COMPUTATIONAL COMPLEXITY OF PROBABILISTIC TURING MACHINES

John T. Gill III
Stanford University, Stanford, CA

Abstract

Probabilistic Turing machines are Turing machines with the ability to flip coins in order to make random decisions. We allow probabilistic Turing machines small but nonzero error probability in computing number-theoretic functions. An example is given of a function computable more quickly by probabilistic Turing machines than by deterministic Turing machines. It is shown how probabilistic linear-bounded automata can simulate nondeterministic linear-bounded automata.

1. Introduction

Randomness is commonly introduced into computations in order to obtain qualititative results, as in the simulation of large systems, or to produce approximations to exact answers, as in the use of Monte Carlo methods for the evaluation of multidimensional integrals. In this paper, however, we shall consider probabilistic algorithms for computing single-valued number-theoretic functions. Our goal is to contrast the computation resources--time or storage--of probabilistic versus strictly deterministic algorithms.

Computability by probabilistic machines was first studied by DeLeeuw, Moore, Shannon, and Shapiro [3]. In that paper, the authors considered the question: Is there anything that can be done by a machine with access to random inputs that cannot be done by a deterministic machine? The random inputs were required to be drawn from a source of independent, equiprobable binary digits, and the tasks to be performed by the machines were the enumerations of sets of natural numbers. This question was answered in the negative. (In the case that the random inputs are independent but biased, say with bias $p \neq 1/2$, then possibly more sets can be listed by probabilistic devices, but only sets that are recursively enumerable in the real number p . In this paper we shall deal only with random elements that are equiprobable, independent bits.)

It is clear that a probabilistic procedure which computes a number-theoretic function with zero probability of error can be simulated by a deterministic procedure for all possible random inputs sequences. Since at least one random input sequence leads to a computation which halts, the deterministic procedure will eventually discover the correct answer.

We shall therefore allow probabilistic procedures to compute with a small but nonzero probability of error. So long as this error is less than $1/2$, it is still the case that no new functions can be computed probabilistically. However, it may be true that through the use of probabilistic devices, we can save on time of computation or memory size. We consider both of these two resource measures in this paper. This paper contains some results of [4].

In section 2, we present the basic machine class and give several general theorems. In particular, we show that all probabilistically computable functions are recursive; that there is a universal probabilistic machine; and that a version of Kleene's recursion theorem holds for probabilistic machines.

In section 3, we compare the computation times of probabilistic and deterministic machines. Average running time is seen not to satisfy the Blum axioms for complexity measures. We prove that at most exponential increases in computation speed can be gained through the use of probabilistic machines; that for certain relativized machine classes, this exponential speedup is achievable; and that there are decision problems that can be settled by a probabilistic machine infinitely often more quickly than by any deterministic machine.

Finally, in section 4, we give evidence that probabilistic machines are more efficient than deterministic machines in the use of tape storage. Any set accepted by a nondeterministic linear-bounded automaton can be recognized by a probabilistic linear-bounded automaton. (It is not known whether such sets can be recognized by deterministic linear-bounded automata.) More generally, every set accepted by a nondeterministic Turing machine within tape $L(n)$ is also recognized by some probabilistic Turing machine within tape $L(n)$.

2. Probabilistic Turing Machines

We can associate with each common class of computers an augmented class allowing the input of random elements. For example, we can modify the class of machine-independent FORTRAN programs (in which variables assume only integer values) by introducing a special subroutine RANDOM(N) ; each subroutine call to RANDOM(N) sets the variable N equal to 0 or 1 equiprobably and independent of previous subroutine calls. Although most of the results we obtain in this paper have a machine-independent character, we shall work exclusively with a single machine class, that of one tape Turing machines. Specifically, the machine class considered consists of one-tape Turing machines with one-way infinite tape. We allow work alphabets of arbitrarily many symbols and adopt a binary input/output convention.

Probabilistic Turing machines are derived from this class of machines by allowing, in addition to the normal action of the machines, instructions calling for random inputs. We adopt the following convention. Certain states of the machine are designated as "coin-tossing" states; on entering one of these states, the machine flips an unbiased coin and branches to one of two specified states according to the outcome of the coin toss.

Suppose that $M*$ is a probabilistic Turing machine. Then the computation of $M*$ with any input x is a random process driven by the random bits supplied to the machine. We will denote by $P\{M(x) = y\}$ the probability that $M*$ gives output y with input x ; by convention, $P\{M(x) = \infty\}$ will denote the probability that $M*$ does not halt on input x . It is reasonable to take as the output of a probabilistic Turing machine the majority output, the (at most one) number y such that $P\{M(x) = y\} > 1/2$. If there is no such y , then the output of the partial function computed by $M*$ is not defined for input x . In all practical cases, we must insist that $P\{M(x) = \varphi(x)\} > \lambda$ for some fixed $\lambda > 1/2$, if the machine $M*$ is to be considered to compute the partial function φ .

We can list the probabilistic Turing machines in a standard fashion and thereby speak of the i th probabilistic Turing machine M_i^* . We shall denote by φ_i the partial function computed by the i th PTM. Then the class $\{\varphi_i^*\}$ includes all and only partial recursive functions. Every partial recursive function is included in this list because every deterministic Turing machine is also a probabilistic Turing machine. Conversely, every partial function computed by a PTM is partial recursive.

Proposition 1: There is an effective procedure which, given the index i of probabilistic machine M_i^* , constructs a deterministic Turing machine M that computes φ_i^* .

Proof: This result is intuitively clear. Given a probabilistic machine $M*$, we simply simulate it deterministically by running $M*$ on the given input with all possible streams of random inputs until we determine that some integer y is the output with probability exceeding $1/2$. Q.E.D.

The next three propositions show that the class of probabilistic Turing machines induces an acceptable Godel numbering in the sense of Rogers [7] for the class of partial recursive functions.

Proposition 2: (Universal probabilistic Turing machine) There is a probabilistic Turing machine M_u^* such that M_u^* with input (e,x) has the same output distribution as M_e^* with input x ; that is,

$$P\{M_u^* (e,x) = y\} = P\{M_e^* (x) = y\} \quad \text{for every } y .$$

In particular, $\varphi_u^* (e,x) = \varphi_e^* (x)$.

Proposition 3: (S-m-n theorem) There is a recursive function s such that M_i^* with input (e,x) has the same output distribution as $M_{s(i,e)}^*$ with input x ; that is,

$$P\{M_i^* (e,x) = y\} = P\{M_{s(i,e)}^*(x) = y\} \quad \text{for every } y .$$

In particular, $\varphi_i^* (e,x) = \varphi_{s(i,e)}^*(x)$.

Proposition 4: (Recursion theorem) Let f be any total recursive function. Then there is an index e such that M_e^* and $M_{f(e)}^*$ have the same output distribution; that is,

$$P\{M_e^* (x) = y\} = P\{M_{f(e)}^*(x) = y\} \quad \text{for every } y .$$

In particular, $\varphi_e^* = \varphi_{f(e)}^*$.

3. Running Times

In this section, we investigate the possible gains in speed of computation through the use of probabilistic rather than deterministic Turing machines. We must first agree on how to measure the running time of a probabilistic machine.

One possibility is maximum running time. However, the maximum running time does not generally reflect the actual behavior of the probabilistic machine. For example, the maximum running time might be infinite in cases where the average running time is finite.

The most natural measure of time of computation of a probabilistic device is average running time. This is the measure that we will use in practical considerations. A theoretical objection to the use of average running time is that average running times of probabilistic Turing machines do not satisfy the Blum axioms [2] for complexity measures. Let us denote the average running time of the i th probabilistic Turing machine by $\bar{\Phi}_i^*$.

Theorem 1: There does not exist a 0,1-valued recursive function $C(i,x,y)$ such that

$$C(i,x,y) = \begin{cases} 1 & \text{if } [\bar{\Phi}_i^* (x)] \leq y \\ 0 & \text{otherwise} \end{cases}$$

(Here $[x]$ denotes the greatest integer in x .)

As a consequence of Theorem 1, we adopt the following formal definition of the running time of a probabilistic Turing machine.

Definition: Suppose $M*$ is a probabilistic Turing machine. Denote by $P\{M*(x) = y \text{ in } n \text{ steps}\}$ the probability that $M*$ with input x gives output y in a computation of length precisely n . Then the running time of $M*$ with input x is defined by

$$T_{M*}(x) = \begin{cases} \text{smallest } n \text{ such that } P\{M*(x) = y \text{ in } n \text{ steps}\} \\ \quad \text{where } y \text{ is the output of } M* \text{ with input } x \\ \\ \text{undefined or infinite if the output of } M* \\ \quad \text{with input } x \text{ is undefined} \end{cases}$$

We will denote the running time of the i th probabilistic machine by Φ_i^* .

Our first observation is that at most exponential speedup can be achieved by probabilistic computation. The proof of the following theorem is by a straightforward simulation [4].

Theorem 2: If M^* is a probabilistic Turing machine with running time $T(x)$, then there is a deterministic Turing machine M which simulates M^* in time at most $O(T(x)^4\, 2^{T(x)}) + |x|$, where $|x|$ is the length of the input x.

Theorem 2 may be nearly as strong as possible. There is evidence that no effective procedure can produce a faster simulation for all probabilistic Turing machines. We can consider relativized probabilistic machines with recursive oracles, where each consultation of the oracle requires one computational step. Then for suitable oracles, the exponential speedup of computation by probabilistic machines can be achieved. Since the relativized and unrelativized classes of deterministic and probabilistic Turing machines are formally almost identical, we would expect that any uniform procedure for deterministically simulating probabilistic machines in less than exponential time could be carried over to the relativized classes. That would provide a contradiction to the next proposition.

Theorem 3: There is a recursive oracle A and a 0,1-valued recursive function f such that every deterministic Turing machine (with oracle A) that computes f requires at least $2^{|x|}$ steps for all but finitely many inputs x, but there is a probabilistic Turing machine (with oracle A) that computes f in time $O(|x|)$.

Proof: The function f is related to the oracle A by

$$f(x) = \begin{cases} 1 & \text{if } A \text{ contains a majority of numbers of length } |x| + 3 \\ 0 & \text{otherwise.} \end{cases}$$

The oracle A is constructed by diagonalization so that two conditions are satisfied: (1) to compute f with oracle A requires $2^{|x|}$ steps almost everywhere; (2) for each n, either at most $1/4$ or at least $3/4$ of all $(n+3)$-bit numbers belong to A. Then a probabilistic Turing machine can calculate f as follows: With input x of length n, select at random a number of length $n+3$ and query the oracle A about this randomly chosen number. Output 1 for $f(x)$ if and only if this number belongs to A. This procedure computes f in running time $O(|x|)$ with error probability at most $1/4$. Q.E.D.

Theorem 3 is very much in the spirit of similar results for relativized nondeterministic machines [1,5].

The next theorem gives an example of probabilistic speedup for the class of Turing machines without oracles. The amount of the speedup is much less than exponential.

Theorem 4: There is a 0,1-valued recursive function f such that for every $\epsilon > 0$ there is a probabilistic Turing machine M^* such that

(1) M^* computes f with error probability less than ϵ;

(2) if M is any deterministic Turing machine which computes f, then the maximum running time of $M^*(x)$ is less than the running time of $M(x)$ for infinitely many inputs x.

Proof: The function f is the characteristic function of the palindrome-like language L of binary strings defined by

$$L = \{\alpha\beta : |\alpha| = |\beta| = 2^k \text{ and } \alpha[i] = \beta[i] \text{ for more than } 3/4 \text{ of the numbers } i \text{ between } 1 \text{ and } 2^k\}$$

(Here $\alpha[i]$ denotes the i th symbol of the string α.)

Clearly there is a one-tape Turing machine which recognizes L-- that is, computes f--in time $O(n^2)$, where n is the length of the input x. Now let $\bar{L}$ be the sublanguage defined by

$$\bar{L} = \{\alpha\beta : |\alpha| = |\beta| = 2^k \text{ and } \alpha = \beta\}.$$

Using crossing sequence arguments of Hennie [6], we can show that for every deterministic Turing machine M which computes f, the running time of M is at least Kn^2 for infinitely many inputs in $\bar{L}$. (K is a constant which depends on M.)

On the other hand, there is a probabilistic Turing machine M^* which computes f with maximum running time $O(n \log n)$ for inputs in the sublanguage $\bar{L}$. M^* operates as follows:

I. M^* checks that the input string can be expressed in the form $\alpha\beta$ where $|\alpha| = |\beta| = 2^k$. Otherwise M^* goes to step III. (This step takes $O(n \log n)$ steps.)

II. Having determined that $|\alpha| = |\beta| = 2^k$, the machine M^* makes s random selections of numbers between 1 and 2^k, say $i_1, i_2, \ldots, i_s$. Then M^* compares $\alpha[i_j]$ with $\beta[i_j]$ for $j = 1,2,\ldots,s$. If all s of these pairs of bits agree, then M^* accepts the input x as a member of L and outputs $f(x) = 1$. If not all agree, then M^* goes to step III. (Time required is $O(n \log n)$ steps.)

III. M^* executes some standard deterministic procedure for computing $f(x)$.

Now inputs in $\bar{L}$ will always cause M^* to give an output $f(x) = 1$ in step II, and so the running time of M^* on $\bar{L}$ is $O(n \log n)$. Moreover, the probability of incorrectly computing $f(x) = 1$ in step II is no more than $(3/4)^s$. So for suitably large s, this error can be made smaller than ϵ. Q.E.D.

We believe that for the unrelativized classes of Turing machines, only speedups for infinitely many inputs can be achieved by probabilistic machines.

Conjecture: If f is a recursive function computed in time T^* by some probabilistic Turing machine with error probability bounded away from $1/2$, then there is a deterministic Turing machine which computes f in time $O(T^*(x))$ for infinitely many x.

4. Storage Requirements

Probabilistic Turing machines are potentially more efficient in the use of tape than deterministic Turing machines.

Theorem 5: Every language that is accepted by a nondeterministic linear-bounded automaton can be recognized by a probabilistic linear-bounded automaton with small error probability.

Proof: Let S be any set accepted by a nondeterministic linear-bounded automaton M. Without loss of generality, we can assume that M can branch to at most two states from any nondeterministic branch state and that for some integer $c > 0$, every computation of M on an input of length n halts within 2^{cn} steps. Then a probabilistic linear-bounded automaton M^* that recognizes S operates as follows:

I. M^* saves the input on one track of its tape.

II. M^* copies the saved input onto another track, then probabilistically simulates M on this tape track. Whenever M encounters a nondeterministic branch point, then M^* flips a coin to decide which state to branch to. When M halts (after at most 2^{cn} simulated steps) then M^* goes to step III.

III. If M (simulated) halted in an accepting state, then M^* accepts and halts. Otherwise, M^* flips 2^{dn} coins (where $d > c$), keeping count on its tape. (Since n is the input length, M^* can comfortably count as high as 2^{dn}.) If all 2^{dn} tosses are heads, then M^* halts and rejects the input. Otherwise, M^* returns to step II.

First note that M^* halts with probability 1 on every input. (For an input of length n, the average running time is bounded by $g(n)2^{2^{dn}}$, where $g(n)$ is a function of exponential order.) Clearly, M^* cannot falsely accept numbers not in S, so that the probability of error is 0 for inputs not in S. Suppose now that the input belongs to S. We will bound above the probability that M^* will reject this input.

Intuitively, M^* can be expected to execute the loop II-III about $2^{2^{dn}}$ times before witnessing a coin tossing sequence of 2^{dn} heads and therefore halting. The probability that M^* randomly chooses an accepting computation in one execution of step III is at least $2^{-2^{cn}}$. Thus the probability that M^* does not find an accepting computation before halting is at most

$$(1 - 2^{-2^{cn}})^{2^{2^{dn}}} \approx e^{-(2^{dn} - 2^{cn})}$$

a vanishingly small number provided $d > c$.

One can easily replace this informal argument by an exact estimate of the probability of error. In fact, we can obtain a rigorous upper bound to the error probability of

$$2^{-(2^{dn} - 2^{cn})}$$ Q.E.D.

It is not known in general whether nondeterministic linear-bounded automata can be simulated by deterministic linear-bounded automata [8]. Thus the preceding theorem gives evidence that probabilistic Turing machines can compute functions using less tape than deterministic Turing machines.

Theorem 6: Suppose that S is a set accepted by a nondeterministic Turing machine within tape L; that is, for every input x in S, there exists an accepting computation using at most $L(x)$ tape squares. Then there is a probabilistic Turing machine which recognizes S within tape L.

Proof: Modify the technique of the last theorem. Q.E.D.

Conclusions

This paper has introduced one model for probabilistic computation, a model which is closely related to the notion of nondeterministic computation. (In fact, Janos Simon [9] has suggested an approach which may include both probabilistic and nondeterministic computation within a single framework.) There are other possible models for probabilistic computation. In particular, our insistence that probabilistic machines should give correct outputs for all inputs may be too restrictive, and we might well consider machines which work correctly for "most" inputs. In this case, specifying what is meant by "most" inputs will involve pinning down natural probability measures for the sets of natural numbers or input strings.

A number of important open questions remain to be settled for our model of probabilistic computation. These questions include

1. Is exponential speedup achievable through the use of probabilistic Turing machines? (This is similar to the P = NP question.)

2. Must all probabilistic speedups be of the infinitely often kind, as in Theorem 4? (Conjecture: yes.)

3. What are the tradeoffs, if any, between the speed and accuracy of probabilistic Turing machines?

4. Find a firm example of a function which requires less tape to compute by probabilistic machines than by deterministic machines.

5. Give an upper bound to the amount of tape required for the deterministic simulation of probabilistic Turing machines. (It seems that the methods of Savitch [8] will yield an upper bound of roughly an exponential increase.)

Acknowledgments

This research was directed by Manuel Blum and was supported by National Science Foundation Grant GJ-708 at the University of California, Berkeley, and Joint Services Electronics Program Contract N-00014-67-A-0112-0044 at Stanford University.

References

[1] T.P. Baker, Computational Complexity and Nondeterminism in Flowchart Programs. Ph.D. dissertation, Cornell University, January, 1974.

[2] M. Blum, A Machine-Independent Theory of the Complexity of Recursive Functions. J. ACM 14, (1967), pp. 322-336.

[3] K. de Leeuw, E.F. Moore, C.E. Shannon, and N. Shapiro, Computability by Probabilistic Machines. Automata Studies, Annals of Mathematics Studies, No.34, Princeton U. Press, Princeton, N.J., 1956.

[4] J.T. Gill III, Probabilistic Turing Machines and Complexity of Computation. Ph.D. dissertation, University of California, Berkeley, June, 1972.

[5] J.T. Gill III, Relativized Nondeterministic Computations. Manuscript, University of California, Berkeley, June, 1972.

[6] F.C. Hennie, One-Tape, Off-Line Turing Machine Computations. Inf. Contr. 8, (1965), pp. 553-578.

[7] H. Rogers, Jr., Godel Numberings of Partial Recursive Functions. J. Symbolic Logic 23, (1958), pp. 331-341.

[8] W.J. Savitch, Deterministic Simulation of Nondeterministic Turing Machines (detailed abstract). Conf. Rec. ACM Symp. on Theory of Computing, 1969, pp. 247-248.

[9] J. Simon, personal communication.