming," pp. 480-491, Lecture Notes in Computer Science, Vol. 52, Springer, Berlin, 1977.
27. A. THOMASON, Hamiltonian cycles and uniquely edge colourable graphs, *Annals Disc. Math.* **3** (1978), 259-268.
28. L. G. VALIANT, Relative complexity of checking and evaluating, *Inform. Process. Lett.* **5** (1976), 20-23.
29. L. G. VALIANT, The complexity of computing the permanent, *Theor. Comput. Sci.* **8** (1979), 189-201.
30. L. G. VALIANT, The complexity of enumeration and reliability problems, *SIAM J. Comput.* **8** (1979), 410-421.
31. L. G. VALIANT AND V. V. VAZIRANI, NP is as easy as detecting unique solutions, in "Proceedings 17th Ann. ACM Symp. on Theory of Computing," Association for Computing Machinery, New York, 1985.
32. U. V. VAZIRANI AND V. V. VAZIRANI, A natural encoding scheme proved probabilistic polynomial complete, *Theor. Comput. Sci.* **24** (1983), 291-300.