Discrete Applied Mathematics 5 (1983) 133–145
North-Holland Publishing Company

RANDOMISED ALGORITHMS

D.J.A. WELSH

Merton College, Oxford, OX1 4JD England

Received 23 December 1981
Revised 30 April 1982

This is an account of some recent developments in the theory of probabilistic or randomised algorithms and their applications in combinatorial optimisation theory.

1. Introduction

An algorithm which has an error probability of less than 2^{-100} and which within a minute identifies $2^{400}-593$ as the largest prime below 2^{400} has an immediate practical and aesthetic appeal.

This primality testing algorithm of Rabin [23] together with a similar one of Solovay and Strassen [30] is the most well-known example of a probabilistic algorithm. Its startling success has had a major impact on the theory of algorithms.

First an example to illustrate the basic idea. Suppose we have a polynomial expression in n variables, say $f(x_1, \ldots, x_n)$ and we wish to check whether or not f is identically zero. To do this analytically could be a horrendous computation. Suppose instead we generate a random n-vector $(r_1, \ldots, r_n)$ and evaluate $f(r_1, \ldots, r_n)$. If $f(r_1, \ldots, r_n) \neq 0$, we know $f \neq 0$; if $f(r_1, \ldots, r_n) = 0$, then either f is identically zero or we have been extremely lucky in our choice of $(r_1, \ldots, r_n)$. Do this several times and if we keep on getting $f = 0$ we conclude f is identically zero. The probability we will have made an error is negligible.

This example illustrates the basic idea behind randomised algorithms. They are not to be confused with probabilistic approximation algorithms typified by the paper of Karp [19]; they tend to be much more powerful results and correspondingly much rarer. Indeed, as we shall see, the existence of a randomised algorithm for a problem seems to correspond in more than one sense with the problem being tractable. For as Adelman and Manders [2] remark, if we can find a fast algorithm for a problem which is correct with probability $> 1 - 2^{-k}$ for some integer k independent of the size of the problem, by making k large enough we can reduce the probability of error far below the probability that a hardware failure causes an incorrect answer.

2. Probabilistic algorithms; some examples

We shall give a more precise definition of terms such as 'randomly decidable' and 'probabilistic algorithm' in later sections. For the moment it is enough to proceed more loosely so that the flavour of a random algorithm can be assimilated by means of different examples.

If π is a computational problem and x is the input or instance of π under consideration a *probabilistic* or *randomised algorithm* for solving π proceeds as follows. At certain junctures in the execution of the program for solving the instance x of π the algorithm makes a random decision. With the exception of these random choices the algorithm proceeds purely deterministically.

For some instances a probabilistic algorithm may sometimes produce an incorrect solution. We shall insist that any probabilistic algorithm has the property that the probability of error can be made arbitrarily small.

Most probabilistic algorithms which we shall consider always terminate. However there do exist in the literature (see for example [18]) examples of algorithms which do not always terminate but do so only with probability one. Such algorithms may still have a better performance with respect to some measure of time or space than any known deterministic ones for the same task. We return to this topic in Section 4. First, however, some more detailed examples.

2.1. Primality

Solovay and Strassen [30] and Rabin [23] give different probabilistic algorithms which test a given integer n for primality. In both algorithms the basic idea is the same and consists of taking k integers i uniformly distributed between 1 and $n-1$ and then for each i checking whether a predicate $W(i,n)$ holds.

The exact form of $W(i,n)$ differs in [23] and [30] but in both cases $W(i,n)$ has the properties:

(a) if n is prime, then $W(i,n)$ is false for all i;

(b) if n is composite, then for at least a half of the integers i in the range chosen $W(i,n)$ is true;

(c) $W(i,n)$ can be computed quickly, that is has a complexity which is bounded in the size of input of n, which in this case is $\log n$.

The difficulty is in finding predicates $W(i,n)$ satisfying these conditions. For example, the $W(i,n)$ used by Rabin depend on the following two theorems.

Theorem 2.1. *If n is an integer and $1<i<n$, then if either of conditions* (i) *and* (ii) *hold the integer n is composite.*

(i) $i^{n-1} \neq 1 \bmod n$.

(ii) *There exists an integer k such that $n-1$ is divisible by 2^k giving an integer m such that*

$$1<(i^m-1,n)<n.$$

Theorem 2.2. *If n is composite at least half of the integers i in the range $1<i<n$ satisfy* (i) *or* (ii). *If n is prime, then neither is satisfied for any i.*

Thus in Rabin's method the predicate $W(i,n)$ is taken to be that either (i) or (ii) holds and when a pair i,n satisfying (i) or (ii) are found we call i a *witness* to the compositeness of n.

2.2. *Checking polynomial identities*

Suppose that we are given a purported polynomial identity of the form $Q(x_1,\ldots,x_n)\equiv 0$, where for simplicity we assume the coefficients in Q are integers.

For example, if we did not know Vendermonde's identity we might need to check the truth of

$$Q\equiv \begin{vmatrix} 1 & x_1 & \cdots & x_1^{n-1} \\ \vdots & & & \vdots \\ 1 & x_n & \cdots & x_n^{n-1} \end{vmatrix} - \sum_{i<j}(x_i-x_j)=0$$

for some fixed value of n, say $n=100$. In this case Q is a polynomial of total degree approximately 5000 and thus the deterministic method of simplifying by direct expansion is hopeless.

The probabilistic algorithm which can be used is very elementary. It makes use of the following lemma.

Lemma 2.3. *Let $Q=Q(x_1,\ldots,x_n)$ be a polynomial in the variables $x_1,\ldots,x_n$ and suppose that Q is not identically zero. If I is any set of elements in the field of coefficients of Q such that $|I|\geq c\deg Q$, then the number of elements of $I\times\cdots\times I$ which are zeros of Q is at most $|I|^n/c$.*

Hence the following probabilistic algorithm enables us to check an identity of the form $Q\equiv 0$.

(1) Choose I so that $|I|>c\deg Q$ with $c>2$.

(2) Choose an integer N such that c^{-N} is small enough for our purposes (say N such that $c^{-N}<2^{-400}$).

(3) Select N variables $(y_1,\ldots,y_n)$ at random from $(I\times\cdots\times I)$. If any one of these y is not a zero of Q, then $Q\not\equiv 0$. If all are zeros of Q, then by the above lemma the probability that we make an error in asserting that $Q\equiv 0$ is very small.

Using a slight extension of this idea which enables the probabilistic tests to be carried out in modular arithmetic, Schwartz [27] has developed an algorithm which can be carried out at almost full arithmetic speed and not needing high precision arithmetic. For example in the example above, Vandermonde's identity with $n=100$ can be checked to a probabilistic accuracy of 10^{-100} in only 60 random tests which it is claimed could be carried out on a reasonably fast computer in about 3 minutes.

For details of these and also probabilistic methods for testing other types of poly-

nomial identities and properties of systems of polynomials, see Schwartz [27], and also Zippel [35].

2.3. *Matrix multiplication*

Consider the problem of deciding for three $n \times n$ matrices A, B, C whether $A \cdot B = C$.

Frievalds [11] gives the following very simple probabilistic algorithm for this problem which has complexity $O(n^2)$ whereas no deterministic algorithm of this order of magnitude is known; see for example Strassen [31] or Pan [22].

Algorithm 2.4. Generate k random $1 \times n$ vectors x with entries from $\{-1, 1\}$, use each to test whether $(A \cdot (Bx)) = C \cdot x$. If at least one test fails then $A \cdot B \neq C$. If the matrices pass all k tests, then it is assumed $A \cdot B = C$.

It is easy to prove that the probability of error by using this algorithm is not more than 2^{-k}, and its complexity is $O(n^2)$.

2.4. *Polynomial multiplication*

Similarly given polynomials $p_1(x)$, $p_2(x)$, $p_3(x)$, if we wish to test whether $p_1(x)p_2(x) = p_3(x)$ we proceed as follows.

Algorithm 2.5. Generate k independent random integers r in the set $A_N = [-N+1, N]$. For each such integer r test the numerical identity $p_1(r)p_2(r) = p_3(r)$. If n is the maximum of the degrees of p_1, p_2, we know that $p_1 p_2 - p_3$ has not more than $2n$ zeros in the set A_N. Hence if we assume the truth of the statement $p_1 p_2 = p_3$ when it holds for each of our k random choices the probability of error is not more than $(n/N)^k$ and hence we can make our probability of error less than ε with a probabilistic algorithm of complexity $O(n)$.

Both this and the preceding algorithm for matrix multiplication suffer the grave disadvantage to the practitioner that they 'demand a knowledge of the answer'. In other words they are *checking* algorithms rather than genuine calculations. This suggests the following question.

Problem 2.6. Is it easier to check deterministically that $A \cdot B = C$ than to calculate the matrix product $A \cdot B$?

2.5. *Perfect matchings in graphs*

Consider the problem of deciding whether a graph G on n vertices has a perfect matching. The algorithm of Edmonds [10] is certainly polynomially bounded, being

of complexity $O(n^3)$. However it is a pretty complicated algorithm and for many programmers its implementation would take exponential time!

The following probabilistic algorithm suggested by Lovász [20] overcomes this demerit and has a certain aesthetic simplicity. It depends on the following theorem.

Theorem 2.7. *Let G be a simple graph. Orient its edges arbitrarily. For each edge e of G let x_e be an indeterminate. Form the matrix $B = (B_{ij})$ where*

$$B_{ij} = \begin{cases} x_e & \text{if } e = (i, j), \\ -x_e & \text{if } e = (j, i), \\ 0 & \text{otherwise.} \end{cases}$$

Then G has a perfect matching iff det B *is not identically zero in the variables x_e.*

But now note that det B is just a polynomial in the variables x_e and hence we can just test whether it is zero by the probabilistic method described earlier.

The only drawback to the above method is that it does not give us a method of actually finding a perfect matching when one exists. In fact this can be done probabilistically but it is a little more complicated. We refer to Lovász [20] for details.

Apart from these probabilistic methods which we have treated in some detail we should also draw attention to the existence of randomised algorithms which improve the time complexity for the following problems.

(1) Finding a nearest pair of points in a collection of points in $\mathbb{R}^k$ (Rabin [23]).

(2) Finding an irreducible polynomial of degree n over a finite field, finding roots of a polynomial and factoring a polynomial into its irreducible factors over a finite field (Rabin [24] but see also Berlekamp [8]). Each of these problems is of considerable importance in algebraic coding theory and for large finite fields these algorithms make an intractable problem feasible.

(3) Finding a set of circuits $\{C_i\}$ of a graph G on n vertices which cover the edges of G and which minimise the quantity

$$c = \sum |C_i|.$$

This is a problem occurring in irrigation theory and electric network theory. Deterministic algorithms are of order $O(n^3)$ but Itai and Rodeh [18] give a probabilistic algorithm which finds a cover with $c \leq e + 2n \log n$ where e is the number of edges and which has expected running time $O(n^2)$.

(4) Deciding whether a polynomial $f \in k[x_1, \ldots, x_n]$ is prime over $k'[x_1, \ldots, x_n]$ where k' is an extension field of k (Heintz and Sieveking [16]).

In addition we shall mention in Section 4 probabilistic algorithms which are more economical with respect to space requirements, see [3].

3. Random polynomial time

Motivated by the success of the Rabin–Solovay–Strassen algorithm for recognising primes it is natural to study the class of languages which, like primes, are randomly decidable. This notion is made precise in this section. Our complexity terminology is essentially that of Garey and Johnson [13] and Hopcroft and Ullman [17], and we assume familiarity with concepts such as NP, P-space, etc., all of which can be found in these references.

A language L is *randomly decidable* if there exists a non-deterministic Turing machine M and a polynomial p such that:

(i) If $x \notin L$ no computation path of machine M on input x will halt.

(ii) If $x \in L$ at least half of the computation paths of length $p(|x|)$ of machine M on input x will halt.

We denote the class of randomly decidable languages by RP.

Another way of looking at RP is as follows.

A language L is in RP if there exists a polynomial f and a polynomial algorithm which computes for each input x and each possible certificate (= guess) y of length $f(|x|)$ a value $v(x, y) \in \{0, 1\}$ such that

(i)′ If $x \notin L$, $v(x, y) = 0 \; \forall y$.

(ii)′ If $x \in L$, $v(x, y) = 1$ for at least half of all possible certificates y.

There are various points to note. First, since the class NP could be defined by replacing 'at least half' in (ii) and (ii)′ by 'at least one' we know

$$\mathrm{RP} \subseteq \mathrm{NP},$$

and obviously

$$\mathrm{P} \subseteq \mathrm{RP}.$$

Secondly, the role of the fraction 'one half' in (ii), is unimportant and it could be replaced by any fixed fraction $1/k$, independent of the size of input.

The importance of showing that a language L belongs to RP is that we can then recognise any string x as a member of L by the following probabilistic procedure: (a) generate a random sequence y of length $f(|x|)$; (b) calculate $v(x, y)$. Repeat this procedure k times and return the answer $x \in L$ iff $v(x, y) = 1$ for at least one of the k random y generated. Then, exactly as in the special case of the language of primes, the probability of error is not more than $1/2^k$.

Thus once we have shown that a language belongs to RP we can essentially recognise it; unfortunately it seems exceptionally difficult to find randomised algorithms for problems for which no polynomial algorithm is known. The reasons for this are twofold. First it is easy to prove the following.

RP = NP if and only if some NP-complete language is randomly decidable.

Hence, since it does seem unlikely that RP = NP we are most unlikely to find a randomised algorithm for any problem which is NP-complete.

Accordingly, candidates for membership of RP \ P (if indeed this is non-empty),

are members of NP \ P which are not yet known to be NP-complete. Of the well-known candidates for this class mentioned in Garey and Johnson [13, p. 155] only ISOMORPHISM remains unsettled and in view of the recent theorem of Luks [21] there seems a reasonable possibility that this problem has in fact a polynomial algorithm.

A key problem in this area therefore is the following.

Problem 3.1. Find examples of languages in RP which do not appear to be in P.

We close this section by drawing attention to a recent result of Adleman [1] which gives added justification to our claim that membership of RP corresponds to practical tractability.

If L is a language let $\hat{L}_n$ be its indicator function defined by

$$\hat{L}_n(x) = \begin{cases} 1 & x \in L, \ |x| = n, \\ 0 & \text{otherwise.} \end{cases}$$

Write $c(L_n)$ for the minimum number of gates (Boolean operations) in a Boolean circuit which realises $\hat{L}_n$; c is known as the *circuit size* of the sequence of functions $(L_n : n = 1, 2, \ldots)$.

We say L has *polynomial size circuits* if there exists a polynomial p such that for $n = 1, 2, \ldots, c(L_n) < p(n)$.

It is easy to see that if L is recognisable in polynomial time then it has polynomial circuits but the converse does not hold, if only because it is possible to compute non-recursive functions using polynomial size circuits. Despite this we can use this 'more concrete' circuit complexity to try and prove P $\neq$ NP by showing that some NP-complete problem does not have polynomial size circuits.

Such a result seems a long way off, and broadly speaking 'polynomial size circuits' also corresponds to our notion of tractable. We next state Adleman's theorem.

Theorem 3.2 (Adleman). *If a language is randomly decidable it has polynomial size circuits.*

The proof of this intriguing result is by a refreshingly simple counting argument.

Proof (Sketch). Let language $L \in$ RP. There is a polynomial $t(n)$ say such that there are at most $2^{t(n)}$ possible witnesses (certificates) to the membership of any input of size n in L. Taking $m = 2^{t(n)}$ let $(w_j : 1 \leq j \leq m)$ be these possible witnesses. Form the incidence matrix (A_{ij}) in which the rows correspond to the possible witnesses w_j. Let A_{ij} be 1 or 0 according as the w_j is in fact a witness to the membership of x_i in L.

By hypothesis at least half the entries in A are non-zero. Hence there must exist some column, say, the first, with at least half its entries ones. Remove the corresponding witness w_1, together with all rows corresponding to inputs for which w_1 is a

reliable witness. Repeat the procedure, since this revised matrix has the same property. In this way we find $v(\leq n)$ witnesses at least one of which is a witness to each possible input. Use these to build your circuit.

4. Random log-space

In exactly the analogous way as we defined RP, the class of languages recognisable by probabilistic algorithms operating in polynomial time, we define *random log-space* RL to be the class of languages recognisable by probabilistic algorithms operating in log-space.

We know from [13], [17] that

$$L \subseteq NL \subseteq P \subseteq NP \subseteq P\text{-SPACE} \tag{1}$$

where L and NL are log-space and non-deterministic log-space respectively. It is also known that equality cannot hold throughout (1) since $L \neq P\text{-SPACE}$.

Exactly analogously to the case with random polynomial time we have

$$L \subseteq RL \subseteq NL. \tag{2}$$

Again it is not known whether each of the inclusions in (2) is strict.

It should be mentioned here that if we allowed algorithms which did not always stop but which only had probability zero of not stopping then we could in space $\log n$ count probabilistically up to 2^n and if we allowed such algorithms we would in fact have $RL = NL$. What we here call RL is the class denoted by R^{poly} in [3].

Note. In order to count up to 2^{2^n} in space n we just wait until 2^n ones occur in a random stream of 0's and 1's - see Gill [15]. There is the slight problem here that we are really 'only approximately counting' 2^{2^n} (since there is a slight probability that we will count to a much larger or smaller number).

Returning to the problems of membership or non-membership of RL, probably the most interesting problem is whether or not $RL = NL$. This would be settled in the affirmative if the following problem could be solved.

Problem 4.1. Find a randomised (terminating) algorithm which uses only log-space for deciding whether a digraph has a directed path linking two specified vertices.

This is because this digraph problem - known as DIRECTED REACHABILITY is complete in nondeterministic log-space.

Examples of languages which can be recognised in log-space by randomised algorithms but which (at the moment at least) are not known to have log-space deterministic recognition algorithms are the class of non-bipartite graphs and the class of connected graphs.

The idea behind both these randomised algorithms is pretty much the same. We describe the algorithm for the class of connected graphs. Move randomly through a

graph of n vertices for about n^2 steps. There is a high probability that two specified vertices will be linked in this number of steps. Repeat until the threshold is reached and repeat for each pair of vertices of the graph. For details see [3].

Natural questions to ask about RL are:

(1) Do there exist complete languages for RL?

(2) Defining co-RL to be the complementary languages of the members of RL is the inclusion

$$\mathrm{L} \subseteq \text{co-RL} \cap \mathrm{RL}$$

strict?

5. The probabilistic hierarchy

From our earlier considerations we know that there is a total order on the classes of languages given by

$$\mathrm{L} \subseteq \mathrm{RL} \subseteq \mathrm{NL} \subseteq \mathrm{P} \subseteq \mathrm{RP} \subseteq \mathrm{NP} \subseteq \text{P-SPACE},$$

and the cut-off point as regards tractability would seem to be at the class RP of randomly decidable sets.

In this section we describe the work of Gill [14, 15], Simon [29] and Valiant [33] and its relation to the upper echelons of this hierarchy.

The most convenient way of doing this seems to be through the probabilistic Turing machine model. These machines (which in contrast to the perhaps more well known nondeterministic machines do exist in the constructive and practical sense), were first introduced by De Leeuw, Moore, Shannon and Shapiro [9].

A *probabilistic Turing machine* (PTM) is a Turing machine with distinguished states called coin-tossing states. For each coin-tossing state, the finite control unit specifies two possible next states. The computation of such a machine is now purely deterministic except that when in a coin-tossing state the machine chooses with equal probability between two possible next states.

In general a PTM computes a random string; for each input x the machine M will produce a random output string y with probability $P[M(x)=y]$. We define the partial function computed by M by

$$\phi_M(x) = \phi(x) = \begin{cases} y & \text{if } P[M(x)=y] > \frac{1}{2}, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

The *error probability* of M is the function e_M defined by

$$e(x) = e_M(x) = \begin{cases} P[M(x) \neq \phi(x)] & \text{if } \phi_M(x) \text{ is defined,} \\ \text{undefined} & \text{if } \phi_M(x) \text{ is undefined.} \end{cases}$$

Thus, whenever it is defined, $e(x) < \frac{1}{2}$.

If x is an input to M for which the output $\phi_M(x)$ is defined the *running time* of M

on input x is defined to be

$$t_M(x) = \text{smallest } n \text{ such that } P[M(x) = \phi(x) \text{ in time } n] \text{ is strictly greater than } \tfrac{1}{2}.$$

Otherwise $t_M(x)$ is undefined.

The machine M is *polynomially bounded* if there is a polynomial $p(n)$ such that every possible computation of M on inputs of length n halts in at most $p(n)$ steps.

Compare now the two classes of languages introduced by Gill [15].

Definition 5.1. (i) PP is the class of languages recognised by polynomially bounded probabilistic Turing machines.

(ii) BPP is the class of languages recognised by polynomially bounded probabilistic Turing machines with error probability bounded by a constant less than $\frac{1}{2}$.

Since the error probability of any probabilistic machine is less than $\frac{1}{2}$ the difference between PP and BPP appears marginal. Remarkably, this subtle difference in definition seems to be of crucial importance in terms of practically solving a problem for the following reasons.

If a language $L \in \text{BPP}$ there exists a PTM say M and ε, $0 < \varepsilon < \frac{1}{2}$, such that M accepts L with error probability $\leq \varepsilon$. Now for any given input x, suppose we repeatedly input x to the machine M, say an odd integer N number of times and take the majority decision on whether or not to accept x as a member of L, then the probability of error is (uniformly over all inputs) reduced to below

$$\sum_{k=0}^{(N-1)/2} \binom{N}{k} \varepsilon^{N-k}(1-\varepsilon)^k. \tag{3}$$

This sum approaches zero at an exponential rate, and thus if $L \in \text{BPP}$, the decision problem for membership of L is essentially solved to whatever degree of accuracy we wish.

However if all we know about L is that it belongs to PP, then it could well be that the error probability $e(x)$ was of the form

$$e(x) = \tfrac{1}{2} - (\tfrac{1}{2})^{|x|}.$$

The corresponding sum to (3) is no longer convergent to zero. Indeed as Gill [15] shows

$$\text{NP} \subseteq \text{PP} \tag{4}$$

and as Simon [29] shows; essentially PP is the class of decision problems associated with the class of counting problems which Valiant [33] calls #P. A typical 'hardest' or 'complete' member of PP is a problem of the following type.

For a given graph G and integer k does G have at least k distinct Hamiltonian cycles?

Thus from the practical point of view the gap between BPP and PP is enormous.

Unlike RP, it is easy to prove that both BPP and PP are closed under complementation and, almost by definition we know

$$\text{RP} \cup \text{co-RP} \subseteq \text{BPP}. \tag{5}$$

The exact relationship between BPP, NP and co-NP is not known.

Since PP turns out to be essentially the class of counting problems #P, the decision problems obtained from the #P-complete problems of Valiant [33] are complete for the class PP. However we know of no 'hardest' problems for the class BPP and indeed it does not seem clear that such problems exist.

We close with the following diagram illustrating what is known about the probabilistic hierarchy.

$$\text{P} \subseteq \text{RP} \cap \text{co-RP} \subseteq \text{RP} \begin{matrix} \subsetneq \\ \subsetneqq \end{matrix} \begin{matrix} \text{BPP} \\ \text{NP} \end{matrix} \begin{matrix} \subsetneqq \\ \subsetneq \end{matrix} \text{PP} \subseteq \text{P-SPACE}. \tag{6}$$

A not unreasonable conjecture is that all inclusions in (6) are in fact strict.

6. Conclusion

From the practical viewpoint a strong case can be made for searching for randomised algorithms which improve the running time of a problem. However it is worth emphasising that in very few cases has it been proved that randomised algorithms are better than the 'best possible deterministic' algorithm. One such case is a result of Freivalds who showed that on 1 a tape Turing machine palindromes can be recognised in time $n \log n$ with probability $1-\varepsilon$ for arbitrary $\varepsilon > 0$ while any deterministic recognition requires at least kn^2 time, for some constant k; see also Gill's remarks on this problem [15, p. 682–685].

For more recent discussions of the different capabilities of probabilistic and deterministic finite state machines we refer to a very interesting recent paper by Freivalds [12].

Finally we close by drawing attention to a recent paper of Bennett and Gill [7] which although not directly concerned with randomised algorithms contains a host of interesting results and problems on the related subject of random oracles.

Acknowledgement

It is a pleasure to acknowledge the helpful comments and conversations which I have had on this subject with Anthony Mansfield, Colin McDiarmid and Leslie Valiant.

References

[1] L. Adleman, Two theorems on random polynomial time, Proc. 19th IEEE symp. Found. Comput. Sci., Ann. Arbor., MI (1978) 75–83.

[2] L. Adleman and K. Manders, Reducibility, randomness and intractability, Proc. 9th ACM Symp. Theory Comput. (1977) 151–153.

[3] R. Aleliunas, R.M. Karp, R.J. Lipton, L. Lovász and C. Rackoff, Random walks, traversal sequences and the complexity of maze problems, Proc. 20th IEEE Symp. Found. Comput. Sci. (1979) 218–223.

[4] D. Angluin and L.G. Valiant, Fast probabilistic algorithms for hamiltonian circuits and matchings, J. Comput. System Sci. (1979) 155–193.

[5] L. Babai, P. Erdös and S.M. Selkow, Random graph isomorphism, SIAM J. Comput. 9 (1980) 628–635.

[6] L. Babai, Monte-Carlo algorithms in graph isomorphism testing, SIAM J. Comput., to appear.

[7] C.H. Bennett and J. Gill, Relative to a random oracle A, $P^A \neq NP^A \neq \omega\text{-}NP^A$ with probability 1, SIAM J. Comput. 10 (1981) 96–113.

[8] E.R. Berlekamp, Factoring polynomials over large finite fields, Math. Comput. 24 (1970) 713–735.

[9] K. De Leeuw, E.F. Moore, C.E. Shannon and N. Shapiro, Computability by probabilistic machines, Automata Studies, Ann. Math. Stud. 34 (1956) 183–212.

[10] J.R. Edmonds, Paths, trees and flowers, Canad. J. Math. 17 (1965) 449–467.

[11] R. Freivalds, Fast probabilistic algorithms, Springer Lecture Notes in Computer Science 74 (1979) 57–69.

[12] R. Freivalds, Probabilistic two-way machines, Springer Lecture Notes in Computer Science 118 (1981) 33–45.

[13] M.R. Garey and D.S. Johnson, Computers and Intractability (Freeman, San Francisco, CA, 1979).

[14] J.T. Gill III, Computational complexity of probabilistic Turing machines, Proc. 6th ACM Symp. Theory Comput. (1974) 91–95.

[15] J.T. Gill III, Computational complexity of probabilistic Turing machines, SIAM J. Comput. 6 (1977) 675–694.

[16] J. Heintz and M. Sieveking, Absolute primality of polynomials is decidable in random polynomial time in the number of variables, Springer Lecture Notes in Computer Science 115 (1981) 16–28.

[17] J.E. Hopcroft and J.D. Ullman, Introduction to Automata Theory, Languages, and Computation (Addison-Wesley, Reading, MA, 1979).

[18] A. Itai and M. Rodeh, Covering a graph by circuits, Springer Lecture Notes in Computer Science 62 (1978) 289–299.

[19] R.M. Karp, The probabilistic analysis of some combinatorial search algorithms, in J.F. Traub, ed., Algorithms and Complexity: New Directions and Recent Results (Academic Press, New York, 1976) 1–19.

[20] L. Lovász, On determinants, matchings and random algorithms, to appear.

[21] E. Luks, Isomorphism of graphs of bounded valence can be tested in polynomial time, Proc. 21st FOCS (1980) 1–8.

[22] V. Ya Pan, New fast algorithms for matrix operations, SIAM J. Comput. 9 (1980) 321–342.

[23] M.O. Rabin, Probabilistic algorithms, in: J.F. Traub, ed., Algorithms and Complexity (Academic Press, New York, 1976) 21–39.

[24] M.O. Rabin, Probabilistic algorithms in finite fields, SIAM J. Comput. 9 (1980) 273–280.

[25] C. Rackoff, Relativised questions involving probabilistic algorithms, Proc 10th ACM Symp. Theory of Comput. (1978) 338–342.

[26] J.E. Savage, The Complexity of Computing (Wiley, New York, 1976).

[27] J.T. Schwartz, Probabilistic algorithms for verification of polynomial identities, Symbolic and Algebraic Computation, Springer Lecture Notes in Computer Science 72 (1979) 200–215; also: JACM (1980) 701–717.

[28] J. Simon, On some central problems in computational complexity, Tech. Rept. TR 75-224, Dept. of Computer Sci., Cornell Univ., Ithaca, NY (1975).

[29] J. Simon, On the difference between the one and the many (preliminary version), Automata Languages and Programming, Springer Lecture Notes in Computer Science 52 (1977) 480–491.

[30] R. Solovay and V. Strassen, A fast Monte-Carlo test for primality, SIAM J. Comput. 6 (1977) 84–85.

[31] V. Strassen, Gaussian elimination is not optimal, Numerische Mathematik 13 (1969) 354–356.

[32] R.A. Trakhtenbrot, On problems solvable by successive trials, in: J. Beevar, ed., Math. Found. Comput. Sci., Lecture Notes in Computer Science 32 (1975).

[33] L.G. Valiant, The complexity of enumeration and reliability problems, SIAM J. Comput. 8 (1979) 410–440.

[34] L.G. Valiant, Completeness classes in algebra, Proc. 11th Annual ACM Symp. Theory Comput. (1979) 249–261.

[35] R. Zippel, Probabilistic algorithms for sparse polynomials, Symbolic and Algebraic Computation, Springer Lecture Notes in Computer Science 72 (1979) 216–226.