

# On the difficulty of computations

*Two practical considerations concerning the use of computing machinery are the amount of information that must be given to the machine for it to perform a given task and the time it takes the machine to perform it. The size of programs and their running time are studied for mathematical models of computing machines. The study of the amount of information (i.e., number of bits) in a computer program needed for it to put out a given finite binary sequence leads to a definition of a random sequence; the random sequences of a given length are those that require the longest programs. The study of the running time of programs for computing infinite sets of natural numbers leads to an arithmetic of computers, which is a distributive lattice. [This paper was presented at the Pan-American Symposium of Applied Mathematics, Buenos Aires, Argentina, August 1968.]*

## Section I

The modern computing machine sprang into existence at the end of World War II. But already in 1936 Turing and Post had proposed a mathematical model of computing machines (figure 1).<sup>1</sup> The mathematical model of the computing machine that Turing and Post proposed, commonly referred to as the Turing machine, is a black box with a finite number of internal states. The box can read and write on an infinite paper tape, which is divided into squares. A digit or letter may be written on each square of the tape, or the square may be blank. Each second the machine performs one of the following

---

<sup>1</sup>Their papers appear in Davis [1]. As general references on computability theory we may also cite Davis [2]–[4], Minsky [5], Rogers [6], and Arbib [7].

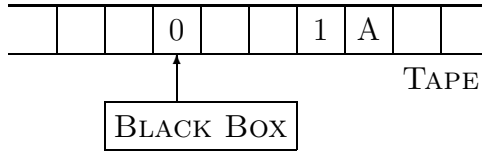


Figure 1. A Turing-Post machine

actions. It may stop, it may shift the tape one square to the right or one square to the left, it may erase the square on which the read-write head is positioned, or it may write a digit or letter on the square on which the read-write head is positioned. The action it performs is determined solely by the internal state of the black box at the moment, and the current state of the black box is determined solely by its previous internal state and the character read on the square of the tape on which its read-write head was positioned.

Incredible as it may seem at first, a machine of such primitive design can multiply numbers written on its tape, and can write on its tape the successive digits of  $\pi$ . Indeed, it is now generally accepted that any calculation that a modern electronic digital computer or a human computer can do, can also be done by such a machine.

## Section II

How much information must be provided to a computer in order for it to perform a given task? The point of view we will present here is somewhat different from the usual one. In a typical scientific application, the computer may be used to analyze statistically huge amounts of data and produce a brief report in which a great many observations are reduced to a handful of statistical parameters. We would view this in the following manner. The same final result could have been achieved if we had provided the computer with a table of the results, together with instructions for printing them in a neat report. This observation is, of course, ridiculous for all practical purposes. For, had we known the results, it would not have been necessary to use a computer. This example, then, does not exemplify those aspects of

computation that we will emphasize.

Rather, we are thinking of such scientific applications as solving the Schrödinger wave equation for the helium atom. Here we have no data, only a program; and the program will produce after much calculation a great deal of printout. Or consider calculating the apparent positions of the planets as observed from the earth over a period of years. A small program incorporating the very simple Newtonian theory for this situation will predict a great many astronomical observations. In this problem there are no data—only a program that contains, of course, a table of the masses of the planets and their initial positions and velocities.

## Section III

Let us now consider the problem of the amount of information that it is necessary to provide to a computer in order for it to calculate a given finite binary sequence. A computing machine is defined for these purposes to be a device that accepts as input a program, performs the calculations indicated to it in the program, and finally puts out the binary sequence it has calculated. In line with the mathematical theory of information, it is natural for the program to be viewed as a sequence of bits or 0's and 1's. Furthermore, in computer engineering all programs and data are represented in the machine's circuits in binary form. Thus, we may consider a computer to be a device that accepts one binary sequence (the program) and emits another (the result of the calculation).

011001001→COMPUTER→1111110010001100110100

As an example of a computer we would then have an electronic digital computer that accepts programs consisting of magnetized spots on magnetic tape and puts out its results in the same form. Another example is a Turing machine. The program is a series of 0's and 1's written on the machine's tape at the start of the calculation, and the result is a sequence of 0's and 1's written on its tape when it stops. As was mentioned, the second of these examples can do anything that the first can.

## Section IV

We are interested in the amount of information that must be supplied to a computer  $M$  in order for it to calculate a given finite binary sequence  $S$ . We may now define this as the size or length of the smallest binary sequence that causes the machine  $M$  to calculate  $S$ . We denote the length of the shortest program for  $M$  to calculate  $S$  by  $L(M, S)$ . It has been shown that there is a computing machine  $M$  that has the following three properties.<sup>2</sup>

1)  $L(M, S) \leq k + 1$  for all binary sequences  $S$  of length  $k$ .

In other words, any binary sequence of length  $k$  can be calculated by this computer  $M$  if it is given an appropriate program at most  $k + 1$  bits in length. The proof is as follows. If no better way to calculate a binary sequence occurs to us, we can always include the binary sequence as a table in the program. This computer is so designed that we need add only a single bit to the sequence to obtain a program for computing it. The computer  $M$  emits the sequence  $S$  when it is given the program  $S0$ .

2) Those binary sequences  $S$  for which  $L(M, S) < j$  are fewer than  $2^j$  in number.

Thus, most binary sequences of length  $k$  require programs of about the same length  $k$ , and the number of sequences that can be computed by smaller programs decreases exponentially as the size of the program decreases. The proof is as follows. There are only  $2^j - 2$  binary sequences less than  $j$  in length. Thus, there are fewer than  $2^j$  programs less than  $j$  in length, for each program is a binary sequence. At best, a program will cause the computer to calculate a single binary sequence. At worst, an error in the program will trap the computer in an endless loop, and no binary sequence will be calculated. As each program causes the computer to calculate at most one binary sequence, the number of sequences calculated must be smaller than the number of programs. Thus, fewer than  $2^j$  binary sequences can be calculated by means of programs less than  $j$  in length.

3) For any other computer  $M'$  there exists a constant  $c(M')$  such that for all binary sequences  $S$ ,  $L(M, S) \leq L(M', S) + c(M')$ .

---

<sup>2</sup>Solomonoff [8] was the first to employ computers of this kind.

In other words, this computer requires shorter programs than any other computer, or more exactly it does not require programs much longer than those required by any other computer. The proof is as follows. The computer  $M$  is designed to interpret the circuit diagrams of any other computer  $M'$ . Given a program for  $M'$  and the circuit diagrams of  $M'$ , the computer  $M$  proceeds to calculate how  $M'$  would behave, i.e., it proceeds to simulate  $M'$ . Thus, we need only add a fixed number of bits to any program for  $M'$  in order to obtain a program that enables  $M$  to calculate the same result. This program for  $M$  is of the form  $PC1$ .

The 1 at the right end of the program indicates to the computer  $M$  that this is a simulation,  $C$  is a fixed binary sequence of length  $c(M') - 1$  giving the circuit diagrams of the computer  $M'$ , which is to be imitated, and  $P$  is the program for  $M'$ .<sup>3</sup>

## Section V

Kolmogorov [9] and the author [11], [12] have independently suggested that computers such as those previously described be applied to the problem of defining what is meant by a random or patternless finite binary sequence of 0's and 1's. In the traditional foundations of the mathematical theory of probability, as expounded by Kolmogorov in his classic [10], there is no place for the concept of an individual random sequence of 0's and 1's. Yet it is not altogether meaningless to say that the sequence

110010111110011001011110000010

is more random or patternless than the sequences

11111111111111111111111111111111  
01010101010101010101010101010101,

for we may describe these last two sequences as thirty 1's or fifteen 01's, but there is no shorter way to specify the first sequence than by just writing it all out.

We believe that the random or patternless sequences of a given length are those that require the longest programs. We have seen that most of the

---

<sup>3</sup>How can the computer  $M$  separate  $PC$  into  $P$  and  $C$ ?  $C$  has each of its bits doubled, except the pair of bits at its left end. These are unequal and serve as punctuation separating  $C$  from  $P$ .

binary sequences of length  $k$  require programs of about length  $k$ . These, then, are the random or patternless sequences. Those sequences that can be obtained by putting into a computer a program much shorter than  $k$  are the nonrandom sequences, those that possess a pattern or follow a law. The more possible it is to compress a binary sequence into a short program calculation, the less random is the sequence.

As an example of this, let us consider those sequences of 0's and 1's in which 0's and 1's do not occur with equal frequency. Let  $p$  be the relative frequency of 1's, and let  $q = 1 - p$  be the relative frequency of 0's. A long binary sequence that has the property that 1's are more frequent than 0's can be obtained from a computer program whose length is only that of the desired sequence reduced by a factor  $H(p, q) = -p \log_2 p - q \log_2 q$ . For example, if 1's occur approximately  $\frac{3}{4}$  of the time and 0's occur  $\frac{1}{4}$  of the time in a long binary sequence of length  $k$ , there is a program for computing that sequence with length only about  $H(\frac{3}{4}, \frac{1}{4})k = 0.80k$ . That is, the program need be only approximately 80 percent the length of the sequence it computes. In summary, if 0's and 1's occur with unequal frequencies, we can compress such sequences into programs only a certain percentage (depending on the frequencies) of the size of the sequence. Thus, random or incompressible sequences will have about as many 0's as 1's, which agrees with our intuitive expectations.

In a similar manner it can be shown that all groups of 0's and 1's will occur with approximately the expected frequency in a long binary sequence that we call random; 01100 will appear  $2^{-5}k$  times in long sequences of length  $k$ , etc.<sup>4</sup>

## Section VI

The definition of random or patternless finite binary sequences just presented is related to certain considerations in information theory and in the methodology of science.

The two problems considered in Shannon's classical exposition [15] are to transmit information as efficiently and as reliably as possible. Here we are interested in examining the viewpoint of information theory concerning the efficient transmission of information. An information source may be redundant, and information theory teaches us to code or compress messages

---

<sup>4</sup>Martin-Löf [14] also discusses the statistical properties of random sequences.

so that what is redundant is eliminated and communications equipment is optimally employed. For example, let us consider an information source that emits one symbol (either an  $A$  or a  $B$ ) each second. Successive symbols are independent, and  $A$ 's are three times more frequent than  $B$ 's. Suppose it is desired to transmit the messages over a channel that is capable of transmitting either an  $A$  or a  $B$  each second. Then the channel has a capacity of 1 bit per second, while the information source has entropy 0.80 bits per symbol; and thus it is possible to code the messages in such a way that on the average  $1/0.80 = 1.25$  symbols of message are transmitted over the channel each second. The receiver must decode the messages; that is, expand them into their original form.

In summary, information theory teaches us that messages from an information source that is not completely random (that is, which does not have maximum entropy) can be compressed. The definition of randomness is merely the converse of this fundamental theorem of information theory; if lack of randomness in a message allows it to be coded into a shorter sequence, then the random messages must be those that cannot be coded into shorter messages. A computing machine is clearly the most general possible decoder for compressed messages. We thus consider that this definition of randomness is in perfect agreement and indeed strongly suggested by the coding theorem for a noiseless channel of information theory.

## Section VII

This definition is also closely related to classical problems of the methodology of science.<sup>5</sup>

Consider a scientist who has been observing a closed system that once every second either emits a ray of light or does not. He summarizes his observations in a sequence of 0's and 1's in which a 0 represents "ray not emitted" and a 1 represents "ray emitted." The sequence may start

0110101110...

and continue for a few million more bits. The scientist then examines the sequence in the hope of observing some kind of pattern or law. What does he mean by this? It seems plausible that a sequence of 0's and 1's is patternless

---

<sup>5</sup>Solomonoff [8] also discusses the relation between program lengths and the problem of induction.

if there is no better way to calculate it than just by writing it all out at once from a table giving the whole sequence. The scientist might state:

*My Scientific Theory:* 0110101110...

This would not be considered an acceptable theory. On the other hand, if the scientist should hit upon a method by which the whole sequence could be calculated by a computer whose program is short compared with the sequence, he would certainly not consider the sequence to be entirely patternless or random. The shorter the program, the greater the pattern he may ascribe the sequence.

There are many parallels between the foregoing and the way scientists actually think. For example, a simple theory that accounts for a set of facts is generally considered better or more likely to be true than one that needs a large number of assumptions. By “simplicity” is not meant “ease of use in making predictions.” For although general relativity is considered to be the simple theory par excellence, very extended calculations are necessary to make predictions from it. Instead, one refers to the number of arbitrary choices that have been made in specifying the theoretical structure. One is naturally suspicious of a theory whose number of arbitrary elements is of an order of magnitude comparable to the amount of information about reality that it accounts for.

## Section VIII

Let us now turn to the problem of the amount of time necessary for computations.<sup>6</sup> We will develop the following thesis. Call an infinite set of natural numbers perfect if there is no essentially quicker way to compute infinitely many of its members than computing the whole set. Perfect sets exist. This thesis was suggested by the following vague and imprecise considerations.<sup>7</sup>

One of the most profound problems of the theory of numbers is that of calculating large primes. While the sieve of Eratosthenes appears to be as quick an algorithm for calculating all the primes as is possible, in recent times hope has centered on calculating large primes by calculating a subset

---

<sup>6</sup>As general references we may cite Blum [16] and Arbib and Blum [17]. Our exposition is a summary of that of [13].

<sup>7</sup>See Hardy and Wright [18], Sections 1.4 and 2.5 for the number-theoretic background of the following remarks.

of the primes, those that are Mersenne numbers. Lucas's test can decide the primality of a Mersenne number with rapidity far greater than is furnished by the sieve method. If there are an infinity of Mersenne primes, then it appears that Lucas has achieved a decisive advance in this classical problem of the theory of numbers.

An opposing point of view is that there is no essentially better way to calculate large primes than by calculating them all. If this is the case, it apparently follows that there must be only finitely many Mersenne primes.

These considerations, then, suggested that there are infinite sets of natural numbers that are arbitrarily difficult to compute, and that do not have any infinite subsets essentially easier to compute than the whole set. Here difficulty of computation refers to speed. Our development will be as follows. First, we define computers for calculating infinite sets of natural numbers. Then we introduce a way of comparing the rapidity of computers, a transitive binary relation, i.e., almost a partial ordering. Next we focus our attention on those computers that are greater than or equal to all others under this ordering, i.e., the fastest computers. Our results are conditioned on the computers having this property. The meaning of "arbitrarily difficult to compute" is then clarified. Last, we exhibit sets that are arbitrarily difficult to compute and do not have any subset essentially easier to compute than the whole set.

## Section IX

We are interested in the speed of programs for generating the elements of an infinite set of natural numbers. For these purposes we may consider a computer to be a device that once a second emits a (possibly empty) finite set of natural numbers and that once started never stops. That is to say, a computer is now viewed as a function whose arguments are the program and the time and whose value is a finite set of natural numbers. If a program causes the computer to emit infinitely many natural numbers in size order and without any repetitions, we say that the computing machine calculates the infinite set of natural numbers that it emits.

A Turing machine can be used to compute infinite sets of natural numbers; it is only necessary to establish a convention as to when natural numbers are emitted. For example, we may divide the machine's tape into two halves, and stipulate that what is written on the right half cannot be erased.

The computational scratchwork is done on the left half of the tape, and the successive members of the infinite set of natural numbers are written on the nonerasable squares in decimal notation, separated by commas, with no blank spaces permitted between characters. The moment a comma has been written, it is considered that the digits between it and the previous comma form the numeral representing the next natural number emitted by the machine. We suppose that the Turing machine performs a single cycle of activity (read tape; shift, write, or erase tape; change internal state) each second. Last, we stipulate that the machine be started scanning the first nonerasable square of the tape, that initially the nonerasable squares be all blank, and that the program for the computer be written on the first erasable squares, with a blank serving as punctuation to indicate the end of the program and the beginning of an infinite blank region of tape.

## Section X

We now order the computers according to their speeds.  $C \geq C'$  is defined as meaning that  $C$  is not much slower than  $C'$ .

What do we mean by saying that computer  $C$  is not much slower than computer  $C'$  for the purpose of computing infinite sets of natural numbers? There is a computable change of  $C$ 's time scale that makes  $C$  as fast as  $C'$  or faster. More exactly, there is a computable function  $f(n)$  (for example  $n!$  or  $n^{n^{n^{\dots}}}$  with  $n$  exponents) with the following property. Let  $P'$  be any program that makes  $C'$  calculate an infinite set of natural numbers. Then there exists a program  $P$  that makes  $C$  calculate the same set of natural numbers and has the additional property that every natural number emitted by  $C'$  during the first  $t$  seconds of calculation is emitted by  $C$  during the first  $f(t)$  second of calculation, for all but a finite number of values of  $t$ . We may symbolize this relation between the computers  $C$  and  $C'$  as  $C \geq C'$ , for it has the property that  $C \geq C'$  and  $C' \geq C''$  only if  $C \geq C''$ .

In this way, we have introduced an ordering of the computers for computing infinite sets of natural numbers, and it can be shown that a distributive lattice results. The most important property of this ordering for our present purposes is that there is a set of computers  $\geq$  all other computers. In what follows we assume that the computer that is used is a member of this set of fastest computers.

## Section XI

We now clarify what we mean by “arbitrarily difficult to compute.”

Let  $f(n)$  be any computable function that carries natural numbers into natural numbers. Such functions can get big very quickly indeed. For example consider the function  $n^{n^{n^{\dots}}}$  in which there are  $n^n$  exponents. There are infinite sets of natural numbers such that, no matter how the computer is programmed, at least  $f(n)$  seconds will pass before the computer emits all those elements of the set that are less than or equal to  $n$ . Of course, a finite number of exceptions are possible, for any finite part of an infinite set can be computed very quickly by including in the computer’s program a table of the first few elements of the set. Note that the difficulty in computing such sets of natural numbers does not lie in the fact that their elements get very big very quickly, for even small elements of such sets require more than astronomical amounts of time to be computed. What is more, there are infinite sets of natural numbers that are arbitrarily difficult to compute and include 90 percent of the natural numbers.

We finally exhibit infinite sets of natural numbers that are arbitrarily difficult to compute, and do not have any infinite subsets essentially easier to compute than the whole set. Consider the following tree of natural numbers (figure 2).<sup>8</sup> The infinite sets of natural numbers that we promised to exhibit are obtained by starting at the root of the tree (that is, at 0) and walking forward, including in the set every natural number that is stepped on.

It is easy to see that no infinite subset of such a set can be computed much more quickly than the whole set. For suppose we are told that  $n$  is in such a set. Then we know at once that the greatest integer less than  $n/2$  is the previous element of the set. Thus, knowing that 1 000 000 is in the set, we immediately produce all smaller elements in it, by walking backwards through the tree. They are 499 999, 249 999, 124 999, etc. It follows that there is no appreciable difference between generating an infinite subset of such a set, and generating the whole set, for gaps in an incomplete generation can be filled in very quickly.

It is also easy to see that there are sets that can be obtained by walking through this tree and are arbitrarily difficult to compute. These, then, are the sets that we wished to exhibit.

---

<sup>8</sup>This tree is used in Rogers [6], p. 158, in connection with retraceable sets. Retraceable sets are in some ways analogous to those sets that concern us here.

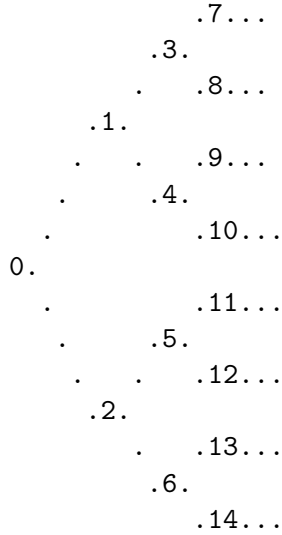


Figure 2. A tree of natural numbers

## Acknowledgment

The author wishes to express his gratitude to Prof. G. Pollitzer of the University of Buenos Aires, whose constructive criticism much improved the clarity of this presentation.

## References

- [1] M. Davis, Ed., *The Undecidable*. Hewlett, N.Y.: Raven Press, 1965.
- [2] —, *Computability and Unsolvability*. New York: McGraw-Hill, 1958.
- [3] —, “Unsolvable problems: A review,” *Proc. Symp. on Mathematical Theory of Automata*. Brooklyn, N.Y.: Polytech. Inst. Brooklyn Press, 1963, pp. 15–22.
- [4] —, “Applications of recursive function theory to number theory,” *Proc. Symp. in Pure Mathematics*, vol. 5. Providence, R.I.: AMS, 1962, pp. 135–138.
- [5] M. Minsky, *Computation: Finite and Infinite Machines*. Englewood Cliffs, N.J.: Prentice-Hall, 1967.
- [6] H. Rogers, Jr., *Theory of Recursive Functions and Effective Computability*. New York: McGraw-Hill, 1967.

- [7] M. A. Arbib, *Theories of Abstract Automata*. Englewood Cliffs, N.J.: Prentice-Hall (to be published).
- [8] R. J. Solomonoff, "A formal theory of inductive inference," *Inform. and Control*, vol. 7, pp. 1–22, March 1964; pp. 224–254, June 1964.
- [9] A. N. Kolmogorov, "Three approaches to the definition of the concept 'quantity of information'," *Probl. Peredachi Inform.*, vol. 1, pp. 3–11, 1965.
- [10] —, *Foundations of the Theory of Probability*. New York: Chelsea, 1950.
- [11] G. J. Chaitin, "On the length of programs for computing finite binary sequences," *J. ACM*, vol. 13, pp. 547–569, October 1966.
- [12] —, "On the length of programs for computing finite binary sequences: statistical considerations," *J. ACM*, vol. 16, pp. 145–159, January 1969.
- [13] —, "On the simplicity and speed of programs for computing infinite sets of natural numbers," *J. ACM*, vol. 16, pp. 407–422, July 1969.
- [14] P. Martin-Löf, "The definition of random sequences," *Inform. and Control*, vol. 9, pp. 602–619, December 1966.
- [15] C. E. Shannon and W. Weaver, *The Mathematical Theory of Communication*. Urbana, Ill.: University of Illinois Press, 1949.
- [16] M. Blum, "A machine-independent theory of the complexity of recursive functions," *J. ACM*, vol. 14, pp. 322–336, April 1967.
- [17] M. A. Arbib and M. Blum, "Machine dependence of degrees of difficulty," *Proc. AMS*, vol. 16, pp. 442–447, June 1965.
- [18] G. H. Hardy and E. M. Wright, *An Introduction to the Theory of Numbers*. Oxford: Oxford University Press, 1962.

**The following references have come to the author's attention since this lecture was given.**

- [19] D. G. Willis, "Computational complexity and probability constructions," Stanford University, Stanford, Calif., March 1969.
- [20] A. N. Kolmogorov, "Logical basis for information theory and probability theory," *IEEE Trans. Information Theory*, vol. IT-14, pp. 662–664, September 1968.
- [21] D. W. Loveland, "A variant of the Kolmogorov concept of complexity," Dept. of Math., Carnegie-Mellon University, Pittsburgh, Pa., Rept. 69-4.
- [22] P. R. Young, "Toward a theory of enumerations," *J. ACM*, vol. 16, pp. 328–348, April 1969.
- [23] D. E. Knuth, *The Art of Computer Programming*; vol. 2, *Seminumerical Algorithms*. Reading, Mass.: Addison-Wesley, 1969.
- [24] *1969 Conf. Rec. of the ACM Symp. on Theory of Computing* (Marina del Rey, Calif.).