Two Lectures on Advanced Topics in Computability

Oded Goldreich
Department of Computer Science
Weizmann Institute of Science
Rehovot, Israel.
oded@wisdom.weizmann.ac.il

Spring 2002

Abstract

This text consists of notes for two short lectures on advanced topics in computability. The topic of the first lecture is Kolmogorov Complexity, and we merely present the most basic definitions and results regarding this notion. The topic of the second lecture is a comparison of two-sided error versus one-sided error probabilistic machines, when confined to the domain of finite automata.

Preface

These notes were prepared on the occasion of giving a guest lecture in David Harel's class on *Advanced Topics in Computability.* David's request was the lecture should not rely on resource bounds (equiv., complexity measures). This was a very challenging request for me, because all my professional thinking evolves around resource bounds. Still, I was able to find within my "knowledge base" two topics that meet David's request, although one may claim that I "cheated": these topics too are "related to quantities" and not merely to "qualities".

1 Kolmogorov Complexity

We start by presenting a paradox. Consider the following description of a natural number: *the largest natural number that can be described by an English sentence of up-to 1000 letters.* (Something is wrong, because if the above is well-defined then so is *the integer-successor of the largest natural number that can be described by an English sentence of up-to 1000 letters.*)

Jumping ahead, we point out that the paradox presupposes that any sentence is a legal description in some adequate sense. One adequate sense of legal descriptions is that there exists a procedure that given a (possibly succinct) "implicit" description of an object outputs the "explicit" description of the object. Passing from implicit descriptions to explicit descriptions is what Kolmogorov Complexity is about.

Let us fix a Turing machine M. The Kolmogorov Complexity (w.r.t M) of a binary string x, denoted $K_M(x)$, is the length of the shortest input y such that $M(y) = x$; that is, $K_M(x) \overset{\text{def}}{=} \min_y\{|y| : M(y) = x\}$. (In case there is no such y, we let $K_M(x) \overset{\text{def}}{=} \infty$.)

Clearly, $K_M(x)$ depends on M (and not only on x), and so a question that arises is what machine M should we fix? The answer is that any universal machine will do. This is justified by the following fact:

Proposition 1.1 *Let U be a universal Turing machine. Then for every Turing machine M, there exists a constant c such that $K_U(x) \leq K_M(x) + c$, for every $x \in \{0,1\}^*$.*

Thus, universal machines provides the "most expressive" syntax for describing strings. Furthermore, the Kolmogorov Complexity is the same (up-to an additive constant) for any two universal machines.

Proof Sketch: Suppose that y satisfies both $|y| = K_M(x)$ and $M(y) = x$, and consider what happens when the input $(\langle M\rangle, y)$ is fed to U. Clearly, $U(\langle M\rangle, y) = M(y) = x$. Using a suitable encoding of pairs (i.e., using a prefix-free code for the first input), we have $K_U(x) \leq |\langle M\rangle| + |y|$, and the proposition follows since $|\langle M\rangle|$ depends only on M. ■

Conceptual Discussion: Kolmogorov Complexity measures the length of the most succinct descriptions of phenomena (or strings), where descriptions are with respect to an "effective" universal language that comes together with a procedure for generating the full description of a phenomenon out of its succinct description. Whereas some phenomena have very succinct descriptions, most phenomena (i.e., random phenomena) do not have succinct descriptions. These facts are stated in Theorem 1.2.

Properties of Kolmogorov Complexity. In light of Proposition 1.1, we may fix an arbitrary universal machine U, and let $K(x) \overset{\text{def}}{=} K_U(x)$. The quantitative properties of Kolmogorov Complexity are captured by the following

Theorem 1.2 (KC – quantitative properties):

1. *There exists a constant c such that $K(x) \leq |x| + c$ for all x's.*
2. *There exist infinitely many x's such that $K(x) \ll |x|$. Furthermore, for every monotonically increasing recursive function $f : \mathsf{N} \to \mathsf{N}$ and infinitely many x's $K(x) < f^{-1}(|x|)$.*
3. *There exists infinitely many x's such that $K(x) \geq |x|$. Furthermore, for most x's of length n, it holds that $K(x) \geq |x| - 1$.*

Proof Sketch: Part 1 follows by applying Proposition 1.1 to the machine M_{id} that computes the identity function (and thus satisfies $M_{id}(x) = |x|$ for all x's). Part 2 can be demonstrated by binary strings of the form yy (for which $K(yy) \leq |y| + O(1)$). Another example is provided by strings of the form 1^n (for which $K(1^n) \leq \log_2 n + O(1)$). The furthermore part can be shown by defining $x_n = 1^{f(n)}$, and observing that $K(x_n) \leq O(1) + \log_2 n < f^{-1}(f(n)) = f^{-1}(|x_n|)$.

To prove Part 3, observe that if $K(x) = i$ then it means that there exists an input y of length i that makes the universal machine U output x (i.e., $U(y) = x$). Since each input may yield only one output, there exists a one-to-one mapping of x's with $K(x) = i$ to i-bit long strings (satisfying $U(y) = x$). Thus, $|\{x \in \{0,1\}^* : K(x) = i\}| \leq |\{y \in \{0,1\}^i : U(y) \in \{0,1\}^*\}| \leq 2^i$ and $|\{x \in \{0,1\}^n : K(x) \leq k\}| \leq 2^{k+1} - 1$ follows (for all k's and in particular for $k = n-1$ and $k = n-2$). ■

The main computational property of Kolmogorov Complexity is that it is not computable. That is:

Theorem 1.3 (KC – a computational property): *The Kolmogorov Complexity function K defined above is not computable.*

The proof will be outlined at the end of the lecture. It is very related to resolving the initial paradox.

Resolving the paradox. The paradox may be recast as follows. For every natural number n, we define $x_n \in \{0,1\}^*$ to be the largest string (according to the standard lexicographic order) that has Kolmogorov Complexity at most n; that is, $x_n \stackrel{\text{def}}{=} \max_x\{x : K(x) \leq n\}$. The paradox *implicitly and wrongly presupposes* that there exists an input y_n of length $O(\log n) \ll n$ that makes the universal machine output x_n (i.e., $x_n = U(y_n)$). (Above, we assumed that the string described by the paradox can be written in an adequate language (i.e., allowing explicit reconstruction) by using less than 1000 symbols.)

The paradox is actually a proof that x_n *cannot be produced by the universal machine on input that is much shorter than n.* This is proved by considering the string x'_n defined as the successor of x_n. Observe that if $y_n = (\pi_n, \lambda)$ is an input that makes U produces x_n then $y'_n = (\pi'_n, \lambda)$ makes U produces x'_n, where π'_n first invokes π_n and next invokes the constant program for computing successors (on the result of π_n). Thus, $K(x'_n) \leq |y'_n| = |y_n| + O(1) = K(x_n) + O(1) \leq n$ (using the contradiction hypothesis $K(x_n) \ll n$), which contradicts the definition of x_n.

We comment that the wrong intuition regarding the existence of short programs for generating x_n is implicitly based on the *wrong assumption* that given n we can effectively enumerate all strings having Kolmogorov Complexity less than n. (Note that if that was possible then we could have computed $K(x)$ by enumerating all strings having Kolmogorov Complexity less than i, for $i = 1, ..., |x| + O(1)$. This does NOT prove that K is not computable, because the reduction is in the wrong direction.)

Outline for the proof of Theorem 1.3: For every n, consider the string $z_n \stackrel{\rm def}{=} \min_z\{z : K(z) \geq n\}$. (Note that z_n is well-defined because there exists strings with Kolmogorov Complexity greater than n.) It is easy to show that if K is computable then there exists an input of length $\log_2 n + O(1)$ that makes the universal machine produce z_n. Thus, $K(z_n) \leq \log_2 n + O(1)$, which (for sufficiently large n) is impossible (because $K(z_n) \geq n$ by definition of z_n). ■

A related exercise: For any unbounded function $f : \{0,1\}^* \to \mathsf{N}$, define $z_n^f \stackrel{\rm def}{=} \min_z\{z : f(z) \geq n\}$. (Note that z_n^f is well-defined because f is unbounded.) Prove that if f is computable then there exists an input of length $\log_2 n + O(1)$ that makes the universal machine produce z_n^f. (Hint: given n, we generate z_n^f by computing f on finitely many inputs (i.e., all z's that precede z_n^f in lexicographic order as well as z_n^f itself).)

2 Probabilistic Finite Automata: Two-Sided versus One-Sided Error

Probabilistic computation defers from non-deterministic computation in that the former is concerned with the *quantity* of accepting computations, whereas the latter is only concerned with their *existence*. That is, a non-deterministic machine M is said to accept (or non-deterministically accept) the set L if

- For every $x \in L$ *there exists* a computation of $M(x)$ that halts in accepting state.
- For every $x \notin L$ *there does not exist* a computation of $M(x)$ that halts in accepting state.

In contrast, a *two-sided error probabilistic machine* M is said to accept (or probabilistically accept) the set L if

- For every $x \in L$ *a strict majority of* the computations of $M(x)$ halt in an accepting state.
- For every $x \notin L$ *a strict majority of* the computations of $M(x)$ halt in a rejecting state.

We stress that (unlike in standard complexity-theoretic treatments), we only required a separation of the two cases, rather a significant "separation-gap"; that is, we only required a strict majority in each direction, rather than asking for a special majority (e.g., a 2/3-majority).

We also consider *one-sided error probabilistic machines*. Such a machine is said to accept the set L (with one-sided error on yes-instances) if

- For every $x \in L$ *a strict majority of* the computations of $M(x)$ halt in an accepting state.
- For every $x \notin L$ *there does not exist* a computation of $M(x)$ that halts in accepting state.

Throughout the lecture we focus on finite automata (i.e., M above is a finite automaton). We mention that similar phenomena (regarding two-sided error that is *not bounded-away from* 1/2) seem to occur also in other models.

The power of one-sided error. Observe that acceptance by probabilistic machines with one-sided error on yes-instances is never stronger (and, in fact, is weaker in standard complexity classes) than acceptance by non-deterministic machines. In our setting, of finite automata, both models coincide with deterministic machines.

The power of two-sided error. In contrast to the above, we will show that a probabilistic machine with two-sided error probability can accept non-regular sets, and thus this model is more powerful than non-deterministic machines. Specifically, we will show a probabilistic machine with two-sided error probability that accepts the set $\{w \in \{0,1\}^* : \#_0(w) < \#_1(w)\}$, where $\#_\sigma(w)$ denotes the number of σ's in w.

The basic idea is to let the machine scan the input while tossing a coin per each 0-symbol it sees, and maintain a record of whether all these coin tosses turned out to be `head`. (This record can be encoded in the machine's state.) Similarly (and in parallel), while scanning the input, the machine tosses a coin per each 1-symbol it sees, and maintain a record of whether all these coin tosses turned out to be `head`. (The latter fact is recorded in a separate part of the state.)

We say a `0-win` occurred if all coins tossed for 0-symbols turned out to be `head`. Similarly, we define the notion of a 1-win. Note that it may be that we have both a 0-win and a 1-win or neither a 0-win nor a 1-win (the latter is most likely for most sufficiently long inputs). The probability that there is 0-win (resp., 1-win) on an input x is exactly $2^{-\#_0(x)}$ (resp., $2^{-\#_1(x)}$). Thus, if $\#_0(x) < \#_1(x)$ then the probability of a 0-win smaller by a factor of at least two than the probability of a 1-win, whereas if $\#_0(x) \geq \#_1(x)$ then these the probability of a 0-win is greater or equal to the probability of a 1-win. This motivates the following procedure.

1. We run the procedure described above, while recording whether a 0-win and/or a 1-win has occurred.

2. If either no win has occurred or both wins have occurred then we accept with probability exactly 1/2.

3. Otherwise (i.e., exactly one win has occurred) then we decide as follows: If a 0-win has occured then we accept else (i.e., a 1-win has occurred) we reject.

Note that on input x, we reach Step 3 with probability exactly

$$\mu(x) \stackrel{\rm def}{=} \left(2^{-\#_0(x)} \cdot (1 - 2^{-\#_1(x)}) + 2^{-\#_1(x)} \cdot (1 - 2^{-\#_0(x)})\right)$$

which may be exponentially vanishing with $|x|$. Still conditioned on reaching Step 3, we accept with probability

$$\begin{aligned}\Pr[\,\text{0-win has occured} \,|\, \text{a single win has occured}\,] &= \frac{2^{-\#_0(x)}}{2^{-\#_0(x)} + 2^{-\#_1(x)}} \\ &= \frac{1}{1 + 2^{\#_0(x) - \#_1(x)}}\end{aligned}$$

(Verification of the first equality is left as an exercise.) Let us denote $\Delta(x) \stackrel{\rm def}{=} \#_0(x) - \#_1(x)$. Thus, if $\#_0(x) < \#_1(x)$ (i.e., $\Delta(x) \leq -1$) then, conditioned on reaching Step 3, we accept with probability $1/(1+2^{\Delta(x)}) \geq 1/(1+0.5) = 2/3$. On the other hand, if $\#_0(x) \geq \#_1(x)$ (i.e., $\Delta(x) \geq 0$) then, conditioned on reaching Step 3, we accept with probability $1/(1+2^{\Delta(x)}) \leq 1/(1+1) = 1/2$. It follows that if $\#_0(x) < \#_1(x)$ (resp., $\#_0(x) \geq \#_1(x)$) then the above finite automaton accepts with probability at least $(1-\mu(x)) \cdot \frac{1}{2} + \mu(x) \cdot \frac{2}{3} = \frac{1}{2} + \exp(-|x|)$ (resp., at most $(1-\mu(x)) \cdot \frac{1}{2} + \mu(x) \cdot \frac{1}{2} = \frac{1}{2}$).

We conclude that x's in the langauge are accepted with probability strictly greater than 1/2, whereas x's not in the language are rejected with probability at least 1/2. This almost meets the definition of two-sided probabilistic acceptance. All that is needed is to "shift the acceptance probabilities" a little. Towards doing so, observe that x's in the langauge are actually accepted with probability at least $(1-\mu(x))\cdot\frac{1}{2}+\mu(x)\cdot\frac{2}{3}=\frac{1}{2}+\frac{\mu(x)}{6}$, where $\mu(x)>2^{-\#_0(x)}\cdot(1-2^{-\#_1(x)})>2^{-|x|/2}$. Thus, if unconditionally decrease the acceptance probability of each x by $2^{-|x|}$ (or so) then we'll be fine. This can be achieved by performing the above process in parallel to tossing a coin per each input bit (regardless of its value), and rejecting if all the latter coins turned out HEAD.

A more complex example

The above example of a non-regular set that is accepted by a finite automaton was suggested in class by Amos Gilboa, after I have presented the following more complex example. Specifically, I showed a probabilistic machine with two-sided error probability that accepts the (non-regular) set $\{w\in\{0,1\}^*:\#_0(w)=\#_1(w)\}$.

Following the above discussion, observe that if $\#_0(x)=\#_1(x)$ then the probability of a 0-win equals the probability of a 1-win, whereas if $\#_0(x)\neq\#_1(x)$ then these probabilities are at least a factor of 2 away from one another. Thus, to decide membership in the language we should approximate both probabilities, or rather the ratio between them. This cannot be done by running the above procedure, which provides us (in case we reach Step 3) with the result of one lottery with odds $2^{-\#_0(x)}:2^{-\#_1(x)}$ for 0-win versus 1-win. In order to approximate the odds (or rather distinguish the 50:50 case from the other cases), we need to obtain several results of the same lottery. This motivates the following procedure.

1. We run 1000 (parallel) copies of the basic procedure (described above), where each copy records whether a 0-win and/or a 1-win has occurred.
2. If in at least one of these 1000 copies either no win has occurred or both wins have occurred then we accept with probability exactly 1/2.
3. Otherwise (i.e., in each of the 1000 copies exactly one win has occurred) then we decide as follows: If the number of 0-wins is between 400 and 600 then we accept else we reject.

Note that on input x, we reach Step 3 with probability exactly

$$\mu(x)\stackrel{\rm def}{=}\left(2^{-\#_0(x)}\cdot(1-2^{-\#_1(x)})+2^{-\#_1(x)}\cdot(1-2^{-\#_0(x)})\right)^{1000}$$

which may be exponentially vanishing with $|x|$. Still conditioned on reaching Step 3, if $\#_0(x)=\#_1(x)$ then we accept with high probability (e.g., higher than 2/3). On the other hand, if $\#_0(x)\neq\#_1(x)$ then in Step 3 we accept with low probability (e.g., lower than 1/3). Proving the last two statements is left as an exercise. It follows that if $\#_0(x)=\#_1(x)$ (resp., $\#_0(x)\neq\#_1(x)$) then the above finite automaton accepts with probability at least $(1-\mu(x))\cdot\frac{1}{2}+\mu(x)\cdot\frac{2}{3}>\frac{1}{2}+\exp(-|x|)$ (resp., at most $(1-\mu(x))\cdot\frac{1}{2}+\mu(x)\cdot\frac{1}{3}<\frac{1}{2}-\exp(-|x|)$).

Exercise. Present a probabilistic machine with two-sided error probability that accepts the set $\{0^n1^n:n\in\mathsf{N}\}$. Same for $\{0^{an}1^{bn+c}:n\in\mathsf{N}\}$, where a,b and c are fixed (positive) integers. Finally, show that the restriction on (a,b and) c being positive can be removed.