

## The NP-Completeness Column: An Ongoing Guide

DAVID S. JOHNSON

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

This is the eighteenth edition of a (usually) 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 2D-150, AT&T Bell Laboratories, Murray Hill, NJ 07974 (or to dsj@btl.csnet). 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 If the results are unpublished, please state explicitly that you are willing for them to be mentioned. 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.

### COMPUTING IN THE MATH DEPARTMENT: PART I

This is the first of several columns about recent interactions between traditional mathematics and complexity theory. In this first column, I discuss new complexity results for problems from the discrete side of the math department, with emphasis on number theory and related topics from the field of algebra.

Algebra and number theory are hardly new domains as far as NP-completeness is concerned. Eighteen NP-hard problems are listed under the heading “Algebra and Number Theory” in the now-ancient [G&J]. Nevertheless, these fields of “pure” mathematics have drawn increasing attention from complexity theorists of late, inspired in part by their new computational applications. For instance, number theory, which Gauss is reputed to have called the queen of mathematics “because of its supreme uselessness” [20], now lies at the heart of much current work in the practical field of cryptography.

Perhaps the most exciting of the new results are the new algorithms that apply the theory of elliptic curves (and other algebraic varieties) to primality testing. These will be discussed in Section 1. Section 2 covers the status of other

number-theoretic problems, primarily those having to do with the solvability of polynomial equations in modular, integer, or rational arithmetic. Section 3 turns from number theory to symbolic algebra, considering the analogues for polynomials of the prime-testing and composite-factorization problems discussed in Section 1. Many of the recently-developed polynomial-time algorithms for these problems depend on an approximation algorithm for a combinatorial problem about short vectors in lattices. This still-open problem is highlighted in the concluding Section 4.

## 1. PRIMES

For many years now, a central open problem in computational number theory has been that of determining the complexity of the PRIMALITY problem: “Given a positive integer  $n$ , is it the case that  $n$  has no positive integer divisors other than 1 and itself?” In its complementary form (COMPOSITE NUMBER), this is one of the six remaining open problems from the list of twelve in [G&J], and despite the exciting new results I am about to discuss, it is still not known whether it can be solved in polynomial time. The new results do, however, make it seem even more unlikely that PRIMALITY is NP-hard.

Admittedly, NP-hardness already seemed fairly unlikely. For instance, it would imply that  $\text{NP} = \text{co-NP}$  and, in addition, that the Extended Riemann Hypothesis (the “ERH,”) is false. (Pratt [50] proved that PRIMALITY is in  $\text{NP} \cap \text{co-NP}$ , and Miller [46] devised a polynomial-time algorithm that would recognize the prime numbers if the ERH were true.) Furthermore, it is unlikely that NP-hard problems can be solved in time  $O(|x|^{c \log \log |x|})$ , where  $|x|$  is the size of instance  $x$ , but as Adleman, Pomerance, and Rumely showed in [5], primality *can* be determined within this bound (with  $|x| = \log n$  taken as the size of instance  $n$ ). Finally, there is the fact that COMPOSITE NUMBERS  $\in \text{RP}$ , i.e., composite numbers can be recognized by random polynomial-time algorithms (such as those of Solovay and Strassen [58] and of Rabin [51]).

As described in the [Sept. 1984] column, such an “RP” algorithm runs in polynomial time, but is allowed the use of a random number generator to help it make some of its decisions. Thus for each instance  $n$ , the algorithm defines a probability distribution over the potential outputs “yes” and “no,” rather than a unique output. For the algorithm to be “RP,” that distribution must obey the following constraints: If the answer is yes (in this case, if  $n$  is composite), the algorithm must output “yes” with probability exceeding  $1/2$ . If the answer is no (i.e.,  $n$  is prime), the algorithm must always output “no.” Thus there is a “one-sided” probability of error: if the output obtained from a run of the algorithm is “yes,” we know for sure that  $n$  is composite. Even if the output is “no,” however, it still may be that  $n$  is composite, although the probability that this happens for a composite  $n$  is less than  $1/2$ .

In itself, this is not a very comforting margin of error. However, by

performing a sequence of independent runs of the algorithm on input  $n$ , we can reduce the probability of a mistaken identification to arbitrarily low levels. If any single run says that  $n$  is composite, then we know for sure that  $n$  is composite, whereas if all runs say that  $n$  is prime, then we know that either  $n$  is prime or an event of probability less than  $2^{-k}$  has occurred, where  $k$  is the number of runs. This suffices for many applications. It remains mildly unsatisfying, however. One can never be totally sure that “primes” identified in this way are truly prime.

In [18], Goldwasser and Kilian describe a new algorithm that goes a long way toward rectifying this situation. Their algorithm, like that of Solovay and Strassen, has one-sided error, only now the error lies in the opposite direction. The algorithm always tells the truth when it asserts primality, and, indeed, can output a polynomial-time verifiable proof of primality to back up its claim.

Pratt [50] had already shown in 1975 that polynomial-time verifiable proofs of primality exist for all primes  $p$ . However, *finding* such proofs seemed to be difficult; Pratt’s proofs require a factorization of  $p-1$ , and factoring may well be intractable even if primality testing is not. Goldwasser and Kilian circumvent this difficulty by using a *new* form of primality proof. Whereas Pratt’s proofs used well-known results about the multiplicative group of integers mod  $p$ , the new proofs exploit a more elaborate number-theoretic group, based on “elliptic curves.”

Suppose that  $n$  is not divisible by 2 or 3 (a property that clearly holds for most primes) and that  $A$  and  $B$  are such that  $4A^3 + 27B^2$  is not congruent to 0 mod  $n$  (a minor technicality). Then the *elliptic curve*  $E_n(A,B)$  is the set of pairs  $(x,y) \in Z_n \times Z_n$  that satisfy the equation  $y^2 \equiv x^3 + Ax + B \pmod{n}$ . If  $n$  is a prime, and we adjoin a special “point at infinity”  $I$ , we obtain a group with  $I$  as the identity under a rather baroque definition of addition. The definition is based on the fact that no straight line can intersect an elliptic curve more than three times. Thus, for any two elements  $e_1 = (x_1, y_1)$  and  $e_2 = (x_2, y_2)$  of  $E_n(A,B)$ , the straight line through  $e_1$  and  $e_2$  (or tangent to the curve at  $e_1$  if  $e_1 = e_2$ ) can contain at most one other element of  $E_n(A,B)$ . Surprisingly, when  $n$  is prime such an element is guaranteed to exist for any choice of  $e_1$  and  $e_2$  that have differing second components (and hence do not lie on the same vertical line). In such a case, the group sum  $e_1 + e_2$  is obtained from this unique element by negating its second component. (If  $e_1$  and  $e_2$  do have the same second component, the sum is the point at infinity  $I$ .)

A polynomial-time verifiable proof that  $n$  is prime then consists of (1) a pair  $(A,B)$  satisfying the above modular restrictions (easily checked), (2) an integer  $q$  with  $(n+1)/2 + n^{1/2} > q > n^{1/2} + 2n^{1/4} + 1$ , (3) an element  $e \neq I$  of  $E_n(A,B)$  such that  $qe$  ( $e$  added to itself  $q-1$  times) equals  $I$ , and, recursively, (4) a proof that  $q$  is prime. Given such  $A, B, q$ , and  $e$ , the theory of elliptic curves implies that  $n$  can have no non-trivial prime factors smaller than  $q$ . Since  $q > n^{1/2}$ , this implies that  $n$  is a prime. Such a “proof” can be verified in time  $O(\log^4 n)$ , since the

recursion depth is  $O(\log n)$ , the number of group addition steps needed to compute  $qe$  by repeated doubling is  $O(\log n)$ , and each group addition step costs  $O(\log^2 n)$ .

To generate proofs of the above form (probabilistically), Goldwasser and Kilian rely on an  $O(\log^9 n)$  deterministic algorithm of Schoof [57] (recently improved to  $O(\log^8 n)$  [42]) for computing the order (size) of an elliptic group, and the fact that this order must lie in the range  $n + 1 \pm 2\sqrt{n}$ . They randomly guess pairs  $(A, B)$  until they find a pair that satisfies the modularity condition and is such that Schoof's algorithm applied to  $E_n(A, B)$  says the order is of the form  $kq$ , where  $k$  is small and  $q$  passes the Rabin or Solovay-Strassen primality test with a very high margin of safety. Next, they randomly guess elements of  $E_n(A, B)$  until they find one satisfying (3) above. Recursively, they then apply the same approach to proving primality of  $q$ , unless  $q$  is sufficiently small to use one of the  $O((\log q)^{c \log \log q})$  deterministic primality algorithms mentioned above. Assuming  $n$  is prime, the densities of good pairs  $(A, B)$  and good elements  $e$  are such that this approach is highly likely to succeed, provided that  $\Omega(\log^{11} n)$  time overall is allowed for repeated guesses.

There is one important qualification, however (not counting any reservations one might have about that 11 in the exponent). Goldwasser and Kilian can show only that the densities are right for *almost all* primes (for each sufficiently large  $k$ , all but roughly  $O(2^{-k^{1/\log \log k}})$  of the primes of length  $k$ ). To extend their claims of efficacy to *all* primes, they must rely on an unproven (but believable) conjecture about the density of primes. Fortunately, this reliance on a number-theoretic conjecture is not as potentially damaging as the reliance of Miller's primality-testing algorithm [46] on the Extended Riemann Hypothesis. If the ERH is false, Miller's algorithm can give wrong answers. With Goldwasser-Kilian, one can always check the proof that is provided along with any assertion of primality, and hence have confidence in the answer. Even if Schoof's algorithm (defined only for primes) should somehow manage to produce a plausible answer when given a composite as input, and even if an  $e \neq 1$  could then be guessed for which  $qe = 1$  ( $qe$  need not even be computable if  $n$  is not prime), one can still not generate a complete proof that  $n$  is a prime unless  $n$  indeed *is* prime. The danger here is just that for some primes, no answer or proof may be forthcoming, even after repeated trials. Thus, the Goldwasser-Kilian algorithm may not be a true "RP" algorithm for PRIMALITY; for that, its probabilistic guarantees would have to hold for *all* primes.

Since the Goldwasser-Kilian algorithm was first announced, there have been two significant attacks on its drawbacks. First, as to the unimpressive running time, A. Atkin has devised a modified approach which, instead of randomly choosing  $A$  and  $B$  and then testing the order of  $E_n(A, B)$ , attempts directly to generate an  $(A, B)$  with an appropriate order. The approach is still probabilistic, but the use of the  $\log^8 n$  Schoof algorithm is bypassed, and an implementation of the resulting algorithm seems to run more quickly than the best of the previous

primality testing algorithms [42]. (Unfortunately, the Goldwasser-Kilian analysis no longer applies to this revised algorithm, and so no equally strong theoretical guarantees are known to hold for it.)

The second improvement is due to Adleman and Huang [1], who have announced a *provably* RP algorithm for PRIMALITY, although one that is even less practical than the (probably) RP algorithm of Goldwasser and Kilian. The essence of the Adleman-Huang approach is to reduce the job of proving  $n$  prime to that of proving a *larger* (rather than smaller) number prime, where the larger number can be *any* prime within a derived interval. By iterating this “expansion” a few times, they eventually arrive at a derived interval that must contain a high proportion of primes for which Goldwasser-Kilian is guaranteed to work. The expansion process, like the Goldwasser-Kilian reductions, is itself randomized, and the mathematics involved goes a few steps beyond elliptic curves, using 2-dimensional algebraic groups called “abelian varieties.” As a theoretical consequence of the Adleman-Huang result, we can conclude that PRIMALITY is in  $RP \cap \text{co-RP}$ , and that there thus is a “Las Vegas” type randomized algorithm for it, i.e., one that runs in expected polynomial time for all inputs and never tells a lie. (Simply alternate runs of Solovay-Strassen and Adleman-Huang until one of them says “yes.”)

## 2. OTHER NUMBERS

Although we seem to be narrowing in on the complexity of PRIMALITY, many other problems in computational number theory remain open. (A recent survey [4] lists 36 open problems in the field.) Foremost is the abovementioned problem of INTEGER FACTORING, which we can state simply as follows: Given a composite  $n$ , find an integer  $m$ ,  $1 < m < n$ , such that  $m$  divides  $n$  without remainder. Even for randomized algorithms, the best running times currently available are exponential in  $(\log n \log \log n)^{1/2}$ . Note that this is substantially worse than the  $\exp(\log \log n \log \log \log n)$  running time of the *deterministic* algorithm of [5] for primality testing. Elliptic curves seem to help here too, however. (In fact, this is where they were first applied.) The factoring algorithm that uses them, due to H. Lenstra [43], has a conjectured running time that beats the above factoring bound whenever  $n$  has a small factor. Nevertheless, cryptographic schemes based on the difficulty of factoring remain safe from these algorithms (at least such schemes as the RSA scheme for public key cryptography [53], which relies on our inability to factor numbers that are the product of two *large* primes).

Here are two other popular number-theoretic problems with open complexity and apparent cryptographic significance.

DISCRETE LOG: Given positive integers  $g$ ,  $b$ , and  $n$ , find an integer  $x$  such that  $g^x \equiv b \pmod{n}$ , if such an  $x$  exists. (This is related to cryptographic schemes from [8,13]. Factoring is randomly reducible to it, and the current best algorithms for it have running times like those for factoring. A once-popular choice for cryptographic schemes was to consider the analogous problem with the logarithms taken over the finite field  $\text{GF}(n)$ , where  $n$  was of the form  $2^m$ . For such fields, however, logarithms can be found in time bounded by a substantially smaller exponential than that needed for factoring, and security may be compromised [11,47].)

QUADRATIC RESIDUOSITY: Given positive integers  $a$  and  $n$  with  $\text{gcd}(a,n) = 1$ , is there an integer  $x$  such that  $x^2 \equiv a \pmod{n}$ ? (This is related to cryptographic schemes such as those in [19]. If  $n$  is prime or its prime factorization is given, the problem is in P [44]. If the ERH holds, the problem of *finding* such an  $x$  is in P if either  $n$  is prime or its complete factorization is given [44], although in the latter case finding the *smallest* such  $x$  is NP-hard ([AN8] in [G&J]). Even if the ERH does not hold, such an  $x$  can be found in random polynomial time if  $n$  is prime [4].)

The complexities of these and the many other open number-theoretic problems surveyed in [4] are linked by an extensive network of polynomial reductions, both deterministic and probabilistic. For details, see [4,60].

As might be conjectured on the basis of the results presented so far, most of the current research into the complexity of number-theoretic problems deals with problems that are likely to be *easier* than the NP-complete problems. Indeed, a large body of current research concentrates on problems known to be solvable in polynomial time, and asks to what extent their computation can be sped up by massive parallelism. See [15,52] for a sampling of this literature.

One NP-hard problem that *has* been the focus of further research is QUADRATIC DIOPHANTINE EQUATIONS ([AN8] in [G&J]). In that problem, proved NP-complete in [44], one is given nonnegative integers  $a$ ,  $b$ , and  $c$ , and is asked if there exist positive integers  $x$  and  $y$  such that  $ax^2 + by = c$ . This naturally raises the question of the complexity of determining solvability (by nonnegative integers) of *other* forms of binary (i.e., two-variable) quadratic Diophantine equations. It appears that a variety of complexities are possible. For instance, the solvability of  $axy + y = c$  is “ $\gamma$ -complete” [2] (see also [G&J, Section 7.1]). The solvability of  $x^2 - a^2y^2 = c$  is “UR-complete” [3] (see also the [Sept. 1984] column). Polynomial-time solvability for  $\gamma$ - and UR-complete problems does not appear to imply  $P = NP$  (although it does imply  $NP = \text{co-NP}$  and  $NP = RP$ , respectively). Hence determining the solvability of these latter two equations just might be a little easier than determining that of the equation  $ax^2 + by = c$  from QUADRATIC DIOPHANTINE EQUATIONS. Could there be a binary quadratic equation that is harder?

A result of Lagarias [34] suggests a candidate. For certain values of  $d$ , the

“non-Pellian” equation  $x^2 - dy^2 = -1$  can only have solutions whose lengths are exponential in the length of  $d$ . Thus for such an equation the standard proof of solvability (exhibit a pair  $x, y$  and demonstrate that it satisfies the equation) is not “short.” This suggests that the solvability of non-Pellian equations might not be in NP. Having made this suggestion, Lagarias subsequently disposed of it, showing in [33] that solvability could be certified without exhibiting an actual solution, and that in fact the solvability of *any* binary quadratic Diophantine equation, i.e., any equation of the form  $ax^2 + bxy + cy^2 + dx + ey + f = 0$ , is in NP. (Solvability of the above non-Pellian equations themselves is in  $\text{NP} \cap \text{co-NP}$  [34].)

Thus to obtain harder problems one must go to more variables and/or higher degrees. (One doesn’t have to go too far; with 13 variables and higher degrees one obtains undecidability [45]). If one goes to more variables but lower degree (i.e., to linear Diophantine equations), one retains NP-completeness so long as one requires the solutions to be *non-negative* integers (as in SUBSET SUM). If, however, arbitrary integers are allowed as solutions, polynomial time suffices to test solvability (and find solutions if solvable), even for systems of *simultaneous* linear equations [30]. This latter result is not obvious, due to the possibility once again that numbers might get too large (in this case the intermediate results). If a system of linear equations has *no* common solution, finding the maximum number of equations that *can* be solved simultaneously turns out to be NP-hard, via a transformation from SUBSET-SUM [32]. The same holds true if we change our field of operation from the integers to  $\text{GF}[2]$ , but now the transformation is from MAXIMUM 2-SATISFIABILITY [21]. (Note in passing that solvability of simultaneous *quadratic* Diophantine equations is once again undecidable [45].)

Let us return to the case of a single non-linear equation,  $\sum_{i=1}^n a_i x^i = 0$ , and consider what happens when we allow non-integral (indeed, irrational) solutions. One of the most time-honored questions of mathematics is, given such an equation, can such a solution be computed “by radicals”? In other words, is there a straight-line program for computing a solution in which each step applies one of the following operations: addition, subtraction, multiplication, division, or extracting the  $k$ th root, for some positive integer  $k$  (the “radical” operation). An operand can be any integer (including the  $a_i$ ) or the result of a previous step. We ignore the fact that our machine model may not be able to cope with the arbitrary-precision complex numbers that can result from the root-taking operations. The key requirement is that, were we able to compute in exact complex arithmetic, the output of the final step would have to be a solution to the equation. This must hold no matter which of the  $k$  choices one makes for the value of  $\alpha^{1/k}$ , so long as one always makes the same choice when evaluating the same radical.

For example, recall the formula  $(-b + \sqrt{b^2 - 4ac})/2a$  for one root of the degree-2 polynomial  $ax^2 + bx + c = 0$ . Similar formulae exist for degrees 3 and 4,

but not in general for degrees 5 or greater. In the nineteenth century, Galois proved that, given the coefficients  $a_i$ , it was *decidable* whether the equation could be “solved by radicals.” (In the process, he invented Galois groups and much other interesting machinery [12].) Unfortunately, his algorithm had double exponential running time, and no subsequent approach has been better than exponential, until very recently. Now, however, a polynomial-time algorithm has at last been discovered that, given a polynomial over the rationals, not only checks for solvability by radicals, but also generates the desired straight-line program. The result is due to Landau and Miller [37], and in a sense can be viewed as a spin-off of the recent flurry of research into polynomial factorization (see Section 3). Its running time, unfortunately, is no better than that of those algorithms. (In Section 3 we will see just how unfortunate that is.) Still open is the question of whether one can in polynomial time create a “formula” for the solution, i.e., an expression using (possibly nested) radical signs (as has been done for the quadratic case). The potential difficulty here is again that of *size*: straight-line programs can be more concise than expressions since they don’t have to repeat radicals that are used twice as operands. In particular, it is not even clear that the individual primitive  $k$ th roots of unity can be expressed by polynomially bounded formulae of this type.

### 3. POLYNOMIALS

In the above discussion we were interested in producing symbolic programs for “computing” the roots of an equation, rather than in generating the roots themselves (in, say, binary notation). Such an approach is unavoidable when the binary representation of the roots may be infinite in length. Suppose, however, we make things easy on ourselves by asking only for those roots that lie in the field  $F$  over which the polynomial is specified. If the elements of  $F$  have nice (finite) descriptions (as is the case when  $F$  is a finite field or the field of rationals), then we may legitimately hope to generate such descriptions for the roots themselves. One way of doing so is by factoring the polynomial (expressing it as a product of irreducible polynomials over  $F$ ). Then each root corresponds to a factor of degree 1. Thus root-finding is no harder than factoring for polynomials. Another problem that is no harder than factoring is *irreducibility-testing*, the analogue for polynomials of prime-testing for integers. (A polynomial is irreducible if it has no factors of degree 1 or greater other than constant multiples of itself.)

In this context, there has been much pleasant news lately. For all the types of fields normally considered, it is now known that factoring can be performed in polynomial time. I alluded to these new results in two past columns ([March 1983] and [Sept. 1984]), but in this section I shall tell the story more fully, and bring it up to date. (For more thorough and historical surveys, see [24,28].)

The major inspiration for all these new results was a now-famous 1982 paper

by A. Lenstra, H. Lenstra, and L. Lovász [41] (commonly referred to as “ $L^3$ ”), which presented the first known polynomial-time algorithm for factoring univariate polynomials with rational (or, equivalently, integer) coefficients. Assuming that multiplications are performed according to the classical  $\Theta(n^2)$  algorithm, the running time for the  $L^3$  algorithm was  $O(n^{12} + n^9|a|^3)$ , where  $n$  is the degree of the polynomial and  $|a|$  is the length of the largest coefficient (assuming integer coefficients). This has since been sped up to  $O(n^8 + n^5|a|^3)$  by Schönhage [54], with a further speedup possible if the  $O(n \log n \log \log n)$  multiplication techniques of [55] are used.

When  $L^3$  appeared, machinery was already in place [23,24] for extending the results to multivariate polynomials. (In part, this machinery depends on constructive analogues of Hilbert’s “Irreducibility Theorem,” which says that if a  $k$ -variable polynomial over the integers is irreducible, then there are infinitely many ways to substitute integers for  $k - 1$  of the variables so that an irreducible polynomial in the single remaining variable is obtained.) In almost no time we had polynomial-time algorithms for factoring multivariate polynomials over the rationals [25,38], over algebraic number fields [36,39], and over finite fields [10,17,40] thus covering all the abovementioned standard possibilities.

In the case of finite fields, a far more ancient source than  $L^3$  is worth mentioning. Factorization of *univariate* polynomials over finite fields was known to be doable in polynomial time more than a decade before  $L^3$ . In 1967, Berlekamp showed that it could be done in time polynomial in  $n$  and  $q$ , where  $q$  is the size of the field [6], and in 1970 he showed that a randomized algorithm could do the factoring in time polynomial in  $n$  and  $\log q$  [7]. (The former algorithm is in fact used as a subroutine in the  $L^3$  algorithm for factoring over the rationals.) The dependence of the running times of the multivariate algorithms of [10,17,40] on  $q$  is the same as that for Berlekamp’s algorithms, with randomization required if the bound is to be in terms of  $\log q$  rather than  $q$ . (Note that  $\log q$  more accurately reflects the size of coefficients when we are dealing with  $\text{GF}[q]$  for large  $q$ .)

There is more to the plot, however. When we say “polynomial time” in the context of problems about polynomials (including the solvability by radicals problem of the previous section), we have not told the whole story. There are many different ways of representing a polynomial  $f(x)$ , and what is “polynomial-bounded” for one encoding may not be for another. For instance, all the above multivariate results assume that the input size is measured by the “dense” metric, i.e., under the assumption that polynomials are represented by the sequence (array in the multivariate case) of their coefficients, including zero coefficients. If  $n_i$  is the maximum degree with which variable  $x_i$  occurs in any term, this means that in the  $k$ -variable case we require only that the running time be bounded by a polynomial in  $\prod_{i=1}^k (n_i + 1)$  and  $|a|$ . (And in [10,17,25,36,38,39,40] that’s all we get, with running times that are worse than would be obtained by substituting  $\prod_{i=1}^k (n_i + 1)$  for  $n$  in the  $L^3$  running times.) This dense-metric definition of “polynomial time” is particularly generous for

*sparse* multivariate polynomials, i.e., polynomials in which most of the coefficients are 0. For instance, under the dense metric even a *linear*-time algorithm would be allowed to spend  $\Omega(3^k)$  time on the polynomial  $\sum_{i=1}^k x_i^2$ .

For such sparse polynomials, a more traditional notion of “size” would be that obtained from the “concise” representation, in which only the terms with nonzero coefficients are described, and these are presented by writing both coefficients and exponents in binary notation. For instance, we could represent an  $m$ -term,  $k$ -variable polynomial by a sequence of  $(k+1)$ -tuples, one for each term, giving the term’s coefficient in binary and, for each variable, the exponent of that variable in the term, again in binary. Unfortunately, no factoring algorithm, randomized or not, is known to run in polynomial time for this “concise” metric.

This is of course to be expected, since the number of factors of a polynomial  $p$ , and the coefficient sizes and numbers of terms in the individual factors, might all be exponential in the size of  $p$  under the concise metric. However, under this metric, even simple decision problems are known to be NP-hard, for instance, the NON-TRIVIAL GREATEST COMMON DIVISOR problem ([AN7] in [G&J]). In a restricted form, recently proved NP-hard by Plaisted [48] (who also proved the original result), this problem asks “Given two concisely represented univariate polynomials with integer coefficients, do they have a non-trivial GCD, i.e., is there a polynomial of degree at least 1 that divides both?” Plaisted also has extended his techniques to concisely represented polynomials over finite fields, although here he must settle for  $\gamma$ -completeness results and proofs that problems are NP-complete “relative to a slowly utilized oracle” [49].

Very few problems having to do with polynomials in concise form are known to be easy. Indeed, of those listed in [G&J], only NON-DIVISIBILITY OF A PRODUCT POLYNOMIAL [AN6] is known to be in NP, and that proof is non-trivial [48]. Even a simple problem like, “Given concisely represented polynomials  $p(x)$  and  $q(x)$ , is the first a divisor of the second?” is not obviously in NP, since the quotient polynomial might be dense and hence exponential in the sizes of  $p(x)$  and  $q(x)$ . (This problem is not known to be NP-hard either, although Plaisted has recently shown that it *is* NP-hard to determine whether a quotient has a nonzero constant term and whether the degree of a remainder obeys a given lower bound [48].)

We thus appear to be asking too much when we ask for algorithms that are polynomial under the concise metric. In fact, experts in the field would be pleased to have running times that are merely polynomial in the concise metric plus a second parameter, the *degree* of the polynomial. (The “degree” of a polynomial is the maximum, over all terms of the polynomial, of the sum of the variable degrees in the term.) The degree seems a reasonable parameter to add, given that it provides an attainable upper bound on the number of factors. Moreover, it still leaves us with a stricter definition of polynomial time than that

provided by the dense metric. (Polynomial time for  $\sum_{i=1}^k x_i^2$  would be polynomial in  $2k$ , not in  $3^k$ .)

A factor of a multivariate polynomial  $p$  may still be of exponential size, however, even when the degree of  $p$  is taken into account. New randomized algorithms of [16,26,27] avoid this difficulty by adopting a representation for polynomials that can be even more compact than the “concise” representation: the “straight-line program.” These straight-line programs (in contrast to those of the previous section that constructed roots) construct the polynomial itself, as a symbolic entity. The operations are now addition, subtraction, multiplication, and division, and the operands can be symbols for the variables, arbitrary scalars (field elements), and results of previous operations. As an example, the program “ $y_1 = x + 1, y_2 = y_1 \cdot y_1$ ” constructs the polynomial “ $x^2 + 2x + 1$ ” over the rationals, and the polynomial “ $x^2 + 1$ ” over GF[2]. The *size* of such a program can be thought of as simply the number of steps times the size of (number of bits needed to represent) the largest scalar used in any step.

Given a polynomial in concise form, it is easy to derive a straight-line program for it with no more than a polynomial blow-up in size, using repeated squaring to obtain the needed powers of the variables. On the other hand, note that the polynomial  $(x + 1)^n$  has straight-line size  $O(\log n)$ , whereas its concise representation has size  $\Omega(n)$  since, when multiplied out, the polynomial has  $n + 1$  terms. (For such dense polynomials, the dense and concise metrics are polynomially related.)

The randomized algorithms of [16,26,27] for “straight-line” polynomials work both for finite fields and for the rationals. They should also extend to algebraic number fields in a straightforward manner [26]. For technical reasons, they are slightly less attractive than the randomized algorithms of Section 1, in that there is always a possibility that the answer they output is incorrect, and at present there is no general deterministic polynomial-time method known for distinguishing the correct answers from the false ones (no “short proofs” are provided). Fortunately, the error probability  $\epsilon$  can be made arbitrarily small, at a cost of a running-time increase that is only polynomial in  $\log(1/\epsilon)$ .

The first of the algorithms to be obtained was due to von zur Gathen [16]. Given a straight-line program  $S$  that constructs a multivariate polynomial  $f[S]$ , it produces (with high probability) the *pattern* of the factors of  $f[S]$  (the degrees and multiplicities of each). More recently, Kaltofen has found an algorithm that does the whole job, producing straight-line programs for all the factors in time polynomial in  $size(S)$  and  $degree(f[S])$  [27]. The fact that some of these factors might be dense, and hence have exponentially many terms in their concise representations, does not effect the running time, since the straight-line programs representing them continue to be polynomially bounded. In the case of polynomials over the rationals, however, the running time does depend on the size of the largest coefficients in the original polynomial, and a bound on this size must at present be provided as input to both algorithms. A bound on  $degree(f[S])$  is

also needed, but this can be provided automatically by using as a preprocessor a probabilistic algorithm for computing degrees [16].

A simpler algorithm, in [26], produces a straight-line program for the GCD of two polynomials (both given as straight-line programs), again in polynomial time independent of the density of the concise representation for the output, and this time independent of the coefficient sizes of the input polynomials. Still another algorithm deals with the difficulties arising from the fact that divisions are allowed in the model, and hence not only polynomials, but also arbitrary rational functions can be constructed. (Divisions allow the more compact representation of certain polynomials, such as  $x^n + x^{n-1} + \dots + x + 1 = (x^{n+1} - 1)/(x - 1)$ .) Given a straight-line program, this algorithm generates with high probability two minimal-degree polynomials  $f$  and  $g$  such that the rational function constructed is  $f/g$  [27]. Thus, if one does not know *a priori* that a given straight-line program constructs a polynomial, one can use this algorithm to settle the question (at least with high probability).

A pleasant side-effect of the above factoring results is that they allow us to derive polynomial-time algorithms that use the concise representation for their inputs and outputs. Given concisely represented polynomials, we first construct (in polynomial time, as indicated above) straight-line programs that compute them. We then apply the appropriate algorithm from [27] to compute straight line programs for the factors (or GCD). By an algorithm in [26] based on ideas of Zippel [61], these can then be turned back into concise representations in time polynomial in the sizes of those representations and their degrees. This of course re-introduces the potential blow-up in running time that can be caused by dense factors of sparse polynomials, but the overall running time is polynomial in the degree of the original polynomial and the combined sizes (under the concise metric) of it and all its factors. Moreover, if one wants, one can specify a bound  $t$ , and only convert to concise form those factors that have  $t$  or fewer terms [27].

To conclude this section, let me return briefly to the question of the relative complexities of factoring and irreducibility testing. These are polynomially related (by default) in those cases where we already have deterministic polynomial-time factoring algorithms. There currently *are* distinctions, however, in those cases where our factoring algorithms are probabilistic. Consider the situation in which we measure running times in terms of the concise metric and the degree. The randomized multivariate factoring algorithm of [27] returns its “factorizations” with only a high probability of correctness and no proof. Irreducibility-testing of multivariate polynomials, on the other hand, is in RP, both for finite fields and for the rationals [16], assuming the input is the concise representation of the polynomial together with its degree in unary. (As far as I know, it remains open whether irreducibility-testing in this situation is also, like it’s number-theoretic analogue, in  $RP \cap co-RP$ .) Under the dense metric, an even greater distinction holds in the case of finite fields  $GF[q]$  for large primes  $q$ .

Here, the only way we currently can factor in time polynomial in the size of the polynomial and  $\log q$  is via randomized algorithms, even in the univariate case. Irreducibility, however, can be tested in *deterministic* polynomial time, even for multivariate polynomials [29].

#### 4. INTEGER LATTICES

Although I have mentioned many open problems in this column, I have reserved the honor of “Open Problem of the Month,” for an algebraic question with an unexpected connection to polynomial factorization.

##### [OPEN] SHORTEST VECTOR IN A LATTICE

INSTANCE: Collection of vectors  $\bar{v}_1, \dots, \bar{v}_n$ , each a member of  $\mathbf{Z}^n$ , integer bound  $B > 0$ .

QUESTION: Is there a nonzero vector  $\bar{a} = (a[1], \dots, a[n]) \in \mathbf{Z}^n$  such that if  $\bar{x}_a = \sum_{i=1}^n a[i] \bar{v}_i$ , then the Euclidean length  $\|\bar{x}_a\| = (\sum_{i=1}^n x_a[i]^2)^{1/2}$  is  $B$  or less?

*Comment.* The “integer lattice” to which the name of this problem refers is the set of all integer combinations of the vectors  $v_i$ . In the optimization version of the problem, we are merely looking for the nonzero member of this lattice with smallest Euclidean norm. The decision problem is solvable in polynomial time if  $B = 0$ , as it then just becomes the problem of solving homogeneous linear equations over  $\mathbf{Z}$ . The related “nearest vector” problem, in which one is in addition given a target vector  $\bar{x}_0$ , and asked if an  $\bar{a}$  exists such that  $\|\bar{x}_a - \bar{x}_0\| \leq B$ , is NP-complete [59]. The same pattern of “open,” “polynomial,” and “NP-complete” holds if one replaces  $\mathbf{Z}$  by GF[2] in all the above. (The open problem for GF[2] is equivalent to EVEN COVER, the “Open Problem of the Month” for [June 1982]. The NP-hardness of “nearest vector” over GF[2] is an exercise for the reader.) The original “shortest vector” problem over  $\mathbf{Z}$  also becomes NP-hard if one replaces the Euclidean norm  $\|\bar{x}_a\|$  by the  $L_\infty$  norm, i.e.,  $\max\{x_a[i] : 1 \leq i \leq n\}$  [59].

The open status of SHORTEST VECTOR IN A LATTICE means that we do not currently have an efficient algorithm for the corresponding optimization problem. There are efficient *approximation* algorithms for this problem, however. The first one that was both fast enough and good enough to have useful applications appeared in the  $L^3$  paper. (Subsequent improvements and variants have appeared in [54,56].) These approximation algorithms are called “basis reduction” algorithms, because they work by finding a “reduced” basis for the abovementioned integer lattice. The “short” vector they produce is the smallest vector in the reduced basis.

The worst-case guarantees these algorithms provide are at first glance not all

that impressive, compared to the kinds of guarantees one is familiar with for problems like bin packing and the traveling salesman problem. All that is claimed is that  $\|\bar{x}_a\|$  is no worse than  $c^n$  times the optimal value, for some constant  $c > 1$  depending on the algorithm used [41,54,56]. Fortunately, in the applications of interest one can arrange that gaps larger than this must exist between the optimal vector and any extraneous short vectors that might also be present.

The relevance of such approximation algorithms to polynomial factorization is most easily illustrated by looking at a variant on the  $L^3$  algorithm, described in [31,54]. In order to factor an integer-coordinate polynomial  $p$  over the rationals, one starts by computing a very close approximation to a complex root  $\alpha$  of  $p$  (using standard techniques). This root  $\alpha$  will be the root of some minimal irreducible polynomial  $f$  that is a factor of  $p$ . Given a guess  $d$  for the degree of  $f$ , one constructs an instance of SHORTEST VECTOR in which the every lattice vector has  $d + 3$  components: the  $d + 1$  coefficients of some polynomial  $g$  of degree  $d$  or less, and the imaginary and real parts of the value of  $g$  evaluated at our approximation to  $\alpha$  (suitably scaled so that everything is integral). If there is a polynomial of degree  $d$  with  $\alpha$  as a root, it will yield by far the shortest vector in the lattice, since only for it will the last two components be sufficiently close to 0. Once the correct value of  $d$  is found by repeated iterations of the above, one has an irreducible factor of  $p$ , can divide out, and can proceed by induction to complete the factorization.

Other domains of application for basis reduction algorithms include the number-theoretic question of ‘‘Simultaneous Diophantine approximation’’ [35], the breaking of certain public-key cryptosystems [9], and the design of ‘‘strongly polynomial’’ algorithms for combinatorial problems like minimum cost network flow [14]. Page allotments being what they are, I shall have to postpone discussion of these to a later date.

#### REFERENCES

1. L. M. ADLEMAN AND M.-D. A. HUANG, Recognizing primes in random polynomial time (Abstract), manuscript (1986).
2. L. M. ADLEMAN AND K. MANDERS, Reducibility, randomness and intractability, in ‘‘Proceedings 20th Ann. Symp. on Foundations of Computer Science,’’ pp. 397-410, IEEE Computer Society, Los Angeles, 1977.
3. L. M. ADLEMAN AND K. MANDERS, Reductions that lie, in ‘‘Proceedings 9th Ann. ACM Symp. on Theory of Computing,’’ pp. 151-163, Association for Computing Machinery, New York, 1979.
4. L. M. ADLEMAN AND K. S. MCCURLEY, Open problems in number theoretic complexity, manuscript (1986).
5. L. M. ADLEMAN, C. POMERANCE, AND R. S. RUMELY, On distinguishing prime numbers from composite numbers, *Annals Math.* **117** (1983), 173-206.

6. E. R. BERLEKAMP, Factoring polynomials over finite fields, *Bell Syst. Tech. J.* **46** (1967), 1853-1859.
7. E. R. BERLEKAMP, Factoring polynomials over large finite fields, *Math. Comp.* **24** (1970), 713-735.
8. M. BLUM AND S. MICALI, How to generate cryptographically strong sequences of pseudorandom bits, *SIAM J. Comput.* **13** (1984), 850-864.
9. E. F. BRICKELL, Breaking iterated knapsacks, in "Advances in Cryptology: Proc. of CRYPTO 84," pp. 342-358, Lecture Notes in Computer Science, Vol. 196, Springer, Berlin, 1985.
10. A. L. CHISTOV AND D. Y. GRIGORYEV, "Polynomial-time factoring of the multivariate polynomials over a global field," Report No. Preprint E-5-82, LOMI, Leningrad, 1982.
11. D. COPPERSMITH, Fast evaluation of logarithms in fields of characteristic two, *IEEE Trans. Inform. Theory* **IT-30** (1984), 587-594.
12. H. EDWARDS, "Galois Theory," Springer-Verlag, New York, 1984.
13. T. EL GAMAL, A public key cryptosystem and a signature scheme based on discrete logarithms, *IEEE Trans. Inform. Theory* **IT-31** (1985), 469-472.
14. A. FRANK AND E. TARDOS, An application of simultaneous approximation in combinatorial optimization, in "Proceedings 26th Ann. Symp. on Foundations of Computer Science," pp. 459-463, IEEE Computer Society, Los Angeles, 1985.
15. J. VON ZUR GATHEN, Parallel algorithms for algebraic problems, *SIAM J. Comput.* **13** (1984), 802-824.
16. J. VON ZUR GATHEN, Irreducibility of multivariate polynomials, *J. Comput. System Sci.* **31** (1985), 225-264.
17. J. VON ZUR GATHEN AND E. KALTOFEN, Factorization of multivariate polynomials over finite fields, *Math. Comp.* **45** (1985), 251-261.
18. S. GOLDWASSER AND J. KILIAN, Almost all primes can be quickly certified, in "Proceedings 18th Ann. ACM Symp. on Theory of Computing," pp. 316-329, Association for Computing Machinery, New York, 1986.
19. S. GOLDWASSER AND S. MICALI, Probabilistic encryption, *J. Comput. System Sci.* **28** (1984), 270-299.
20. G. H. HARDY, "A Mathematician's Apology," p. 120, Cambridge University Press, Cambridge, England, 1967.
21. J. HASTAD, "An NP-complete problem: Some aspects of its solution and some possible applications," Report No. 16, Institut Mittag-Leffler, Stockholm, Sweden, 1984.
22. E. KALTOFEN, A polynomial reduction from multivariate to bivariate integer polynomial factorization, in "Proceedings 14th Ann. ACM Symp. on Theory of Computing," pp. 261-266, Association for Computing Machinery, New York, 1982.
23. E. KALTOFEN, A polynomial-time reduction from bivariate to univariate integral polynomial factorization, in "Proceedings 23rd Ann. Symp. on Foundations of Computer Science," pp. 57-64, IEEE Computer Society, Los Angeles, 1982.
24. E. KALTOFEN, Factorization of polynomials, in "Computer Algebra, Symbolic and Algebraic Computation," pp. 95-113, B. Buchberger, G. E. Collins, and R. Loos (eds.), Springer-Verlag, Vienna, 1983.
25. E. KALTOFEN, Polynomial reductions from multivariate to bi- and univariate integral polynomial factorization, *SIAM J. Comput.* **14** (1985), 469-489.
26. E. KALTOFEN, Computing with polynomials given by straight-line programs I: Greatest common divisors, in "Proceedings 17th Ann. ACM Symp. on Theory of Computing," pp. 131-142, Association for Computing Machinery, New York, 1985.
27. E. KALTOFEN, Uniform closure properties of P-computable functions, in "Proceedings 18th Ann. ACM Symp. on Theory of Computing," pp. 330-337, Association for Computing Machinery, New York, 1986.

28. E. KALTOFEN, "Polynomial factoring 1982-1986," Report No. 86-19, Department of Computer Science, Rensselaer Polytechnic Institute, Troy, N.Y., 1986.
29. E. KALTOFEN, "Deterministic irreducibility testing of polynomials over large finite fields," Report No. 01818-86, Mathematical Sciences Research Center, Berkeley, Calif., 1986.
30. R. KANNAN AND A. BACHEM, Polynomial algorithms for computing the Smith and Hermite normal forms of an integer matrix, *SIAM J. Comput.* **8** (1979), 499-507.
31. R. KANNAN, A. K. LENSTRA, AND L. LOVÁSZ, Polynomial factorization and nonrandomness of bits of algebraic and some transcendental numbers, in "Proceedings 16th Ann. ACM Symp. on Theory of Computing," pp. 191-200, Association for Computing Machinery, New York, 1984. (*Math. Comp.*, to appear.)
32. M. KOLINEK, "Geometric Methods in Computational Complexity," Doctoral Dissertation, University of Washington, Seattle, Wash., 1985.
33. J. C. LAGARIAS, Succinct certificates for the solvability of binary quadratic Diophantine equations, in "Proceedings 20th Ann. Symp. on Foundations of Computer Science," pp. 47-54, IEEE Computer Society, Los Angeles, 1979.
34. J. C. LAGARIAS, On the computational complexity of determining the solvability or unsolvability of the equation  $x^2 - Dy^2 = -1$ , *Trans. Amer. Math. Soc.* **260** (1980), 485-508.
35. J. C. LAGARIAS, The computational complexity of simultaneous Diophantine approximation problems, *SIAM J. Comput.* **14** (1985), 196-209.
36. S. LANDAU, Factoring polynomials over algebraic number fields, *SIAM J. Comput.* **14** (1985), 184-195.
37. S. LANDAU AND G. L. MILLER, Solvability by radicals is in polynomial time, *J. Comput. System Sci.* **30** (1985), 179-208.
38. A. K. LENSTRA, Factoring multivariate integral polynomials, *Theor. Comput. Sci.* **34** (1984), 207-213.
39. A. K. LENSTRA, Factoring multivariate polynomials over algebraic number fields, in "Mathematical Foundations of Computer Science," pp. 389-396, Lecture Notes in Computer Science, Vol. 176, Springer, Berlin, 1983. (*SIAM J. Comput.*, to appear.)
40. A. K. LENSTRA, Factoring multivariate polynomials over finite fields, *J. Comput. System Sci.* **30** (1984), 235-248.
41. A. K. LENSTRA, H. W. LENSTRA, JR, AND L. LOVÁSZ, Factoring polynomials with rational coefficients, *Math. Ann.* **261** (1982), 515-534.
42. H. W. LENSTRA, JR, "Elliptic curves and number-theoretic algorithms," Report No. 86-19, Mathematisch Instituut, Universiteit van Amsterdam, Amsterdam, 1986.
43. H. W. LENSTRA, JR, Factoring integers with elliptic curves, manuscript (1986).
44. K. MANDERS AND L. ADLEMAN, NP-complete decision problems for binary quadratics, *J. Comput. System Sci.* **16** (1978), 168-184.
45. Y. MATIJASEVIC AND J. ROBINSON, Reduction of an arbitrary Diophantine equation to one in 13 unknowns, *Acta Arith.* **27** (1975), 521-553.
46. G. L. MILLER, Riemann's hypothesis and tests for primality, *J. Comput. System Sci.* **13** (1976), 300-317.
47. A. M. ODLYZKO, Discrete logarithms in finite fields and their cryptographic significance, in "Advances in Cryptology: Proc. EUROCRYPT 84," pp. 224-314, Lecture Notes in Computer Science, Vol. 209, Springer, Berlin, 1985.
48. D. A. PLAISTED, New NP-hard and NP-complete polynomial and integer divisibility problems, *Theor. Comput. Sci.* **31** (1984), 125-138.
49. D. A. PLAISTED, Complete divisibility problems for slowly utilized oracles, *Theor. Comput. Sci.* **35** (1985), 245-260.
50. V. PRATT, Every prime has a succinct certificate, *SIAM J. Comput.* **4** (1975), 214-220.
51. M. O. RABIN, Probabilistic algorithm for testing primality, *J. Number Theory* **12** (1980), 128-138.

52. J. REIF, Logarithmic depth circuits for algebraic functions, in "Proceedings 24th Ann. Symp. on Foundations of Computer Science," pp. 138-145, IEEE Computer Society, Los Angeles, 1983.
53. R. RIVEST, A. SHAMIR, AND L. ADLEMAN, A method for obtaining digital signatures and public-key cryptosystems, *Comm. ACM* **21** (1978), 120-126.
54. A. SCHÖNHAGE, Factorization of univariate integer polynomials by Diophantine approximation and an improved basis reduction algorithm, in "Automata, Languages, and Programming," pp. 436-447, Lecture Notes in Computer Science, Vol. 172, Springer, Berlin, 1984.
55. A. SCHÖNHAGE AND V. STRASSEN, Schnelle Multiplikation grosser Zahlen, *Computing* **7** (1971), 281-292.
56. C. P. SCHNORR, A hierarchy of polynomial time basis reduction algorithms, in "Theory of Algorithms, Coll. Math. Soc. Janos Bolyai, Vol 44," North-Holland, Amsterdam, 1986.
57. R. J. SCHOOF, Elliptic curves over finite fields and the computation of square roots mod  $p$ , *Math. Comp.* **44** (1985), 483-494.
58. R. SOLOVAY AND V. STRASSEN, A fast Monte-Carlo test for primality, *SIAM J. Comput.* **6** (1977), 84-85.
59. P. VAN EMDE BOAS, "Another NP-complete partition problem and the complexity of computing short vectors in a lattice," Report No. 81-04, Department of Mathematics, University of Amsterdam, 1981.
60. H. WOLL, "Reductions among number theoretic problems," Report No. 84-05-02, Department of Computer Science, University of Washington, Seattle, Wash., 1984.
61. R. ZIPPEL, Probabilistic algorithms for sparse polynomials, in "Proc. EUROSAM 79," pp. 216-226, Lecture Notes in Computer Science, Vol. 72, Springer, Berlin, 1979.