

## Random Problems\*

YASER S. ABU-MOSTAFA

*Departments of Electrical Engineering and Computer Science, California Institute of Technology, Pasadena, California 91125*

Received July 7, 1987

A problem (a Boolean function  $f: \{0, 1\}^N \rightarrow \{0, 1\}$ ) is characterized by its randomness (*à la Kolmogorov*)  $R(f)$  and its entropy (*à la Shannon*)  $H(f)$ . Random problems have large values of  $R(f)$  and are a good model for many natural pattern recognition problems.  $R(f)$  and  $H(f)$  are shown to be lower and upper bounds, respectively, for a minimum-size circuit that computes  $f$ . False entropy, namely the hidden structure of a problem, is related to the difference between  $H(f)$  and  $R(f)$ . © 1988 Academic Press, Inc.

### 1. INTRODUCTION

It is difficult to find good mathematical models for many natural problems such as pattern recognition. Not only does this difficulty preclude finding good solutions for these problems, but it also precludes estimating their complexity using the standard tools of the theory of computational complexity (Traub, 1985). Part of the difficulty can be traced to symptoms such as ill-definition, fuzziness, and inexactness. However, the difficulty of modeling these problems may be inherent in some cases. To illustrate what we mean, consider the following problem:

*A. Input: 255-45-5237; output: Is this the social security number of a convict?* To solve this problem in general, one needs a list of the social security numbers of all convicts. It is highly improbable that there exists a simple relation between the social security number and the legal status of a person. Barring such a relation, one cannot hope for an algorithm to "manipulate" the input so as to arrive at the output. In other words, the

\* Research supported by the Air Force Office of Scientific Research under Grant AFOSR-86-0296.

above list cannot be compacted into a simple algorithm. Contrast this problem with the following familiar problem:

*B. Input: 255,455,237; output: Is this number a prime?* Although one can resort to consulting a list of primes, there is the option of writing a simple algorithm to test for primality and applying it to this number. It may take a long time to execute, but the algorithm itself is short in comparison with the list of primes. What is the basic difference between problems A and B? The notion of a prime has a short effective definition, while the notion of a convict does not. The long list of social security numbers of convicts is the effective definition of the notion of a convict.

Problems which do not have a concise effective definition are called *random problems* (Abu-Mostafa, 1985). Randomness here is based on the length of the shortest algorithm (Kolmogorov, 1965) and has nothing to do with probability or fuzziness. If this length exceeds a certain threshold, the problem is considered random. On the other hand, for *structured problems* such as B, the algorithm can be quite short. The difficulty in modeling random problems is inherent. This is because an effective model could be viewed as an algorithm (not necessarily a very efficient one), and hence has to be long in the case of a random problem.

Many natural pattern recognition problems can be considered random problems. This fact is usually overlooked due to the apparent "structure" some of these problems have, e.g., visual images which have many clear regularities. However, after these regularities are considered, a major random component is left. A complete model for visual scenes, or an algorithm for computer vision, will cover the random as well as the structured components of the problem, and hence will have to be sufficiently long.

Three factors contribute to the significance of random problems and widen their scope. First of all, the definition of algorithmic randomness is based on universal Turing machines (Turing, 1936) which are more powerful than any physical system. This makes some problems which are not truly random *look* random for all practical purposes and will have to be treated as such. The second factor is that there is no constructive way in general to find the shortest algorithm for an arbitrary problem (Kolmogorov, 1965). In spite of its generality, this fact reflects the difficulty of modeling and suggests that some problems will have to be treated as random problems just because no one will be able to find their concise model. Finally, although it may be logically impossible to tell whether a problem is random (Chaitin, 1982), a problem generated by probabilistic means is very likely to be random. Therefore, the assumption that a natural problem is random is probably valid.

This last remark leads to another observation. Most of the practical random problems turn out to be ill-defined as well. However, we maintain separation of concerns in this paper. Only randomness, as defined above,

is assumed in the problems we address here. Our aim is to characterize random problems and their computational demands.

In Section 2, we introduce the formal definitions of randomness and entropy and prove some basic facts about them. In Section 3, we relate these quantities to the complexity of implementing the function. Finally, we draw some insight into the class of random problems in Section 4. We shall restrict ourselves to binary alphabets for simplicity; the generalization to arbitrary finite alphabets is straightforward. All logarithms and exponentials are to the base 2.

## 2. RANDOMNESS AND ENTROPY

Let  $N$  be an arbitrary fixed positive integer and consider the Boolean functions of the form  $f: \{0, 1\}^N \rightarrow \{0, 1\}$ . Any such function is fully characterized by its truth table which can be listed as a ( $2^N$ -bit long) binary string  $\tau(f) = \tau_0\tau_1 \cdots \tau_k \cdots \tau_{2^N-1}$ , where  $\tau_k$  is the value of  $f$  when the argument is the  $N$ -bit binary representation of the number  $k$ .

Let  $U$  denote a fixed universal Turing machine with binary input alphabet  $\{0, 1\}$ .  $U$  takes a binary string  $\mathbf{p}$  as input (program) and runs on  $\mathbf{p}$  to produce an output string  $\mathbf{s}$  (if it eventually halts). In this case, we say that  $U(\mathbf{p}) = \mathbf{s}$ . Based on  $U$ , the Kolmogorov complexity of a string  $\mathbf{s}$  is defined by

$$K(\mathbf{s}) = \min \{|\mathbf{p}| \mid U(\mathbf{p}) = \mathbf{s}\},$$

where  $|\mathbf{p}|$  denotes the length of the string  $\mathbf{p}$ .

The Kolmogorov complexity measures the degree of randomness of a string; if two binary strings of the same length have different Kolmogorov complexities, the one with the higher complexity is more "random." The randomness of a Boolean function  $f$  is based on the Kolmogorov complexity of its truth table;

$$R(f) = \log K(\tau(f)) \text{ bits,}$$

where the logarithm (to the base 2) is taken to make the range of  $R(f)$  run from  $\approx 0$  (where  $\mathbf{p}$  is a very short program, hence  $\tau(f)$  has a very regular pattern) to  $\approx N$  (where  $\mathbf{p}$  is as long as  $\tau(f)$ , hence  $\tau(f)$  has no pattern whatsoever).

A problem will be considered random if the corresponding function  $f$  has a large value of  $R(f)$ . How large? We fix a threshold  $R_0$  and make the definition relative to  $R_0$ . The choice of a particular  $R_0$  is not critical to most of the theory, and may therefore be motivated by practical considerations.

**DEFINITION.** A problem  $f: \{0, 1\}^N \rightarrow \{0, 1\}$  is said to be *random* if  $R(f) \geq R_0$ .

Since  $R(f)$  can be at most  $\approx N$ , the threshold  $R_0$  should be smaller than  $N$ . This is necessary to make the definition interesting, i.e., to guarantee that some of the problems are indeed random. In fact, the overwhelming majority of all problems will be random even if  $R_0$  is only a few bits smaller than  $N$ . To see this, we observe that there are  $2^{2^N}$  problems (Boolean functions of  $N$  variables), while there are at most  $2^0 + 2^1 + \dots < 2^{K+1}$  programs  $\mathbf{p}$  of length  $\leq K$ . Therefore, at most  $2^{2^{R_0+1}}$  problems can be *nonrandom*, and this is only a negligible fraction of  $2^{2^N}$ .

On the other hand, if  $R_0$  is not very small, it will be impossible to pinpoint a specific problem and prove that it is random. This is a consequence of Chaitin's version of Gödel's incompleteness theorem; there is a number  $K_0$  (depends on the axiomatic system) such that no statement of the form " $K(\mathbf{s}) \geq K_0$ 's is provable (within the system). If we pick  $R_0 \geq \log K_0$ , where  $K_0$  corresponds to axiomatic set theory, it will be impossible to prove (using regular mathematics) that any given problem is random.

The difficulty of proving randomness for specific problems is by no means a serious drawback for the notion of random problems. There are many cases where the probability that the problem is random is sufficiently high to warrant treating it as a random problem, in spite of the lack of a *proof* of randomness. In fact, whenever probability is involved in generating,  $f$ , the chances are  $f$  will be a random problem.

**EXAMPLE.** Let  $f: \{0, 1\}^N \rightarrow \{0, 1\}$ , where  $N$  is large, be generated as follows. Each bit of the truth table  $\tau(f) = \tau_0 \cdot \dots \cdot \tau_{2^N-1}$  is (independently) set to 1 with probability  $\varepsilon$  and to 0 with probability  $1 - \varepsilon$ , where  $0 < \varepsilon < 1$ . The expected number of 1's in  $\tau(f)$  is therefore  $\varepsilon 2^N$ . With high probability (law of large numbers),  $f$  will have about that many 1's in its truth table. Therefore,  $f$  will be any of  $\approx \binom{2^N}{\varepsilon 2^N}$  functions with approximately equal probability. This number can be estimated as  $\binom{2^N}{\varepsilon 2^N} \approx 2^{\mathcal{H}(\varepsilon)2^N}$ , where  $\mathcal{H}(\varepsilon) = -\varepsilon \log \varepsilon - (1 - \varepsilon) \log(1 - \varepsilon)$ . We can again enumerate the programs  $\mathbf{p}$  of length, 0, 1, etc., and conclude that  $R(f) \geq N - \Delta$  with high probability, where  $\Delta$  is only a few bits more than  $-\log \mathcal{H}(\varepsilon)$ .

This example also illustrates that the randomness of a problem is affected by the number of 1's in the truth table. If the number of 1's is very small, one can write a short program  $\mathbf{p}$  to generate  $\tau(f)$  by specifying *where the 1's are* in  $\tau(f)$ . The same can be done in the dual case where the number of 0's is small. Problems which have few 1's (or few 0's) in their truth tables are of special interest because they model the cases where only a small fraction of inputs to  $f$  are of interest, a condition commonly encountered in natural problems. The quantity that captures this property is entropy.

Let  $h(f) = \min\{|f^{-1}(1)|, |f^{-1}(0)|\}$ , i.e., the number of 1's or the number of 0's (whichever is smaller) in the truth table of  $f$ . The (deterministic) entropy of  $f$  is defined by

$$H(f) = \log(1 + h(f)) \text{ bits.}$$

The logarithm (and the added 1) make the range of  $H(f)$  run from 0 (the two constant functions) to  $\approx N$  (functions with as many 1's as 0's). Hence,  $H(f)$  has essentially the same range as  $R(f)$ .

Except for a small "error" term (at most the order of  $\log N$ ),  $H(f)$  serves as an upper bound for  $R(f)$ . To see this, we assume that  $\tau(f)$  has only a limited number of 1's (the dual case of 0's is similar) and write a relatively short program  $\mathbf{p}$  that generates  $\tau(f)$ . The program is a listing of the locations of the 1's in  $\tau(f)$ . Since each location in  $\tau(f)$  can be specified by  $N$  bits, the length of  $\mathbf{p}$  is approximately  $N2^{H(f)}$ . The logarithm of that, which is an upper bound for  $R(f)$ , is  $H(f) + \log N$ . An enumeration of all programs of length 0, 1, etc., and of the number of functions of a certain level of entropy shows that  $H(f)$  is a *tight* upper bound for  $R(f)$ . In fact, for most functions,  $R(f) \approx H(f)$ .

It is interesting to notice that  $R(f)$  and  $H(f)$  can be considered two extremes in a spectrum of quantities that measure the complexity of specifying  $f$  based on models of varying degree of sophistication. On the one hand,  $H(f)$  measures the complexity of specifying  $f$  if the specification is done by simply listing the 1's or the 0's of the function. Hence, the model on which the measure  $H(f)$  is based is the "lookup" model. On the other hand, the model on which  $R(f)$  is based is the universal Turing machine.  $R(f)$  measures the complexity of specifying  $f$  based on a very powerful tool that generates  $\tau(f)$  from the specification. Hence  $R(f)$  is as small as can be; it can take advantage of any effective property  $f$  may have. There are many models that fall between these two extremes. For example, a time-bounded version of  $R(f)$  can be based on the time-bounded Kolmogorov complexity of  $\tau(f)$ . The underlying model will be a universal Turing machine that is allowed to run for only a limited number of steps. Another model which is treated in more detail in Section 3 is combinational circuits, where regularities of  $f$  that can be captured by logic elements help reduce the size of the specification of  $f$ .

The difference between  $H(f)$  and  $R(f)$  is a significant quantity. It is called *false entropy* (Abu-Mostafa, 1986) and expresses how much "hidden" structure the problem has. This interpretation follows from the above discussion; the model on which  $H(f)$  is based is the lookup model that does not take any advantage of the location of the 1's and 0's of  $f$ , just their number, while the model on which  $R(f)$  is based takes advantage of any effective regularity, no matter how subtle, to reduce the

size of the specification. The difference expresses how much of the entropy of  $f$  can be removed if the structure of  $f$  is taken into consideration. The problem of pattern recognition hinges on removing as much of the false entropy as possible.

### 3. COMPLEXITY BOUNDS

Consider the problem of implementing the function  $f$ , e.g., using a combinational circuit (Savage, 1976). We wish to estimate the complexity of such implementation and investigate the relation between complexity, entropy, and randomness. Since  $N$  is fixed, the number of input instances is finite and our measure of complexity will be a nonuniform one, i.e., not based on a finite process that works for any of an infinite number of input instances. Nonuniformity of the complexity measure in our context is more realistic for two reasons. First, the pattern recognition problems we are modeling are finite in nature with no clear extension into an infinite problem. Second, the systems that are projected for pattern recognition are based on learning, i.e., automatic development of the system from training samples. As such, the final system does not have to be uniform although the learning mechanism itself may be uniform.

There are several ways of defining circuits all of which are equivalent, and a corresponding number of ways of defining circuit complexity (size) all of which are within a constant (independent of  $N$  and  $f$ ) from one another. For concreteness, we give one such definition in detail. Our circuits are combinational (loop-free), with unlimited fan-out (an output of a gate can be used as an input to any number of gates). The only type of gate we use is the two-input NAND gate (whose output is 0 if, and only if, both inputs are 1's) which by itself is a complete basis (any Boolean function can be simulated using a circuit consisting exclusively of copies of this gate). The independent Boolean variables (inputs of  $f$ ) are called  $x_1, \dots, x_N$ , and are available to be used as inputs to any gate in the circuit.

A circuit is a chain  $\Gamma = \gamma_1 \cdots \gamma_Q$  gates. The outputs of these gates are called  $y_1, \dots, y_Q$ , respectively. Each gate  $\gamma_q$  can have as inputs any of the independent variables  $x_1, \dots, x_N$  as well as the outputs of the *previous* gates  $y_1, \dots, y_{q-1}$ . Since all the gates are two-input NAND gates, we only need to specify the inputs to each  $\gamma_q$ . Therefore, formally, each  $\gamma_q$  is a pair (not necessarily distinct, order does not matter) of elements from the set  $\{x_1, \dots, x_N, y_1, \dots, y_{q-1}\}$ . Finally, the output of the circuit is the output of the last gate,  $y_Q$ . We say that a circuit  $\Gamma$  simulates a function  $f$  if  $f = y_Q$  for all assignments of the variables  $x_1, \dots, x_N$ . The number of gates in  $\Gamma$ , namely  $Q$ , is denoted by  $c(\Gamma)$ . The (circuit) complexity of a problem  $f$  is defined by

$$C(f) = \log \min\{c(\Gamma) \mid \Gamma \text{ simulates } f\} \text{ bits.}$$

Again we have the range of  $C(f)$  running from  $\approx 0$  (simple functions) to  $\approx N$  (complex functions). The fact that the maximum value  $C(f)$  can assume is  $\approx N$  follows from the exhaustive implementation of any Boolean function of  $N$  variables that uses  $\approx 2^N$  gates (which can be improved to  $2^N/N$  due to a classical result of Shannon). We now show that, except for an error term of at most the order of  $\log N$ , the value  $C(f)$  is at least  $R(f)$  and at most  $H(f)$ , which we write as

$$R(f) \leq C(f) \leq H(f).$$

This relationship reflects the fact that combinational logic is more sophisticated than table lookup, but less sophisticated than universal Turing machines.

To see that  $C(f) \leq H(f)$ , consider the case where  $f$  has  $h(f)$  1's (the dual case is similar). One can build a circuit with  $N$  one-input NOT gates,  $h(f)$   $N$ -input AND gates, and one  $h(f)$ -input OR gate to simulate  $f$  (implementing the 1's of the function directly by a sum of products). One can replace all these gates by at most  $\alpha N(1 + h(f))$  two-input NAND gates (where  $\alpha$  is a suitable constant). Hence  $c(f) \leq H(f) + \log N + \text{constant}$ . Therefore, apart from the error term, the bound is valid.

To see that  $C(f) \geq R(f)$ , consider a program  $\mathbf{p}$  that encodes a minimum-size circuit  $\Gamma = \gamma_1, \dots, \gamma_Q$  that simulates  $f$ . Each  $\gamma_q$  is a pair of variables selected from at most  $N + Q$  variables. Hence it takes at most  $2 \log(N + Q)$  bits to encode each  $\gamma_q$ . Hence, the program that encodes the whole circuit  $\Gamma$  will be at most  $\alpha Q \log(N + Q)$  bits long (where  $\alpha$  is a suitable constant). The logarithm of that is an upper bound for  $R(f)$ . By definition,  $\log Q = C(f)$ . The other (error) terms are at most the order of  $\log N$  since  $Q$  is at most the order of  $2^N$ .

The fact that  $R(f)$  is a lower bound for  $C(f)$  implies that random problems cannot be solved by small circuits.

#### 4. CONCLUSION

The notion of random problems was introduced to capture the inherent difficulty of natural pattern recognition problems and estimate their computational demands. The paper introduced the main concepts and proved some basic facts for the idealized case where the problem is defined as a deterministic Boolean function. The results are summarized in the relationship

$$R(f) \leq C(f) \leq H(f).$$

The methods of rate distortion may be employed to accommodate the practical case of continuous-valued functions. To accommodate the case where there is a probability distribution over the input variables, one can define a probabilistic version of the measures  $R(f)$ ,  $H(f)$ ,  $C(f)$ , e.g.,

$$R_\delta(f) = \min\{R(g) | \Pr(f \neq g) \leq \delta\}.$$

In words, the randomness of  $f$  with tolerance for error  $\delta$  of the time is the minimum randomness of any function  $g$  that differs from  $f$  at most  $\delta$  of the time.

We also made the remark that the difference between  $H(f)$  and  $R(f)$  highlights the hidden structure of the problem which a pattern recognition system has to be able to detect, at least partially.

#### ACKNOWLEDGMENT

I thank Mr. Alan Blanchard for his assistance.

#### REFERENCES

- ABU-MOSTAFA, Y. S. (1985), Complexity of random problems, in "Abstracts of Papers, IEEE International Symposium on Information Theory, Brighton, England," IEEE Catalog #85 CH 2201-2, p. 84.
- ABU-MOSTAFA, Y. S. (1986), The complexity of information extraction, *IEEE Trans. Inform. Theory* **IT-32**, 513-525.
- ABU-MOSTAFA, Y. S., AND PSALTIS, D. (1987), Optical neural computers, *Sci. American*, **256** (3), 88-95.
- CHAITIN, G. J. (1982). Gödel's theorem and information. *Internat. J. Theoret. Phys.* **22**, 941-954.
- KOLMOGOROV, A. N. (1965), Three approaches for defining the concept of information quantity, *Inform. Transmission* **1**, 3-11.
- MARTIN-LÖF, P. (1966), The definition of random sequences, *Inform. and Control* **9**, 602-619.
- PIPPENGER, N. (1977), Information theory and the complexity of Boolean functions, *Math. Systems Theory* **10**, 129-167.
- SHANNON, C. E. (1948), A mathematical theory of computation, *Bell Syst. Tech. J.* **27**, 379-423.
- SAVAGE, J. E. (1976), "The Complexity of Computing," pp. 14-66, Wiley-Interscience, New York.
- TRAUB, J. F. (1985), Complexity of approximately solved problems, *J. Complexity* **1**, 3-10.
- TURING, A. M. (1936), On computable numbers with an application to the Entscheidungs problem, *Proc. London Math. Soc.* **42**, 230-265.