

The NP-Completeness Column: An Ongoing Guide

DAVID S. JOHNSON

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

This is the eleventh 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.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 If the results are unpublished, please state explicitly that you would like them to be mentioned in the column.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

An interesting pair of notes is to appear in a forthcoming issue of *IEEE Transactions on Automatic Control*. Both refer to an earlier paper by Pichai, Sezer, and Siljak [35] that proposed a decomposition technique for acyclic graphs, called *input-output decomposition*, which would simplify the analysis of dynamic systems. The first note [40], by R. E. Tarjan, proves that it is NP-complete to tell whether a given graph has such a decomposition and concludes that "the possible benefits of having an input-output decomposition may be outweighed by the difficulty of finding one." The second note [36], by Pichai et al., can only be characterized as an "authors' rebuttal" to the NP-completeness result.

Although this is the first formal rebuttal I have seen, it is certainly not the first time someone has wanted to cry "foul" when his problem was shown to be NP-complete. Often this is simply due to the frustration of seeing a promising new idea crumble into the dust of intractability. However, there are cases where

one can plausibly argue that the NP-completeness result is misleading. This is the approach taken by Pichai et al., who claim that the instances arising in their proposed application will all be so small that one can afford to use exponential time algorithms. Another argument that is often used against the relevance of NP-completeness results to practical applications begins by pointing out that NP-completeness is a worst-case concept, and concludes with a claim that, in practice, the “hard instances” never (or at least hardly ever) arise.

Both these arguments raise the fundamental question of when NP-complete problems can be solved “in practice.” In this column I wish to address the related question of when they can be solved “on average.” (Note that I said *solved*; this column will not deal with the well-studied possibility of circumventing NP-completeness by settling for near-optimal rather than optimal solutions.)

There is, of course, a difference between the two notions of “in practice” and “on average.” Claims about an algorithm’s performance “in practice” are most convincing when supported by extensive tests of the algorithm on real instances from the application in question. Claims about performance “on average,” on the other hand, are mathematical in nature and hence can be supported by theorems. Depending on the relative scarcity of machine cycles and theoreticians, this may be an advantage, although there are trade-offs. To state a theorem about behavior “on average,” one needs to be able to describe mathematically the probability distribution (on instances) over which the average is taken, and this may not always be possible for distributions occurring in practice. To *prove* such a theorem may place even more constraints on the distribution, given the current state of the theorem-proving art. As a result, the set of distributions to which the current results apply may not match up exactly with anyone’s notion of the distributions that occur in practice, and the results themselves should be viewed as suggestive rather than definitive (just as worst case results are only suggestive).

The remainder of this column is divided into two sections. The first surveys results that show NP-complete problems to be easy for given distributions, i.e., to be solvable in polynomial time “on average.” The final section then considers how one might argue that a problem is *hard* “on average,” and discusses an important new result of L. A. Levin (one of the original discoverers of the theory of NP-completeness) that suggests a theoretical analogue of NP-completeness for the average case.

2. SOME AVERAGE RESULTS

Just as there are different kinds of “averages” one can derive from a set of numbers (means, medians, modes, etc.), so there are different ways in which problems can be easy to solve “on average.” Minimally, there should be a polynomial time algorithm that, for all sufficiently large instance sizes, solves the problem with probability at least some fixed constant $c > 0$. (By “solve” I

mean “determine the correct answer and provide a proof that it is correct.”) The next step up in strength is a polynomial time algorithm that solves the problem with probability $1 - \epsilon(n)$ for instances of size n , where $\epsilon(n)$ approaches 0 as n approaches ∞ . This can be referred to as solving the problem “almost always” in polynomial time. The strongest type of result exhibits an algorithm that solves *all* instances and whose *expected* running time is bounded by a polynomial function of n . (Note that such a result implies a result of the previous form.)

This section will contain examples of all three types of results, but let me begin with some examples that illustrate the difference between the latter two. These examples involve a type of graph theoretic probability distribution that is especially natural from a mathematical point of view. Suppose we generate n -vertex graphs G according to the following procedure, in which the parameter $p(n)$ is a function from the positive integers to $[0,1]$. Let the vertices of G be labelled v_1, \dots, v_n . The edge set of G is chosen as follows. For each pair v_i, v_j of vertices, let G contain the edge $\{v_i, v_j\}$ with probability $p(n)$, independently of which other edges are included. (Note that if $p(n)$ is identically equal to $1/2$, then for each n all graphs on the given set of n labelled vertices are equally likely, an appealing property.) Let us now look at how distributions of this form can reduce three well-known problems from presumed intractability in the worst case to polynomial time solvability on average.

HAMILTONIAN CIRCUIT [GT37]. In 1975, Pòsa [37] showed that for sufficiently large n , if $p(n) > \alpha \ln n/n$, where $\alpha > 1$, then G almost surely has a Hamiltonian circuit, i.e., the probability that it has one approaches 1 as n approaches ∞ . (This is interesting from a graph theoretic point of view since, if $\alpha < 1/2$, then almost surely G is not even connected [16].) Thus for such graphs an algorithm for the HAMILTONIAN CIRCUIT problem that always answered “yes” would almost surely be correct, although it wouldn’t “solve” the problem in the sense of providing a proof when it was correct. To do that, one would presumably have to exhibit a Hamiltonian circuit. Fortunately, Pòsa’s proof was constructive, and in 1976 Karp showed that it yielded a polynomial time algorithm that would construct a tour with probability $1 - O(n^{-c})$ for some $c > 0$ depending on α [25]. These results were tightened and generalized in 1977 by Angluin and Valiant [2], who devised an improved algorithm with running time $O(n^2 \log n)$. They also extended the results to DIRECTED HAMILTONIAN CIRCUIT and to the related distribution in which for each n we consider only graphs with exactly $\alpha n \ln n$ edges where $\alpha > 1/2$, and consider all such graphs equally likely.

Note that the above algorithms do not solve all instances of the problem; they can halt without finding a Hamiltonian circuit even when one exists. However, it is easy to convert such an algorithm to one that *does* solve all instances. For example, whenever the algorithm fails to find a circuit, we could call as a subroutine the $O(n^2 2^n)$ dynamic programming algorithm of [23], which is

guaranteed to find a circuit if one exists. Note, however, that such a composite algorithm would not necessarily run in expected polynomial time, even though its running time is $O(n^2 \log n)$ for almost all instances. For all we know from the above bounds, the probability that the subroutine is called may be as large as $\Omega(n^{-c})$, which could yield a contribution as high as $2^n/n^{c-2}$ to the expected running time, rendering it exponential.

To obtain a polynomial expected running time by this approach, we would need to find Hamiltonian circuits with probability $1 - O(2^{-n})$. Note that this is not possible unless $p(n) \geq 1/2$. The reason is almost trivial: for $p(n) = \alpha$, the probability that G has an isolated vertex is at least $(1 - \alpha)^n$ and hence exponentially greater than 2^{-n} when $\alpha < 1/2$. Even if our algorithm first checks for isolated vertices, the case when $p(n) < 1/2$ seems difficult to analyze, as does the case when equality holds. It is not until we have $p(n) \geq 1/2 + \epsilon$ for some $\epsilon > 0$ and all n , that a polynomial expected time algorithm is clearly within reach. (Under this assumption, the degree sequence of G must, with probability $1 - o(2^{-n})$, satisfy the hypothesis of a theorem due to Chvátal [11], in which case G has an easily constructible Hamiltonian circuit).

GRAPH 3-COLORABILITY [GT4]. The answer to the previous problem was almost always “yes.” Here the answer is almost always “no.” Even for quite small values of $p(n)$, most random graphs are not 3-colorable; in fact, most of them contain 4-cliques. The crucial technical result is that the probability that a random graph does not contain a 4-clique is $O(c^{-p(n)^6 n^2})$ for some constant $c > 1$ [8]. Thus one can easily determine whether such a random graph is 3-colorable in expected time $O(n^4)$ even when $p(n)$ is growing only slightly faster than $n^{-1/6}$ (for example, when $p(n) = \log n/n^{1/6}$): First test by exhaustive search whether the graph contains a 4-clique; then, if it does not, use the $O(n^3 2.445^n)$ algorithm of [28] to determine the chromatic number exactly.

For denser distributions, in which $p(n)$ is a fixed constant, a substantially improved expected running time is possible. Wilf [42] has shown that for such distributions the 3-colorability can be solved, not just in expected polynomial time, but in expected *constant time*. One need only perform the standard (and worst-case exponential) backtrack search algorithm; for $p(n) = 1/2$, the expected number of nodes in the search tree is about 197 (mainly because one will normally encounter a 4-clique very quickly). This result generalizes to GRAPH k -COLORABILITY for any fixed k , although the expected size of the search tree grows exponentially as a function of k (for $k = 5$ it is already about 750,000). Moreover, the variance and all higher moments of the running time are also constants, and the results extend to the model in which all graphs with n vertices and m edges are equally likely (so long as $m = \Omega(n^2)$) [4].

Our third example is a problem whose worst-case complexity remains open, although many suspect that it is neither in P nor NP-complete.

GRAPH ISOMORPHISM [OPEN1]. Once again, the answer is almost always “no.” If we fix one graph, the probability that a random graph is isomorphic to it is less than 2^{-cn^2} for any $c < 1/2$. The proof of the above statement is non-constructive (simply divide the number of ways the vertices of an n -vertex graph can be permuted by the total number of graphs on n vertices). Nevertheless, certificates of non-isomorphism are easy to find with probability approaching 1. The probability that the two graphs even have the same number of edges is $O(n^{-1/2})$, and more sophisticated tests, such as computing degree sequences or the spectra of inter-vertex distances, can reveal non-isomorphism with far higher probability. Unfortunately, this probability does not seem to be high enough to make possible an expected polynomial time algorithm that combines a collection of such non-isomorphism tests with a backup algorithm for general graph isomorphism (at least I have been unable to unearth such a result).

Expected polynomial time algorithms *can* be constructed, however, if a different route is taken. The idea is to devise a method for converting graphs into *canonical form*, such that two graphs are isomorphic if and only if they have the same canonical form. Thus canonical forms can serve as certificates of isomorphism as well as non-isomorphism. Building on earlier work, Babai and Kucera in 1980 [3] devised a method of constructing canonical forms in polynomial time that limited the class of exceptions to probability $O(c^{-n \log n / \log \log n})$, thus making expected polynomial time possible.

(It would suffice to apply the $O(c^n)$ general graph isomorphism algorithm of [21] on the exceptions, but Babai and Kucera take the more elegant approach of simply extending their canonical form construction algorithm so that it works on all graphs. They spend exponential time on the graphs not previously covered, but the overall expected running time for the canonical form construction algorithm is *linear* in the expected number of edges in the graph.)

It remains open whether expected polynomial time graph isomorphism algorithms can be constructed for random graphs with lower edge probability, or for the distribution that makes all graphs on n vertices and m edges equally likely.

At this point, the reader may well be inclined to make another cry of “foul.” The results we have seen all depend on distributions that are very one-sided, yielding answers that are either almost always “yes” or almost always “no,” and the stronger the result we wish to prove, the more likely the preferred answer must be. One might argue that this trivializes the results, but as we also saw above, there can still be some interesting graph-theoretic issues involved, especially in going from “almost always” to “expected” polynomial time. Let us now turn to some famous problems that do not deal with graphs, and against which the above objections are (possibly) less potent.

SATISFIABILITY [LO1]. Perhaps the most famous algorithm for telling whether a conjunctive normal form formula F is satisfiable is the “Davis-Putnam Procedure” (DPP) [14], and there has recently been a flurry of

seemingly conflicting reports on the average-case behavior of this algorithm. The conflicts arise because the results are not only based on different assumptions about probability distributions, but also concern different versions of the algorithm.

DPP is a backtrack search procedure, augmented with a few shortcuts. The two main shortcuts are (1) the *unit-clause* rule, which says that a literal should be set true if it occurs in a clause all by itself, and (2) the “pure literal” rule, which says that a literal should be set true if its negation never occurs. Even with these shortcuts, DPP still takes exponential time in the worst case. However, Goldberg, Purdom, and Brown [19] have recently shown that, for a certain type of probability distribution on formulae, DPP runs in expected polynomial time.

In this distribution, which we can denote as $G(n, m, p)$, $0 < p \leq 1/2$, we generate n random clauses on m variables as follows: For each variable v_i , $1 \leq i \leq m$, a random clause will contain either the literal v_i , the literal \bar{v}_i , or neither, with probabilities of p , p , and $1 - 2p$, respectively. These choices are made independently for the different variables, and each of the n random clauses is generated independently of the others. Goldberg et al. [19], extending earlier work of Goldberg alone [18], consider the variant on DPP in which the unit-clause rule is omitted and the variables are treated in a pre-specified order, with the pure literal rule only applied when the *current* variable is “pure” (restrictions imposed mainly to facilitate the proof). They show that this variant has an expected running time that is polynomial in n and m (although it is exponential in $1/p$). Note that for certain extremes of the parameters, we once again get instances for which the answers almost always go the same way. If m is fixed, the probability that F is unsatisfiable goes to 1 as n goes to ∞ . On the other hand, it is shown in [17] that if m grows as fast as the $n^{1/k}$ for some k , then F is almost always satisfiable (by almost all truth assignments). However, between these extremes there may well be a relationship between m and n (say $m = c \log n$) such that “yes” and “no” answers both occur with positive probability. By the above result, the modified DPP still runs in expected polynomial time for such a combination.

Note, however, that if p is fixed, then letting m grow results in clauses whose expected length $2pm$ can be arbitrarily large. If one considers this unreasonable, and chooses to let p be proportional to $1/m$ so that the expected length is constant, the expected running time becomes exponential in m . This is not necessarily as bad as “almost always exponential,” but a result of this latter form is obtained in [17] for a related distribution in which all clause lengths equal a fixed constant, given that one further handicaps DPP by *totally* omitting the pure literal rule. (For our original distribution, omitting the pure literal rule also causes trouble, leading to an expected running time that is exponential in m even for fixed p [19].)

If the reader is a bit confused by the various incomparable combinations of probability models and algorithmic variants mentioned for SATISFIABILITY, he should be glad that LINEAR PROGRAMMING is now known to be in P, and hence can be omitted from the current discussion. The [March 1983] column mentioned two models for which variants of the worst-case exponential time Simplex algorithm had been proved to run in some version of “expected polynomial time” [9,39]. Since then at least two more combinations of probability models and Simplex variants have been shown to yield “expected polynomial” running times [1,22] (see also [41]). Few of these results are strictly comparable, and although they all provide insight, they also all have drawbacks that prevent them from providing a complete explanation of why the standard implementations of the Simplex algorithm perform so well in practice. When and if the dust ever settles in this area, I may provide a more detailed summary of the results. (This digression would be incomplete, however, if it failed to point out the *real* news on the linear programming front: N. Karmarkar’s new *worst-case* polynomial time algorithm, which improves on the ellipsoid method by a factor of $n^{2.5}$ [24].) Let us now go on to another problem discussed in the [March 1983] column.

SUBSET SUM [SP13]. This problem was mentioned in the [March 1983] column because of its use in the proposed Merkle-Hellman “knapsack” public key cryptosystem [34], a system whose security had just been seriously put into question by an algorithm of Shamir [38] for recovering the “trapdoor” information. A new and complementary result, due to Lagarias and Odlyzko [27], raises even more doubts. It shows that, for certain distributions, SUBSET SUM can be solved almost always in polynomial time, even without the trapdoor information, and hence its presumed intractability is not widespread enough to serve as a basis for guaranteeing a code’s unbreakability.

Recall that an instance of SUBSET SUM consists of a target sum t and a collection a_i , $1 \leq i \leq n$, of potential summands, and we ask whether there is a subset of the a_i ’s that sums to t . To “prove” that the answer is “yes” one must exhibit the subset, and this is what the new algorithm attempts to do, using the “basis reduction” algorithm of [29] as a subroutine. The algorithm runs in polynomial time, and succeeds “almost always” for distributions of the following kind. Given d , let all length n integer vectors \bar{a} with $1 \leq a_i \leq 2^{n/d}$ be equally likely and let all length n binary vectors \bar{e} with $0 < \sum_{i=1}^n e_i < n$ also be equally likely. A “random” n -item instance is constructed by independently choosing one vector of each type, letting the a_i ’s be the potential summands, and letting the target sum be $t = \sum_{i=1}^n a_i e_i$. For sufficiently small d (provably for $d < 1/n$ and perhaps for larger values, although not for $d > 0.645$), the algorithm succeeds “almost always.” (An alternative, less rigorously analyzed algorithm, due to Brickell [10], also seems promising for low density SUBSET SUM instances.)

Note that, for the given distribution, the decision problem version of SUBSET SUM is trivial; the answer isn't just almost always "yes," it is *always* "yes" — there always is a subset of the a_i 's that sums to t . To "solve" the problem, however, we must find this subset, which isn't made any easier by the fact that we know the subset exists: If we had a polynomial time algorithm that, when restricted to instances of SUBSET SUM for which the answer is "yes," always found the desired subset, we could use it on arbitrary instances of SUBSET SUM to determine whether the answer was "yes" or "no" (the answer is "yes" if and only if the algorithm finds a subset that does sum to t), and hence P would equal NP.

These results complement Shamir's original result, as Shamir's algorithm was effective mainly for systems in which the largest of the a_i 's was bounded by $2^{n/d}$ for relatively *large* values of the density d . (Lagarias [26] showed that the approach could be implemented in time polynomial in n for any fixed d , but was exponential in $1/d$.)

KNAPSACK [MP9]. Although the Merkle-Hellman "knapsack" cryptosystem does not actually have anything to do with what is commonly known as the "knapsack problem," there is an average case result, due to Goldberg and Marchetti-Spaccamela [20], that does. The result is weaker than those discussed so far in that the polynomial time algorithms presented "solve" the problem only with probability exceeding a fixed non-zero lower bound, not "almost always." It is also stronger, however, in that the algorithms solve the *optimization* form of the problem, rather than just the decision problem. Thus our requirement that the algorithm must provide a "certificate" along with each answer becomes much more onerous. That certificate can't just be a simple structure or configuration that forces the answer to be "yes" or "no," but must be a proof of optimality.

In the optimization form of the knapsack problem, an instance can be viewed as a sequence of pairs of rational numbers (s_i, v_i) , $0 < s_i, v_i < 1$, $1 \leq i \leq n$, where s_i is the "size" of item i and v_i is its "value," together with a "knapsack capacity" C . The goal is to find a subset of the items whose total value is maximum over all subsets whose total size is C or less. The distributions over which averages are to be taken view the pairs (s_i, v_i) as points in the unit square, and choose the points uniformly out of that square. (Thus irrational values for the parameters are allowed; an interesting question is whether this result can be extended to the more realistic case of limited precision.) To avoid the dependency effects that would arise if n were fixed in advance, the points are chosen by a Poisson process with n as the *expected* number picked. This makes the numbers of points in two disjoint regions independent of each other. The probability that a region of area A contains exactly h points is $e^{-nA} (nA)^h / h!$. In addition to n , the distribution has a parameter β , and the knapsack capacity is chosen to be βn . Note that the problem is trivial for $\beta \leq 0$ (in which case none of the items will fit in the knapsack) and for $\beta > 1/2$ (in which case, with

probability approaching 1 as n approaches ∞ , *all* the items will fit).

The new results show that for any β , $0 < \beta < 1/2$ and any $\varepsilon > 0$, there is a polynomial time algorithm that, for all sufficiently large n , solves the knapsack problem with probability $1 - \varepsilon$. The algorithm's running time is exponential in $1/\varepsilon$, and so these results do not seem to lead directly to a polynomial time algorithm that almost always solves the problem. It also should be noted that the algorithm itself (not just the analysis of its behavior) depends on the distribution under consideration; the value of β is used in the algorithm's calculations. These reservations aside, however, the algorithm (and its analysis) are quite ingenious. The algorithm works by first finding the solution I^* to the relaxed problem in which one item can have just "part" of itself included. It relies on a theorem that the optimal subset must come from a set X of alternatives "close" to I^* , where $|X|$ is polynomially bounded with high probability, although there is a tradeoff between the polynomial and the probability. Given a desired probability, the algorithm can thus search through X until it has exhausted either X or the corresponding time bound. In the latter case it has failed; in the former it must have the optimal solution.

In this section on positive results, I have concentrated on the *mathematical* analysis of average case behavior, made possible by the assumption of a specific, mathematically defined probability distribution on instances. Note, however, that such distributions can also serve as the starting point for experimental evaluations. By testing algorithms on many instances generated randomly according to these distributions, experimenters may well obtain more precise (although less asymptotic) information about how the algorithms behave on average. Experiments of this type for SATISFIABILITY are reported in [15]. I conclude this section by describing another series of experiments that are relevant to the topic at hand.

The experiments, reported in [5], concerned the familiar problem of BIN PACKING [SR1], viewed as an optimization problem, and the equally familiar algorithm "First Fit Decreasing" (FFD), normally viewed as an approximation algorithm. The bin size was normalized to 1 and, for each n , instances were generated by independently choosing n random (limited precision) numbers from the interval $(0, u]$, where u is a parameter of the distribution and could be fixed at any value from 0 to 1. The experiments revealed, rather surprisingly, that when u was $1/2$ or less, FFD found an optimal solution approximately 75 per cent of the time. The "certificate" of optimality in those cases was the observation that the number of bins used was precisely $B = \left\lceil \sum_{i=1}^n a_i \right\rceil$, where a_i is the size of the i^{th} item, which is an obvious lower bound on the optimal number of bins.

The ratio of $3/4$ was observed to hold for values of n from 125 to 128,000. This of course provides no information about *asymptotic* behavior (although an instance of size 128,000 may well be far more "asymptotic" than any instance

one is likely to encounter in practice). Still, it does suggest the possibility that FFD solves the problem with probability $3/4$ for such distributions and *all* n . Although no such theorem has been proved, there is a theoretical result in [6] that, together with the experiments, lends it credence.

Suppose we reformulate the experimental results so that the quantity measured is the difference between the number of bins used and the sum of the item sizes (whenever this difference is less than 1, the solution is optimal). Then one obtains an average value of about 0.7 (and a range from 0.2 to 1.2), and this holds true for each of the values of n tried. The experiments thus say that no growth in this average is apparent as n increases. The theorem in [6] says that no growth is to be expected; the expected value of the difference is bounded by a constant. Unfortunately, the constant provided by the theorem's proof is slightly larger than that provided by the experiments (it is at least 10,000,000,000), but it is tempting to attribute this difference to the limitations of the proof techniques used rather than to the limitations of the experiments.

3. NP-COMPLETENESS FOR THE AVERAGE CASE

In the previous section we saw that certain NP-complete problems are solvable in polynomial time "on average." In this section we address the natural question raised by such results: Are there NP-complete problems that are *hard* on average? More specifically, is there an NP-complete problem and a probability distribution over that problem's instances such that no algorithm that solves the problem can run in expected polynomial time (or polynomial time almost always, or polynomial time for some fixed fraction of the instances)?

Most research into this question has had the more limited goal of showing that specific approaches to a given <problem,distribution> pair do not perform well on average. Of particular interest is a growing body of results showing that entire *classes* of approaches are ineffectual for certain distributions, in that they require exponential time "almost always." The first such result was due to Chvátal [12] and concerned the optimization version of INDEPENDENT SET [GT20] and the distribution in which all graphs on n vertices and dn edges are equally likely (d fixed and sufficiently large). Two later results of this form concern optimization versions of problems discussed in the previous section.

For the optimization version of GRAPH k -COLORABILITY, where one is trying to find the minimum number of colors necessary (i.e., the chromatic number), McDiarmid [33] proves an almost always exponential time result for a class of branch and bound algorithms and the standard random graph model, with edge probability $p(n)$ fixed, say at $1/2$. Although this is the same distribution as used in the result of Wilf [42] mentioned above, there is no contradiction; the goal of Wilf's algorithm is to verify 3-colorability; it is not concerned with how many colors are needed when 3 do not suffice. For the optimization version of KNAPSACK, Chvátal [13] proves an almost always exponential time

result for a quite general class of algorithms (including the standard branch-and-bound and dynamic programming approaches among others), but for a distribution considerably more restrictive than that used by Goldberg and Marchetti-Spaccamela in their result mentioned above [20].

Results like these are interesting, but cannot be used to prove that a given <problem,distribution> pair is inherently hard on average. For this the class of algorithms covered would somehow have to be extended to include “all” algorithms, at which point we would of course also have proved that $P \neq NP$. Therefore, given the current state of the art, the best one can hope for is to show that a given pair is hard on average if *any* such pair is. L. A. Levin has recently proved just such a result, using a concept he calls “random NP-completeness” [31]. In the remainder of this section I shall describe this concept and the <problem,distribution> pair that is the object of Levin’s result.

It is easiest to describe Levin’s result in terms of the question of solvability in expected polynomial time (although the result has more general consequences). In order to get a true analogue of NP-completeness, however, the definition of “expected polynomial time” will have to be altered somewhat. The definition we have been using says that an algorithm runs in expected polynomial time if there is an integer k such that for all n , if I is the set of instances of size n , $\mu(x)$ is the probability that a random such instance is x , and $T(x)$ is the algorithm’s running time on x , then $\sum_{x \in I} \mu(x)T(x) = O(n^k)$.

This definition has a serious drawback if one wants to build a general complexity theory around it. Unlike the standard notion of (worst case) polynomial time, this notion is not “machine independent.” (To see this, suppose that Machine Model A can simulate an algorithm running in time T on Machine Model B in time T^2 , and consider an algorithm that in model B runs in time n with probability $1 - 2^{-n/2}$ and in time $2^{n/2}$ otherwise.) Fortunately, the solution to this difficulty involves only a slight reorientation (and only a slight weakening of our original notion). Instead of requiring that $\sum_{x \in I} \mu(x)T(x) = O(n^k)$ for some k , we require that, for some $k > 0$, $\sum_{x \in I} \mu(x)T(x)^{1/k} = O(n)$.

If we wish our results to be encoding-independent as well as machine-independent, it is also necessary to dispense with the idea of a separate probability distribution for each instance size n . Thus Levin considers only probability distributions μ that cover *all* instances. (Note that it is easy to adapt the distributions we have been discussing to this format: multiply the probabilities of instances of size n by $1/n^2$ and normalize by dividing through by $\sum_{n=1}^{\infty} 1/n^2 = \pi^2/6$.) Our definition of “expected polynomial time” then reduces to the requirement that, for some $k > 0$, $\sum_{x \in I} \mu(x)T(x)^{1/k} / |x|$ converges, where $|x|$ is the length of x and I is now the set of *all* instances.

Since our domain is now <problem,distribution> pairs rather than just decision problems, we also need a new notion of “polynomial transformation,” one that transforms distributions as well as instances. Here Levin uses the old

notion as far as instances go, but introduces a new concept, “domination,” to help with the distributions. A measure μ_1 is said to *dominate* a measure μ_2 if there is a k such that $\mu_1(x) \geq \mu_2(x)/|x|^k$ for all instances x (the dominating measure can be arbitrarily larger but only polynomially smaller than the dominated one). A function f from the instances of X_1 to the instances of X_2 is a *polynomial transformation* from $\langle X_1, \mu_1 \rangle$ to $\langle X_2, \mu_2 \rangle$ if and only if (a) f maps “yes”-instances to “yes”-instances and “no”-instances to “no”-instances, (b) f is computable in (worst case) polynomial time, and (c) μ_2 dominates the induced probability function $f(\mu_1)$ defined by $f(\mu_1)(x) = \sum_{y \in f^{-1}(x)} \mu_1(y)$.

Given these definitions, it is not difficult to prove that polynomial transformations can be composed, and that if there is an expected polynomial time algorithm for $\langle X_2, \mu_2 \rangle$ and a polynomial transformation to this pair from $\langle X_1, \mu_1 \rangle$, then there is an expected polynomial time algorithm for $\langle X_1, \mu_1 \rangle$.

Now, to complete the analogy, we need to define what it means for a pair $\langle X, \mu \rangle$ to be in what Levin calls “random NP.” Here Levin’s choice may at first seem overly restrictive. X may be any problem in (ordinary) NP, but μ must obey a stringent condition: As in ordinary NP-completeness, we can view all instances as strings over $\{0,1\}$ and hence as distinct positive integers (if we assume an implicit leading 1). The measure μ must be such that its *distribution function* $M(x) = \sum_{i=1}^x \mu(i)$ is computable in (worst case) polynomial time. Note that the polynomial time computability of $M(x)$ implies that of $\mu(x)$, but the implication need not go in the other direction. However, it is not difficult to devise encodings that make $M(x)$ polynomial time computable for each of the distributions we have discussed in this column, and Levin hypothesizes [32] that any “natural” distribution either has a polynomial time computable distribution function $M(x)$, or else is dominated by a distribution that does.

Levin’s main result can now be stated. The following problem is “random NP-complete,” (i.e., every problem in random NP can be polynomially transformed to it) and hence can be solved in expected polynomial time only if all problems in random NP can be so solved.

RANDOM TILING

INSTANCE: Positive integers k and N , $k \leq N$, set T of “tile types,” where a tile type is an oriented square with each corner (vertex) labeled by one of the 26 letters of the alphabet, a sequence t_1, \dots, t_k of “legal” tiles (i.e., ones conforming to tile types in T), where the labels on the upper and lower right corners of t_i are the same as the labels on the upper and lower left corners, respectively, of t_{i+1} , $1 \leq i < k$, and a sequence of $N - k$ 0’s (included to insure that the instance size is at least in N).

QUESTION: Is there a way of filling an $N \times N$ square with N^2 legal tiles such that every vertex shared by two or more tiles has the same label in each, and

such that the sequence of the first k tiles along the bottom row of the square agrees with t_1, \dots, t_k ?

DISTRIBUTION: Instances are generated by randomly choosing N , k , T , and t_1, \dots, t_k , in sequence, according to the following probabilities. The probability that $N = n$ is n^{-2} divided by the appropriate constant. The probability that $k = K$ is $1/N$ for $1 \leq K \leq N$. All subsets of the 26^4 possible tile types are equally likely as choices for T . The type of tile t_1 may be any member of T with equal probability, and for each i , $1 \leq i < k$, t_{i+1} may, with equal probability, be any of the types in T that can legally go to the right of t_i . (If none exist, then the instance is automatically replaced by one whose answer is known to be “no”.)

Comment. The basic tiling problem was proved NP-complete by Levin in his original NP-completeness paper [30]. That construction is used implicitly in the current proof, which Levin, in his usual style, presents in a single, exceedingly concise paragraph [31]. The real surprise here is that a problem with such a smooth and uniform distribution can possibly be the transformed image of everything in random NP, which includes distributions that are the opposite of uniform (say, ones with all the probability concentrated in a small collection of instances). The trick is to map instances with more than their fair share of probability (assuming that all instances of size n should be equally likely and hence should have probability proportional to $1/(n^2 2^n)$) to instances that are *shorter* and hence have a higher allocation of probability. It is here that the polynomial time computability of the distribution function comes in. If x has high probability, then the interval from $M(x-1)$ to $M(x)$, whose length is $\mu(x)$, must be long enough to contain a fraction $q(x)$ with a short denominator, and this can be used as a short signature for x in the rest of the construction. (Given $q(x)$, x can be reconstructed in time polynomial in $|x|$ by binary search, using a polynomial time subroutine for M .) Eventually, after liberal doses of domination and a detour through a universal Turing machine, we end up with a tiling instance in which N , the length of the square to be tiled, is a polynomial function of $|x|$, and t_1, \dots, t_k , the pre-specified sequence of tiles, is a disguise for $q(x)$. Cryptanalytically inclined readers are referred to [31] for further details. \square

RANDOM TILING is probably not the only “random NP-complete” problem. Indeed, relying on the same sort of arguments that are used to support the claim that all “known” NP-complete problems are polynomial time isomorphic [7], one can plausibly argue that each such problem X has associated with it at least one distribution μ such that $\langle X, \mu \rangle$ is random NP-complete [32]. Such distributions, however, may be exceedingly baroque. For a challenging “Open Problem of the Month,” readers are invited to search for additional “natural” random NP-complete pairs $\langle X, \mu \rangle$. I am eager to report any discoveries.

REFERENCES

1. I. ADLER AND N. MEGIDDO, A simplex algorithm whose average number of steps is bounded between two quadratic functions of the smaller dimension, in "Proceedings 16th Ann. ACM Symp. on Theory of Computing," Association for Computing Machinery, New York, 1984.
2. D. ANGLUIN AND L. G. VALIANT, Fast probabilistic algorithms for Hamiltonian circuits and matchings, in "Proceedings 9th Ann. ACM Symp. on Theory of Computing," pp. 30-41, Association for Computing Machinery, New York, 1977.
3. L. BABAI AND L. KUCERA, Canonical labelling of graphs in linear time, in "Proceedings 20th Ann. Symp. on Foundations of Computer Science," pp. 39-46, IEEE Computer Society, Los Angeles, 1979.
4. E. A. BENDER AND H. S. WILF, A theoretical analysis of backtracking in the graph coloring problem, *J. Algorithms*, to appear.
5. J. L. BENTLEY, D. S. JOHNSON, F. T. LEIGHTON, AND C. C. MCGEOCH, An experimental study of bin packing, in "Proceedings 21st Ann. Allerton Conf. on Communication, Control, and Computing," pp. 51-60, Department of Electrical Engineering and the Coordinated Science Laboratory, University of Illinois, Urbana, Ill., 1983.
6. J. L. BENTLEY, D. S. JOHNSON, F. T. LEIGHTON, C. C. MCGEOCH, AND L. A. MCGEOCH, Some unexpected expected behavior results for bin packing, in "Proceedings 16th Ann. ACM Symp. on Theory of Computing," Association for Computing Machinery, New York, 1984.
7. L. BERMAN AND J. HARTMANIS, On isomorphism and density of NP and other complete sets, *SIAM J. Comput.* **6** (1977), 305-322.
8. M. R. BEST, A. E. BROUWER, F. J. MACWILLIAMS, A. M. ODLYZKO, AND N. J. A. SLOANE, Bounds for binary codes of length less than 25, *IEEE Trans. Inform. Theory* **IT-24** (1978), 81-93.
9. K.-H. BORGWARDT, The average number of steps required by the simplex method is polynomial, *Zeitschrift für Oper. Res.* **26** (1982), 157-177.
10. E. F. BRICKELL, Are most low density knapsacks solvable in polynomial time?, *Congressus Numerantium* **39** (1983), 145-156.
11. V. CHVÁTAL, On Hamilton's ideals, *J. Combinatorial Theory Ser. B* **12** (1972), 163-168.
12. V. CHVÁTAL, Determining the stability number of a graph, *SIAM J. Comput.* **6** (1977), 643-662.
13. V. CHVÁTAL, Hard knapsack problems, *Operations Res.* **28** (1980), 1402-1411.
14. M. DAVIS, G. LOGEMANN, AND D. LOVELAND, A machine program for theorem proving, *Comm. ACM* **5** (1962), 394-397.
15. A. K. DEWDNEY, T. R. S. WALSH, AND D. L. WEHLAU, Average-time testing of satisfiability algorithms, *Congressus Numerantium* **39** (1983), 305-325.
16. P. ERDÖS AND A. RÉNYI, On random graphs I, *Publicationes Mathematicae* **6** (1959), 290-297.
17. J. FRANCO AND M. PAULL, Probabilistic analysis of the Davis Putnam procedure for solving the satisfiability problem, *Disc. Applied Math.* **5** (1983), 77-87.
18. A. GOLDBERG, "On the complexity of the satisfiability problem," Courant Computer Science Report No. 16, New York University, New York, 1979.
19. A. GOLDBERG, P. PURDOM, AND C. BROWN, Average time analysis of simplified Davis-Putnam procedures, *Inform. Process. Lett.* **15** (1982), 72-75. Corrigendum, *Inform. Process. Lett.* **16** (1982), 213.
20. A. V. GOLDBERG AND A. MARCHETTI-SPACCAMELA, On finding the exact solution of a zero-one knapsack problem, in "Proceedings 16th Ann. ACM Symp. on Theory of Computing," Association for Computing Machinery, New York, 1984.
21. M. K. GOLDBERG, A nonfactorial algorithm for testing isomorphism of two graphs, *Disc. Applied Math.*, to appear.

22. M. HAIMOVICH, The simplex algorithm is very good! — On the expected number of pivot steps and related properties of random linear programs, manuscript (1983).
23. M. HELD AND R. M. KARP, A dynamic programming approach to sequencing problems, *SIAM J.* **10** (1962), 196-210.
24. N. KARMAKAR, A new polynomial-time algorithm for linear programming, in "Proceedings 16th Ann. ACM Symp. on Theory of Computing," Association for Computing Machinery, New York, 1984.
25. R. M. KARP, The probabilistic analysis of some combinatorial search algorithms, in "Algorithms and Complexity: New Directions and Recent Results" (J. F. Traub, Ed.), pp. 1-19, Academic Press, New York, 1976.
26. J. C. LAGARIAS, Knapsack-type public key cryptosystems and Diophantine approximation, in "Advances in Cryptography: Proc. of CRYPTO-83 (D. Chaum, ed.)," pp. 3-24, Plenum Press, New York, 1984.
27. J. C. LAGARIAS AND A. M. ODLYZKO, Solving low-density subset sum problems, in "Proceedings 24th Ann. Symp. on Foundations of Computer Science," pp. 1-10, IEEE Computer Society, Los Angeles, 1983.
28. E. L. LAWLER, A note on the complexity of the chromatic number problem, *Inform. Process. Lett.* **5** (1976), 66-67.
29. A. J. LENSTRA, H. W. LENSTRA, AND L. LOVÁSZ, Factoring polynomials with rational coefficients, *Math. Annalen* **261** (1982), 515-534.
30. L. A. LEVIN, Universal sorting problems, *Problemy Peredaci Informacii* **9** (1973), 115-116 (in Russian). English translation in *Problems of Information Transmission* **9** (1973), 265-266.
31. L. A. LEVIN, Problems, complete in "average" instance, in "Proceedings 16th Ann. ACM Symp. on Theory of Computing," Association for Computing Machinery, New York, 1984.
32. L. A. LEVIN, private communication (1984).
33. C. J. H. MCDIARMID, Determining the chromatic number of a graph, *SIAM J. Comput.* **8** (1979), 1-14.
34. R. MERKLE AND M. HELLMAN, Hiding information and signatures in trapdoor knapsacks, *IEEE Trans. Inform. Theory* **IT-24** (1978), 525-530.
35. V. PICHAI, M. E. SEZER, AND D. D. SILJAK, A graph-theoretic algorithm for hierarchical decomposition of dynamic systems with applications to estimation and control, *IEEE Transactions SMC-13* (1983), 197-207.
36. V. PICHAI, M. E. SEZER, AND D. D. SILJAK, Author's reply on "Input-output decomposition of dynamic systems is NP-complete", *IEEE Trans. Automat. Contr.*, to appear.
37. L. PÓSA, Hamiltonian circuits in random graphs, *Discrete Math.* **14** (1976), 359-364.
38. A. SHAMIR, A polynomial algorithm for breaking the basic Merkle-Hellman cryptosystem, in "Proceedings 23rd Ann. Symp. on Foundations of Computer Science," pp. 145-152, IEEE Computer Society, Los Angeles, 1982.
39. S. SMALE, On the average number of steps of the simplex method of linear programming, *Math. Programming* **27** (1983), 241-262.
40. R. E. TARJAN, Input-output decomposition of dynamic systems is NP-complete, *IEEE Trans. Automat. Contr.*, to appear.
41. M. J. TODD, "Polynomial expected behavior of a pivoting algorithm for linear complementarity and linear programming problems," Report No. 595, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, New York, 1983.
42. H. S. WILF, Backtrack: An $O(1)$ expected time graph coloring algorithm, *Inform. Process. Lett.*, to appear.