

## The NP-Completeness Column: An Ongoing Guide

DAVID S. JOHNSON

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

This is the twelfth edition of a quarterly column that provides continuing coverage of 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.rabbit.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 If the results are unpublished, please state explicitly that you would like them to be mentioned in the column.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

This column continues the theme of the last, again concentrating on algorithms for hard problems that do well "on average." However, whereas last time our averages were computed assuming a probability distribution on problem instances, this time the probability is in the algorithms, not the instances. More specifically, we shall discuss "randomized algorithms," which make some of their decisions based on the flipping of a coin (or the output of a random number generator). Instead of deterministically defining a unique outcome for each instance, such an algorithm defines, for each instance, a probability distribution over the set of possible outcomes. It is over such distributions that we now take our averages.

Perhaps the most famous randomized algorithms are the primality testing algorithms of Solovay and Strassen [44] and of Rabin [30,31]. Although the technical details of the two approaches differ, they both rely on theorems of the following form: If  $n$  is not prime, then more than half the integers  $b$ ,

$1 \leq b \leq n-1$ , are “witnesses” to this non-primality, in that, given  $b$  and  $n$ , one can verify that  $n$  is composite in time polynomial in the length of  $n$  (although one still may not know its factors). The actual time bound for the verification is  $O(\log^3 n)$  in both cases, assuming that multiplications are performed using  $\Theta(\log^2 n)$  bit operations per multiplication.

Given such a result, one can test primality as follows: randomly choose an integer  $b$ ,  $1 \leq b \leq n-1$  and test whether  $b$  is a witness of non-primality. If so, truthfully declare that  $n$  is composite. Otherwise, tentatively label  $n$  as “prime.” The probability that our conclusion is in error is less than  $1/2$ : If  $n$  were prime we would have given the correct answer, and if it were composite the probability that we labeled it prime would be equal to the fraction of integers between 1 and  $n-1$  that were not witnesses, which is less than  $1/2$ .

Being only half certain that we have the right answer is not necessarily reassuring. One way to improve accuracy would be to show that there are actually more witnesses than originally claimed. (This has been done for Rabin’s approach [31] and a variant on Solovay-Strassen’s [6], reducing the error probability of each to  $1/4$  or less.) The beauty of this type of approach, however, is that we can make the error probability arbitrarily small merely by repeating the experiment. Suppose we repeat Rabin’s test  $k$  times, with  $k$  independent random choices of  $b$ , and declare  $n$  to be prime only if none of the  $b$ ’s turn out to be a witness. It is easy to see that the probability that a composite number will be labeled “prime” is now reduced to  $(1/4)^k$ . Taking  $k = 50$  will reduce our probability of error far below the probability that a hardware error will interfere with our computations, and yet will still yield a running time of  $O(\log^3 n)$ . And note that this error reduction technique does not require that the initial error probability be small, only that it be less than  $1 - \epsilon$  for some  $\epsilon > 0$ . If instead of  $1/4$  the error probability had been .99, we would merely need to increase the number of repetitions by a fixed multiplicative factor to obtain the same ultimate error probability of  $(1/2)^{100}$ , still in  $O(\log^3 n)$  time. In contrast, the best running time currently known for a deterministic primality-testing algorithm is non-polynomial, and even assuming the extended Riemann hypothesis is  $O(\log^5 n)$  [28].

Comparisons of probabilistic and deterministic results are straightforward here because, despite the probabilities involved, the results about randomized algorithms are, in a sense, worst-case results. For each primality testing instance we have an error probability (probabilistic analysis), but we prove a bound on this probability that holds for all instances (worst-case analysis). An alternative use of randomization would be to devise an algorithm that always finds the correct answer, but whose *running time* (instead of its correctness) obeys some probability distribution. Rabin [30] has done this for the problem of finding the nearest pair in a collection of points in a unit square. Again the relevant analysis is a hybrid of probabilistic and worst-case. The algorithm is shown to have an expected running time that is  $O(n)$  for *every* instance, as opposed to the

$\Omega(n \log n)$  worst case running time of the best deterministic algorithms of the day.

Results such as these suggest the possibility that, even should P not equal NP, we might still be able to solve NP-complete problems quickly using randomized algorithms. The remainder of this column will address this question, along with related theoretical topics having to do with randomized complexity. Section 2 surveys further successes of randomization, especially those that offer improvements by more than a polynomial factor over the best that determinism currently has to offer. Section 3 then examines the theoretical possibilities for randomization by considering the complexity classes defined by various notions of randomized algorithm and discussing how these classes relate to the ones we all know and love. Section 4 then concludes with the question, “Given randomized algorithms, why not randomized polynomial transformations?” (Note that all the results I shall describe are predicated on our ability to make truly random choices. This is an ability we can only approximate with the pseudo-random number generators to which we must resort in practice, although, in practice, the approximation seems to be good enough.)

## 2. A RANDOM HARVEST

How much can one really gain by using randomization?

Often not as much as one first thought. Consider the “nearest pair” problem mentioned in the previous section. Here an improvement by a factor of  $\log n$  was attributed to randomization. It turns out, however, that Rabin’s algorithm used a second trick in addition to randomization. Whereas previous algorithms had relied only on the operation of comparing distances, Rabin added to this the computation of the “floor” function. Subsequently, Fortune and Hopcroft [13] showed that the use of the floor function alone, without randomization, was sufficient to improve the time from  $O(n \log n)$  to  $O(n \log \log n)$ , making the improvement due to randomization small indeed.

On the other hand, there *is* a substantial gap left between the best randomized and the best deterministic algorithms known for primality testing, the other example from Section 1. If we do not assume the extended Riemann Hypothesis, this gap is, if not exponential, then at least super-polynomial. The fastest known algorithm not depending on that hypothesis has running time  $O((\log n)^{c \log \log \log n})$  [3] as compared to our  $O(\log^3 n)$  randomized algorithms.

Algebra and number theory provide still more examples of problems for which the only currently known polynomial-time algorithms are randomized ones. Consider the problem of determining whether two collections  $P = \{P_1, P_2, \dots, P_n\}$  and  $Q = \{Q_1, Q_2, \dots, Q_m\}$  of multivariate polynomials over the rationals yield the same product polynomial, i.e., whether  $\prod_{i=1}^n P_i \equiv \prod_{i=1}^m Q_i$ . Schwartz [38] has shown that a simple randomized test for this problem has error probability less than  $1/2$ , and can be iterated to reduce this probability to an

arbitrarily small  $\epsilon$ . The test involves randomly choosing values for the variables (within specified ranges), computing the values of the  $P_i$ 's and  $Q_i$ 's for these values, and then testing whether the product of the values for the  $P_i$ 's equals that for the  $Q_i$ 's. Suppose that, instead of the standard complexity-theoretic measure of instance size, we take as our measure the "sparse metric," in which the size of a polynomial is the sum of the sizes of its terms, and the size of a term is the number of bits needed to write its coefficient plus the sum of its variables' exponents (rather than the number of bits required to write them). For any fixed error probability, the randomized algorithm runs in polynomial time with respect to the sparse metric.

No deterministic polynomial-time algorithm is known for this problem (and this metric). Expanding the two products may yield an exponential blow-up in the number of terms. The only obvious alternative is to factor each of the  $P_i$ 's and  $Q_i$ 's into its irreducible factors, and then compare the two composite lists of irreducible factors for  $P$  and  $Q$  to see if they are identical. Unfortunately, this too may result in an exponential blow-up, since there exist polynomials whose irreducible factors have exponentially more terms than they themselves do [49].

Another obstacle to this latter approach is that no one knows how to factor multivariate polynomials deterministically in time polynomial in the sparse metric (even if we are willing to accept a bound that is polynomial in the size of the polynomial *plus* the sizes of its factors). The recent flurry of results about factoring polynomials in polynomial time mentioned in the [March 1983] column assumes the *dense* metric, where the size of a polynomial is the number of bits in its largest coefficient plus the product of the maximum degrees, *over all terms*, for each of the variables. This can be exponential with respect to the sparse metric, since the "dense" size of  $\sum_{i=1}^k x_i^2$  is  $1 + 2^k$  whereas its "sparse" size is  $k(1 + 2)$ .

If we want a polynomial-time factoring algorithm with respect to the sparse metric, our only currently available option again involves randomization. In [49], von zur Gathen describes a randomized algorithm whose error probability can be made arbitrarily small and whose expected running time is polynomial (in the combined sparse metric size of the polynomial and its factors). Note that here we must settle both for a possibility of error *and* a running time that is polynomially bounded in an expected rather than a worst-case sense. (If, however, one wants only to check that the polynomial is irreducible, i.e., has no non-trivial factors, the worst-case running times *are* polynomial [49].)

If the factorization is over a finite field, rather than the rationals, then the gap between the best of the known randomized and deterministic algorithms is even greater: the deterministic algorithms are polynomial only in the dense metric and the size of the field [24,50], whereas the randomized ones of [49] are (expected) polynomial in the sparse metric and the *log* of the size of the field. A second randomized algorithm, from [50], trades the expected polynomial time for worst-case polynomial time, at a cost of replacing the sparse metric by the

dense one, although the polynomial still involves the log of the field size rather than the size itself. Another benefit of this latter algorithm is that it also upgrades Monte-Carlo to Las Vegas.

There are perhaps some who do not normally think of Las Vegas as an enhanced version of Monte-Carlo, but then there is no accounting for tastes where technical terminology is concerned. “Monte-Carlo algorithm” has long been used as a generic term for a randomized algorithm, apparently having been introduced by S. Ulam in the context of simulation techniques developed at Los Alamos in the late 1940’s [47]. More recently, the term “Las Vegas algorithm” was introduced by L. Babai (while in the vicinity of the gaming tables of San Juan [8]) to characterize those randomized algorithms that do not lie: They either give the correct answer or (with low probability) announce that they have failed to find the answer. Note that the primality testing algorithms mentioned in Section 1 were not Las Vegas algorithms, as they could incorrectly identify a composite number as a prime, albeit with low probability.

Babai introduced the term in conjunction with his research into randomized algorithms for the graph isomorphism problem [7]. The graph isomorphism problem, despite its purely combinatorial definition, shares with the above problems a strongly algebraic flavor. Determining whether two graphs are isomorphic is equivalent to determining the order of the automorphism group of a graph, and Babai’s algorithm was heavily group-theoretic in nature. The problem it solved was a variant of graph isomorphism in which the vertices of the two graphs are colored and one is looking for a color-preserving isomorphism. The algorithm was of the Las Vegas variety, and for any fixed bound  $k$  on the maximum size of a color class, ran in polynomial time. This result preceded (and in fact was the technical inspiration for) the current profusion of polynomial-time algorithms for special cases of the standard graph isomorphism problem described in part in the [Dec. 1981] column. It is now, however, only a historical footnote and an unpublished manuscript, since shortly after the result was announced, Furst et al. [15] showed that the colored graph isomorphism problem could be solved *deterministically* in polynomial time. The only current contributions of randomization to graph isomorphism testing are speed-ups in the expected running times of deterministic polynomial-time algorithms. For instance, the best current deterministic algorithm for testing isomorphism of trivalent graphs, with worst-case running time of  $O(n^3 \log n)$ , can be modified, using error-free randomization, to run in expected time  $O(n^3)$  for every instance (while still obeying the same worst-case bound) [16].

Because some randomized algorithms have been superseded by deterministic ones, skeptics have been led to wonder whether the other apparent successes of randomization might prove similarly illusory. Note that, although I have described a variety of problems whose only known polynomial-time “solutions” are randomized ones, not one of these problems is known to be NP-hard. Thus even if  $P \neq NP$  there still could be polynomial-time deterministic algorithms for

each.

As a side remark, I should mention that there *are* NP-hardness results related to factoring polynomials. These assume that the size of the polynomial is measured by the standard complexity-theoretic metric, where exponents contribute the logs of their values rather than the values themselves (as in the sparse and dense metrics). With the standard metric, even the problem of determining whether a collection of multivariate polynomials has a nontrivial greatest common divisor, an easier problem than factoring, becomes NP-hard, as does the problem of telling whether the product of a set of polynomials is divisible by a given polynomial. See [AN6] and [AN7] in [G&J].

Defenders of randomization can, however, point to arenas where it is probably a win. These can be found in such fields as automata theory [14], VLSI design [27], data structure manipulation [11], and distributed computing [20,27,33,52]. For a simple example from the last-mentioned domain, consider the problem of electing a leader in a ring of identical communicating processes. This is an impossible task if the processes are deterministic, as can be seen by a simple argument based on symmetry. If the processes can flip coins, however, the election can be accomplished with arbitrarily low probability of failure [20].

For a random sample of further successes of randomization, see [4,22,32,35,45]. Of particular interest is the recent excellent survey by D.J.A. Welsh [51], which I have found helpful in preparing this column, and which covers much of the same material from a slightly different point of view.

### 3. PROBABILISTIC TURING MACHINES AND THEIR COMPLEXITY CLASSES

The formal computational model used to capture the notion of a randomized algorithm is the *probabilistic Turing machine* or *PTM*, a variant on our familiar nondeterministic Turing machine. Recall that the computation of an NDTM can be viewed as a tree of possible computation paths. To turn from nondeterministic to probabilistic Turing machines, we need only revise our formal definition of the “answer” of a computation, and alter our informal interpretation of the meaning of the bifurcations (forks) in the tree. (We assume without loss of generality that each vertex of the tree has at most two successors.) A fork is now interpreted as a random choice between equiprobable alternatives, rather than just a nondeterministic choice. The “answer” specified by a given computation tree is “yes” if more than half of the leaves correspond to yes-states and “no” if more than half correspond to no-states, and “maybe,” otherwise (in contrast to the normal nondeterministic answer, which is “yes” if *any leaf* corresponds to a yes-state and “no” otherwise). In a seminal paper from the mid-70’s [17], J. Gill built on this definition to define a sequence of complexity classes, corresponding to different types of randomized algorithms, which I will now describe. Each class was defined by imposing additional constraints on the PTM and the computation trees it generates.

First, let us restrict ourselves to PTM's whose computation trees have polynomial-bounded depth (with all leaves at the last level) and never yield the overall answer "maybe." If we impose no further restrictions, we obtain a class that Gill calls *PP*, for "Probabilistic Polynomial time." Note that this class allows algorithms that are much less practical than those that we have seen above, since the probability of error could be as large as  $1/2 - (1/2)^n$ , where  $n$  is the instance size, in which case that probability could not be made arbitrarily small without an exponential number of iterations. Thus it is not a great advantage for a problem to be in *PP*. Indeed, it is easy to see that  $NP \cup \text{co-NP} \subseteq PP$ . (It is also easy to see that  $PP \subseteq PSPACE$ .)

If we further constrain the PTM so that, for some fixed  $\epsilon > 0$ , the proportion of leaves with the correct answer is always greater than  $1/2 + \epsilon$ , we define the class *BPP* ("B" for "Bounded away from 1/2"). Within this class we *can* obtain arbitrarily low error probability with a bounded number of iterations. Thus *BPP* corresponds to perhaps the most general notion of randomized algorithm that has practical potential, although as yet I know of no algorithms that use the full power of the class. *BPP* appears to be incomparable to *NP* (i.e., neither class contains the other).

If we modify our definition still further, and require that the computation trees have *no* yes-leaves when the answer is "no," we obtain the class *R* (short for "Random polynomial time"), sometimes called "RP." All the decision problems solved by the randomized algorithms mentioned in the two previous sections belong to this class or its complement (*co-R*). Gill called this class "VPP," but didn't say what the "V" stood for, which may be why the name was so easily supplanted when Adleman and Manders reintroduced the class in [2] and called it *R*. It is easy to see that the class *R* is contained in *NP*. However, although *PP* and *BPP* are by definition closed under complement, *R* does not appear to be. The algorithms of Rabin and Solovay-Strassen show that COMPOSITE NUMBERS is in *R*, but no one has yet been able to show that PRIMES is.

It would thus seem that  $R \cap \text{co-R}$  might be a proper subset of *R*. This intersection class has significance all its own. With a little effort, it can be shown to correspond precisely to our notion of those problems solvable by polynomial-time Las Vegas algorithms. With a little additional effort, it can be shown also to correspond to the set of all problems that can be solved with zero probability of error in expected polynomial time (for each instance), a class Gill calls *ZPP*. Thus two of the types of randomized algorithms discussed in the previous section turn out to be equivalent! (See [46] for an even stronger equivalence that holds for the problem of integer factorization.)

To summarize the inclusion relations mentioned above, we have

$$P \subseteq ZPP = R \cap \text{co-R} \subseteq R \subseteq \left\{ \begin{array}{l} NP \\ BPP \end{array} \right\} \subseteq PP \subseteq PSPACE.$$

It is conjectured that all these inclusions are proper. Besides the standard type of evidence for such conjectures (i.e., our inability to prove the converses), there is also a bit more solid support for the conjecture that  $R \neq NP$ . First there is the result, due to Adleman [1], that if a language is in  $R$ , it can be recognized using circuits of polynomially bounded size. (For more on the Boolean circuit model of computation, see the [June, 1983] column and its references.) The proof provides no uniform method for constructing these circuits, given the size of the problem to be solved. Even so, it is thought unlikely that all problems in  $NP$  (in particular the  $NP$ -complete ones) can be solved this simply, especially since this would imply that the polynomial hierarchy  $PH$  collapses into  $\Sigma_2^P$  [21], which would be another surprise. (The same evidence makes it unlikely that  $NP \subseteq BPP$ , since this would imply  $R = NP$  [23].)

A second result bearing on the  $R$  versus  $NP$  question is stated by Yao in [53]. It says that if certain presumably hard problems are indeed hard (in an appropriate technical sense), then every problem in  $R$  can be solved in time  $o(2^{n^\epsilon})$  for every  $\epsilon > 0$ , a property not expected to hold for  $NP$ . The “presumably hard” problems in question are two of the currently leading candidates for use in public key cryptosystems: factoring the product of two  $n$ -bit primes, and computing the *discrete logarithm*, i.e., given a prime  $p$ , a generator  $g$  for  $Z_p$ , and a positive integer  $y < p$ , find an  $x$  such that  $g^x \equiv y \pmod{p}$ . The functions described in these two problems, although seemingly hard to compute, are easy to invert, and this “one-way” property is the source of their cryptographic usefulness. Yao’s result actually says that if there is *any* suitably “strong” one-way function, then  $R$  has the above property. Whether this should be taken as evidence for  $R \neq NP$  or for the non-existence of strong one-way functions remains open to debate.

The debate is not likely to end soon, as there is strong evidence that our standard proof techniques will be no more effective here than they have been on the question of  $P$  versus  $NP$ . This evidence comes in the form of relativization results, analogous to those mentioned in Chapter 7 of [G&J]. To be specific, whereas it has long been known [9] that there is an oracle  $D$  that yields  $P^D = NP^D$  (and hence  $= ZPP^D = R^D$ ), Rackoff [34] has shown that there also exist oracles  $E$  and  $F$  such that  $P^E = R^E \neq NP^E$  and  $P^F \neq R^F = NP^F$ . Hunt has shown that even  $ZPP$  can be separated from  $P$  by an oracle [19]. These latter two oracles, however, are relatively rare, as Bennett and Gill [10] have shown that, relative to a random oracle  $A$ ,  $P^A = ZPP^A = R^A = BPP^A$  and all are properly contained in  $NP^A$ , with the containments  $NP^A \cup co-NP^A \subseteq PP^A \subseteq PSPACE^A$  also being proper. (My literature search has not, however, been able to turn up oracles  $X$  validating such possibilities as  $ZPP^X \neq R^X$ ,  $R^X \neq BPP^X$ , or  $R^X \neq NP^X \cap co-NP^X$ .)

Two other relations between the above randomized complexity classes and more traditional ones are of note. The first concerns  $BPP$  and the polynomial hierarchy  $PH$ . We have already noted that if  $NP \subseteq BPP$ , this hierarchy collapses to its second level and is contained in  $\Sigma_2^P$ . What is perhaps more surprising is

that BPP itself is in  $\Sigma_2^P$  [25,42], and if  $\text{NP} \subseteq \text{BPP}$ , the polynomial hierarchy actually collapses into BPP [23,54]. It is not immediately clear what randomization has to do with the relativized computations that define the levels of the hierarchy (recall that  $\Sigma_2^P = \text{NP}^{\text{NP}}$ ). There are even more connections, however, and the reader is referred to [18,23,45,54,55] for more details, including attempts to define a “randomized hierarchy.”

The other interesting correspondence is between the class PP and Valiant’s class #P of hard enumeration problems, discussed in [G&J], Chapter 7. The similarity between these two classes is best illustrated by the versions of SATISFIABILITY that are complete for each class. Given an instance of SATISFIABILITY, it is #P-complete to determine the number of satisfying truth assignments. It is PP-complete to determine whether more than half the truth assignments are satisfying [17]. These are clearly closely related problems. Indeed, it is not difficult to see that they are Turing reducible to each other, and that  $P^{PP} = P^{\#P}$ , i.e., an oracle to a problem in PP is just as good as an oracle to a problem in #P [5]. It might even be the case that  $PP = P^{\#P}$ , but this would depend on whether Turing reductions are more powerful than polynomial transformations in this domain, an as-yet-unanswered question.

Having whetted the reader’s appetite with a PP-complete problem, I might well be expected to continue by listing problems that are complete for BPP, R, and ZPP. Unfortunately, none are known. We are so accustomed to the notion that complexity classes have complete problems that this may come as something of a shock. However, those who have tried to find complete problems for these particular classes have encountered major technical problems. For the class R at least, these difficulties can be explained by a relativization result. Sipser [41] has constructed an oracle  $C$  such that  $R^C$  does not have complete sets. Since the types of simulation techniques commonly used in our generic transformations do relativize ( $\text{NP}^X$  has complete sets for all oracles  $X$  [41]), this implies that if a complete set for ordinary (unrelativized)  $R$  is to be found, non-standard methods will be required. ( $\text{NP} \cap \text{co-NP}$  is in the same boat as R, as shown by a second oracle in [41], and the techniques would seem to extend to BPP in a straightforward manner [43].)

To find another probabilistic complexity class with complete problems, we will have to turn to *space-bounded* rather than time-bounded computations. Here there are interesting definitional problems. In the time-bounded case we could assume that all leaves of the computation tree had the same probability (and that these probabilities summed to 1), since the depth of the tree was polynomially bounded. This cannot be done in the space-bounded case, since there can be a positive probability that the PTM goes into an infinite loop (which is the equivalent of a final “maybe-state”). Even worse, because of the probabilistic nature of the computation, there can be leaves at arbitrarily large depths in the tree. Thus we assign to each leaf  $v$  a probability equal to  $(1/2)^{k(v)}$ , where  $k(v)$  equals the number of forks encountered on the path to  $v$  from the vertex

corresponding to the initial configuration.

A space-bounded probabilistic complexity class, related to PP much the way PSPACE is related to NP, is “probabilistic PSPACE.” It consists of those languages recognizable by PTM’s that operate in polynomial-bounded space and accept an input if and only if the set of yes-leaves has total probability exceeding  $1/2$ . (Note that this definition, due to Gill [17], involves a slightly different acceptance criterion than the symmetric one he used in defining PP. In particular, if neither the no-leaves nor the yes-leaves have total probability exceeding  $1/2$ , the answer is now “no” rather than “maybe.”) One might prefer to call this class “PPSPACE” for short, but, unfortunately, that name was never reserved for it, and has recently been claimed for a class with a different, though still probabilistic, definition.

This terminological aggrandizement took place in C. H. Papadimitriou’s paper on “games against an indifferent opponent” [29], mentioned in the [Dec. 1983] column. According to that paper, PPSPACE is a time-bounded rather than a space-bounded class, and is defined in terms of the same polynomially bounded computation trees that were used in the definition of PP. Here, however, an input is accepted if and only if there is a way to “prune” the tree (by cutting off one of the two possibilities for each fork occurring at an even level of the tree) so that more than half of the remaining leaves are yes-leaves. (This corresponds to a player making deterministic choices of moves at the even levels, while his “indifferent opponent” makes random choices of moves at the odd levels.)

Fortunately, the confusion that might result from two different interpretations of the term “PPSPACE” is more superficial than real: they both turn out to define the same class! Moreover, that class is simply PSPACE, and so we don’t need the name PPSPACE anyway (and there is an ample supply of ready-made candidates for the complete problems I promised). For probabilistic PSPACE, this equivalence follows from two fundamental results about space-bounded probabilistic computations. Let *NSPACE* and *DSPACE* stand for “nondeterministic” and “deterministic” space, respectively, and consider any space bound  $S(n) = \Omega(\log n)$ . Then,

(a)  $\text{NSPACE}(S(n)) \subseteq \text{probabilistic SPACE}(S(n))$  [17] and

(b)  $\text{probabilistic SPACE}(S(n)) \subseteq \text{DSPACE}(S(n)^6)$  [40].

Inclusion (a) holds even if it is required that the error probability of the PTM be less than  $1/2 - \epsilon$  for some fixed  $\epsilon > 0$ . (If, in addition, the error probability is required to be 0 when the answer is no, in analogy with R, then the inclusion can be replaced by equality [17].) The exponent in (b) has recently been improved to 2 [12], so that (a) and (b) together provide an alternative (and much more complicated) proof of Savitch’s theorem [37] that  $\text{NSPACE}(S(n)) \subseteq \text{DSPACE}(S(n)^2)$ . A final nontrivial result for such space bounds is that probabilistic  $\text{SPACE}(S(n))$  is closed under complement [36,39].

When the space bound  $S(n) = \log n$ , there is much more of interest that can

be said. Unfortunately, I have only log space left in which to say it, and so pointers to the literature must suffice [4,35,36].

#### 4. RANDOM TRANSFORMATIONS

Although the NP-complete problems are, by definition, the “hardest” problems in NP, there may well be hard problems in NP that are not NP-complete. Indeed, a classical result of Ladner shows that this must be the case. Assuming  $P \neq NP$  [24], there must be problems that are in  $NP - P$  that are not NP-complete. One proposed method for identifying such problems is to consider problems that are complete for NP under a weaker notion of reducibility than the “many-one” reducibility of polynomial transformation. To this end, Adleman and Manders in [2] proposed the concept of “ $\gamma$ -reducibility,” based on the notion of a *nondeterministic* reduction, and defined a problem in NP to be “ $\gamma$ -complete” if all problems in NP were  $\gamma$ -reducible to it.

This was discussed in [G&J], but a second notion of reducibility, proposed in that same paper and intermediate between  $\gamma$ -reducibility and many-one reducibility, went unmentioned. This intermediate reducibility was called *R-reducibility*. One reason for its omission in [G&J] was that, whereas [2] could give three problems that were “ $\gamma$ -complete” but not known to be NP-complete, to get a candidate for an analogous “R-complete but not NP-complete” problem, the authors had to assume the extended Riemann hypothesis. R-reducibility is worth mentioning in this column, however, because that “R” stands for “random,” and because there have been some new developments.

The machine model behind both  $\gamma$ - and R-reducibility is the polynomial-time nondeterministic Turing machine with outputs. At each yes-leaf of the computation tree, the contents of the TM’s worktape is viewed as the corresponding “output.” In an ordinary polynomial transformation from decision problem  $X$  to decision problem  $Y$ , the TM is deterministic. Given an instance  $x$  of  $X$  as input, the TM’s computation tree has but one leaf, and hence has a uniquely defined output. That output must be an instance  $y$  of  $Y$  such that the answer for  $x$  is yes if and only if the answer for  $y$  is yes. For a  $\gamma$ -reduction, there may be more than one yes-leaf, and hence more than one possible output. The definition requires only that there be at least one such output and that for each possible output  $y$ , the answer for  $y$  be yes if and only if the answer for  $x$  is. R-reducibility differs from this only in that we now require that the proportion of leaves that are yes-leaves exceed some fixed constant  $c > 0$ .

It is not difficult to prove that R-reducibility, like  $\gamma$ - and many-one reducibility, is transitive. It is also easy to show that

$$X \propto Y \Rightarrow X \propto_R Y \Rightarrow X \propto_\gamma Y.$$

It is conjectured that the implications do not go the other way. Hence a problem that is  $\gamma$ -complete or R-complete may not necessarily be NP-complete.

However, such a problem may still be hard. An NP-complete problem (a) cannot be in P unless  $P = NP$ , but it also (b) cannot be in co-NP unless  $NP = \text{co-NP}$ , (c) cannot be in R unless  $NP = R$ , and (d) cannot be in ZPP unless  $NP = \text{ZPP}$ . Property (b) still holds for  $\gamma$ -complete problems, and (b), (c), and (d) all still hold for R-complete ones [2]. None of the consequences is currently believed to be likely, and so R-completeness, like  $\gamma$ -completeness, offers a new way to prove “intractability” results.

When these notions were introduced, however, they seemed to have a major limitation. The only problems that seemed to profit from the added power provided by  $\gamma$  and R-reductions were number-theoretic ones. This limitation may recently have been removed, at least in the case of R-reducibility, by a proof that the problem below is R-complete. I say “may” because, of course, the problem might still be NP-complete, a question I leave to interested readers as the “Open Problem of the Month” (along with the still-open question of the status of the number-theoretic  $\gamma$ -complete problems mentioned in [2]).

#### [1] ENCODING BY TM

INSTANCE: Strings  $x, y$  over a finite alphabet  $\Sigma$ , positive integer  $K$ .

QUESTION: Is there a deterministic TM with  $K$  or fewer states that, when started with  $x$  on its work tape and with its read-write head located at the leftmost symbol of  $x$ , writes  $y$  on its output tape in just  $|y|$  steps?

*Reference.* Vazirani and Vazirani [48]. R-reduction from MONOTONE 3-SAT.

*Comment.* R-complete for NP, even if  $|\Sigma| = 5$ ; not known to be NP-complete. However, the variant in which we ask whether the task can be performed by a one-way finite state transducer with  $K$  or fewer states is known to be NP-complete, even if  $|\Sigma| = 3$  [48]; this variant is used as an intermediate step in the reduction proving R-completeness.

#### REFERENCES

1. L. ADLEMAN, Two theorems on random polynomial time, in “Proceedings 19th Ann. Symp. on Foundations of Computer Science,” pp. 75-83, IEEE Computer Society, Los Angeles, 1978.
2. L. 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.
3. L. M. ADLEMAN, C. POMERANCE, AND R. S. RUMELY, On distinguishing prime numbers from composite numbers, *Annals Math.* **117** (1983), 173-206.
4. R. ALELIUNAS, R. M. KARP, R. J. LIPTON, L. LOVÁSZ, AND C. RACKOFF, Random walks, traversal sequences and the complexity of maze problems, in “Proceedings 20th Ann. Symp. on Foundations of Computer Science,” pp. 218-223, IEEE Computer Society, Los Angeles, 1979.

5. D. ANGLUIN, On counting problems and the polynomial-time hierarchy, *Theor. Comput. Sci.* **12** (1980), 161-173.
6. A. O. L. ATKIN AND R. G. LARSON, On a primality test of Solovay and Strassen, *SIAM J. Comput.* **11** (1982), 789-791.
7. L. BABAI, Monte-Carlo algorithms in graph isomorphism testing, manuscript (1979).
8. L. BABAI, Talk presented at the 21st Annual Symposium on Foundation of Computer Science, San Juan, Puerto Rico, October 29, 1979 (not in the Proceedings).
9. T. BAKER, J. GILL, AND R. SOLOVAY, Relativizations of the P =? NP question, *SIAM J. Comput.* **4** (1975), 431-442.
10. C. H. BENNETT AND J. GILL, Relative to a random oracle  $A$ ,  $P^A \neq NP^A \neq co-NP^A$  with probability 1, *SIAM J. Comput.* **10** (1981), 96-113.
11. J. L. BENTLEY, D. F. STANAT, AND J. M. STEELE, Analysis of a randomized data structure for representing ordered sets, in "Proceedings 19th Ann. Allerton Conf. on Communication, Control, and Computing," pp. 364-372, Department of Electrical Engineering and the Coordinated Science Laboratory, University of Illinois, Urbana, Ill., 1981.
12. A. BORODIN, S. COOK, AND N. PIPPENGER, Space-bounded probabilistic computation, to appear.
13. S. FORTUNE AND J. HOPCROFT, A note on Rabin's nearest neighbor algorithm, *Inform. Process. Lett.* **8** (1979), 20-23.
14. R. FREIVALDS, Probabilistic machines can use less running time, in "Information Processing 77," (B. Gilchrist, Ed.), pp. 839-842, IFIP, North-Holland, 1977.
15. M. FURST, J. HOPCROFT, AND E. LUKS, A subexponential algorithm for trivalent graph isomorphism, *Congressus Numerantium* **28** (1980), 421-446.
16. Z. GALIL, C. M. HOFFMAN, E. M. LUKS, C. P. SCHNORR, AND A. WEBER, An  $O(n^3 \log n)$  deterministic and an  $O(n^3)$  probabilistic isomorphism test for trivalent graphs, in "Proceedings 23rd Ann. Symp. on Foundations of Computer Science," pp. 118-125, IEEE Computer Society, Los Angeles, 1982.
17. J. GILL, Computational complexity of probabilistic Turing machines, *SIAM J. Comput.* **6** (1977), 675-695.
18. H. HELLER, On relativized exponential and probabilistic complexity classes, manuscript (1983).
19. J. W. HUNT, "Topics in Probabilistic Complexity," Ph.D. Dissertation, Department of Electrical Engineering, Stanford University, Stanford, Calif., 1978.
20. A. ITAI AND M. RODEH, Symmetry breaking in distributed networks, in "Proceedings 22nd Ann. Symp. on Foundations of Computer Science," pp. 150-158, IEEE Computer Society, Los Angeles, 1981.
21. R. M. KARP AND R. J. LIPTON, Some connections between nonuniform and uniform complexity classes, in "Proceedings 12th Ann. ACM Symp. on Theory of Computing," pp. 302-309, Association for Computing Machinery, New York, 1980.
22. R. M. KARP AND M. LUBY, Monte-Carlo algorithms for enumeration and reliability problems, in "Proceedings 24th Ann. Symp. on Foundations of Computer Science," pp. 56-64, IEEE Computer Society, Los Angeles, 1983.
23. K.-I. KO, Some observations on the probabilistic algorithms and NP-hard problems, *Inform. Process. Lett.* **14** (1982), 39-43.
24. R. E. LADNER, On the structure of polynomial time reducibility, *J. Assoc. Comput. Mach.* **22** (1975), 155-171.
25. C. LAUTEMANN, BPP and the polynomial hierarchy, *Inform. Process. Lett.* **17** (1983), 215-218.
26. A. K. LENSTRA, Factoring multivariate polynomials over finite fields, in "Proceedings 15th Ann. ACM Symp. on Theory of Computing," pp. 189-192, Association for Computing Machinery, New York, 1983.
27. K. MEHLHORN AND E. M. SCHMIDT, Las Vegas is better than determinism in VLSI and distributed computing, in "Proceedings 14th Ann. ACM Symp. on Theory of Computing," pp. 330-337, Association for Computing Machinery, New York, 1982.

28. G. L. MILLER, Riemann's hypothesis and tests for primality, *J. Comput. System Sci.* **13** (1976), 300-317.
29. C. H. PAPADIMITRIOU, Games against nature, in "Proceedings 24th Ann. Symp. on Foundations of Computer Science," pp. 446-450, IEEE Computer Society, Los Angeles, 1983.
30. M. O. RABIN, Probabilistic algorithms, in "Algorithms and Complexity: New Directions and Recent Results" (J. F. Traub, Ed.), pp. 21-39, Academic Press, New York, 1976.
31. M. O. RABIN, Probabilistic algorithm for testing primality, *J. Number Theory* **12** (1980), 128-138.
32. M. O. RABIN, Probabilistic algorithms in finite fields, *SIAM J. Comput.* **9** (1980), 273-280.
33. M. O. RABIN,  $N$ -process mutual exclusion with bounded waiting by  $4 \log_2 N$ -valued shared variable, *J. Comput. System Sci.* **25** (1980), 66-75.
34. C. RACKOFF, Relativized questions involving probabilistic algorithms, *J. Assoc. Comput. Mach.* **29** (1982), 261-268.
35. J. H. REIF, Symmetric complementation, *J. Assoc. Comput. Mach.* **31** (1984), 401-420.
36. W. L. RUZZO, J. SIMON, AND M. TOMPA, Space-bounded hierarchies and probabilistic computations, in "Proceedings 14th Ann. ACM Symp. on Theory of Computing," pp. 215-223, Association for Computing Machinery, New York, 1982.
37. W. J. SAVITCH, Relationship between nondeterministic and deterministic tape classes, *J. Comput. System Sci.* **4** (1970), 177-192.
38. J. T. SCHWARTZ, Fast probabilistic algorithms for verification of polynomial identities, *J. Assoc. Comput. Mach.* **27** (1980), 710-717.
39. J. SIMON, Space-bounded probabilistic Turing machine complexity classes are closed under complement, in "Proceedings 13th Ann. ACM Symp. on Theory of Computing," pp. 158-167, Association for Computing Machinery, New York, 1981.
40. J. SIMON, On tape-bounded probabilistic Turing machine acceptors, *Theor. Comput. Sci.* **16** (1981), 75-91.
41. M. SIPSER, On relativization and the existence of complete sets, in "Automata, Languages, and Programming," pp. 523-531, Lecture Notes in Computer Science, Vol. 140, Springer, Berlin, 1982.
42. M. SIPSER, A complexity theoretic approach to randomness, in "Proceedings 15th Ann. ACM Symp. on Theory of Computing," pp. 330-335, Association for Computing Machinery, New York, 1983.
43. M. SIPSER, private communication (1984).
44. R. SOLOVAY AND V. STRASSEN, A fast Monte-Carlo test for primality, *SIAM J. Comput.* **6** (1977), 84-85.
45. L. STOCKMEYER, The complexity of approximate counting, in "Proceedings 15th Ann. ACM Symp. on Theory of Computing," pp. 118-125, Association for Computing Machinery, New York, 1983.
46. M. TOMPA, "Probabilistic factoring algorithms can be made errorless," Report No. 83-09-01, Department of Computer Science, University of Washington, Seattle, Wash., 1983.
47. S. M. ULAM, "Adventures of a Mathematician," pp. 198-201, Charles Scribner's Sons, New York, 1976.
48. U. V. VAZIRANI AND V. V. VAZIRANI, A natural encoding scheme proved probabilistic polynomial complete, *Theor. Comput. Sci.* **24** (1983), 291-300.
49. J. VON ZUR GATHEN, Factoring sparse multivariate polynomials, in "Proceedings 24th Ann. Symp. on Foundations of Computer Science," pp. 172-179, IEEE Computer Society, Los Angeles, 1983.
50. J. VON ZUR GATHEN AND E. KALTOFEN, Polynomial-time factorization of multivariate polynomials over finite fields, in "Automata, Languages, and Programming," pp. 250-265, Lecture Notes in Computer Science, Vol. 154, Springer, Berlin, 1983.

51. D. J. A. WELSH, Randomised algorithms, *Disc. Applied Math.* **5** (1983), 133-145.
52. A. C.-C. YAO, Some complexity questions related to distributed computing, in "Proceedings 11th Ann. ACM Symp. on Theory of Computing," pp. 209-213, Association for Computing Machinery, New York, 1979.
53. A. C. YAO, Theory and application of trapdoor functions, in "Proceedings 23rd Ann. Symp. on Foundations of Computer Science," pp. 80-91, IEEE Computer Society, Los Angeles, 1982.
54. S. ZACHOS, Robustness of probabilistic computational complexity classes under definitional perturbations, *Information and Control* **54** (1982), 143-154.
55. S. ZACHOS AND H. HELLER, A decisive characterization of BPP, manuscript (1983).