The Modeling Chasm: Counting Turing Machines using Occam's razor

Pieter Adriaans[1]

FNWI
University of Amsterdam,
Science Park 107
1098 XG Amsterdam,
The Netherlands. P.W.Adriaans@uva.nl

Abstract.

keywords: Two-part code optimization, facticity, Occam's Razor, Turing Machines, Kolmogorov complexity, randomness, compressibility.

1 Introduction

A number of authors have suggested that the amount of model information in a binary string can be measured as the length of the optimal model code under two-part optimization ([3], [6], [11], [2], [1]). Two-part code optimization can be seen as a variant of the Minimum Description Length principle (MDL [8]) where one defines the optimal computational model for a data set as the Turing Machine (TM) that 1) produces the data set and 2) minimizes the sum of its description (the Model code) and the length of its input (the Data to Model code). These attempts have been marginally successful due to the following problems:

1. It is unclear how and why machine indexes and input code should be balanced. Can one balance indexes of computational models with binary input strings on a bit by bit basis? There are signs that MDL does not perform optimally in many circumstances [7], which gives cause for concern.
2. Unlike the traditional theory of Kolmogorov complexity, where the choice of a reference universal Turing machine evens out asymptotically [12], any theory that balances the length of indexes of Turing machines with the length of Data to Model code appears to be extremely sensitive to the choice of a reference Turing machine. An extreme example is the so-called Nickname problem "(...) in which a particular TM which is constructed to contain the string x in its state description is given a low index, and thus achieves the minimal two-part coding" [5].

In this paper enumerations of Turing machine satisfying, what is called, Occam's condition are studied: machines are ordered according to their number of states. This facilitates the analysis of asymptotic behavior of classes of Turing machines and their indexes [9]. It is shown that under Occam's condition

It introduces facticity as a formal method to measure the amount of model information in a string. It is shown that the adoption The non-robustness, which with hindsight, is more a feature than a bug will be treated in a future publication. It is shown that

The fact that all Turing machines can be enumerated is essential in Alan Turing's proof of the existence of uncomputable numbers. Ever since, in computer science the length of an index of a Turing machine is commonly used as a measure for its complexity. Also it is commonly assumed that a string can be produced by a machine with a comparable index length. One way to order Turing machines is by applying Occam's razor: machines with less states get lower numbers. Such an 'Occam condition' seems plausible: machines with fewer states are in a way less complex. One should only introduce new states when necessary. Thus Occam's razor automatically forces a kind of parsimony when selecting models. In this paper it is shown that this use of indexes, even when applying Occam's razor, is far from harmless. Under Occam's principle the chasm between the length of a string and its computational model can in the limit be arbitrary large.

2 Definitions

Let $x, y, z \in \mathbb{N}$, where $\mathbb{N}$ denotes the natural numbers and we identify $\mathbb{N}$ and $\{0,1\}^*$ according to the correspondence

$$(0, \varepsilon), (1, 0), (2, 1), (3, 00), (4, 01), \ldots$$

Here ε denotes the *empty word*. The *length* $|x|$ of x is the number of bits in the binary string x, not to be confused with the *cardinality* $|S|$ of a finite set S.

2.1 One-part code complexity

The standard reference [12] for the definitions concerning Kolmogorov complexity is followed. K is the prefix-free Kolmogorov complexity of a binary string. It is defined as:

Definition 1. $K(x|y) = \min_i \{|\bar{i}| : U(\bar{i}y) = x\}$

i.e. the shortest self-delimiting index of a Turing machine T_i that produces x on input y, where $i \in \{1, 2, ...\}$ and $y \in \{0,1\}^*$. Here $|\bar{i}|$ is the length of a self-delimiting code of an index and U is a universal Turing machine that runs program y after interpreting $\bar{i}$. The length of $|\bar{i}|$ is limited for practical purposes by $n + 2\log n + 1$, where $n = |i|$. The actual Kolmogorov complexity of a string is defined as the one-part code that is the result of, what one could call, the:

Definition 2 (Forcing operation). $K(x) = K(x|\varepsilon)$

Here all the compressed information, model as well as noise, is forced on to the model part.

2.2 Balanced two-part code complexity

The reason to use $\bar{i}$ in $U(\bar{i}y) = x$ lies in the fact that it allows us to separate the concatenation $\bar{i}p$ into its constituent parts, i and p. Here i is the index of a Turing machine which can be seen as capturing the *regular* part of the string x, and p describes the input for the machine, i.e. the *irregular* part, e.g. errors in the model, noise and other missing information. Based on this insight one can define a balanced two-part code complexity. The length of the regular part is defined as the:

Definition 3 (Facticity). $\varphi(x) = \min_{(i,j)}\{|i| : U(\bar{i}j) = x\}$

The length of the irregular part is defined as the:

Definition 4 (Residual entropy). $\rho(x) = \min_{(i,j)}\{|j| : U(\bar{i}j) = x\}$

Note that these definitions suppose that the term $\bar{i}j$ is minimized, i.e. there is an optimal separation between model information and noise.

2.3 Turing machines

The following definitions can be found in [10]. A *Turing machine* (TM) is described by a 7-tuple

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

Here, as usual, Q is the finite set of states, Σ is the finite set of input symbols with $\Sigma \subset \Gamma$, where Γ is the complete set of tape symbols, δ is a transition function such that $\delta(q, X) = (p, Y, D)$, if it is defined, where $p \in Q$ is the next state, $X \in \Gamma$ is the symbol read in the cell being scanned, $Y \in \Gamma$ is the symbol written in the cell being scanned, $D \in \{L, R\}$ is the direction of the move, either left or right, $q_0 \in Q$ is the start state, $B \in \Gamma - \Sigma$ is the blank default symbol on the tape and $F \subset Q$ is the set of accepting states. A *move* of a TM is determined by the current content of the cell that is scanned and the state the machine is in. It consists of three parts:

1. Change state
2. Write a tape symbol in the current cell
3. Move the read-write head to the tape cell on the left or right

3 Counting Turing machines

The class of Turing machines is countable. A specific machine can be referenced by means of a natural number. There are many ways to count Turing machines but in the following it will only be assumed that our ordering of machines observes the number of states:

Definition 5 (Occam's Condition). *For each $i \in \mathbb{N}$ the tuple*

$$M_i = (Q_i, \Sigma_i, \Gamma_i, \delta_i, q_{0i}, B_i, F_j)$$

is the Turing machine with index i. Occam's condition is observed when it is the case that $|Q_j| > |Q_i|$ for all M_j such that $j > i$.

Here $|Q_j|$ is the cardinality of the set Q_j. This can be seen as a variant of Occam's razor: a preference for machines with small indexes is a preference for machines with fewer states. In the following machines that use binary input code $\{0, 1\}$ will be studied. They have binary strings as indexes using the standard mapping from natural numbers to strings. The following theorem gives an estimate lower bounds for the length of the indexes:

Theorem 1. *Under Occam's condition a deterministic Turing machine with $k > 2$ states has an index of at least 7.8 bits and this length grows super-linearly with a factor of at least $(k-1)\log_2(k-1)$.*

Proof: The amount δ-relations is used to estimate lower bounds for the complexity of Turing machines with k states. Let

$$I = \texttt{\#input-states} \times \texttt{\#input-symbols} = k \times 3$$

be the state symbol-complexity for k states and the O the possible set of moves per state-symbol combination, then O^I is the amount of possible delta-functions. For deterministic Turing machines this amounts to:

$$O = \texttt{\#output-states} \times \texttt{\#output-symbols} \times \texttt{\#left-right-moves} = k \times 3 \times 2$$

This gives $(6k)^{3k}$ as a lower bound for the amount of machines with k states. Because of Occam's condition in definition 5 a machine with k states has at least an index after number $\sum_{i=1}^{k-1}(6i)^{3i} > (6(k-1))^{3(k-1)}$. A binary index for such an index had at least the following length:

$$\log_2(6(k-1))^{3(k-1)} = 3(k-1)(\log_2 6 + \log_2(k-1)) >$$

$$(k-1)\log_2(k-1)$$

Note that already for small values of k the index is blown up by a factor of at least $3\log_2 6 \approx 7.8(k-1)$. □

This analysis triggers a number of observations.

Observation 1: under Occam's condition there are exactly 216 one-state Turing machines of the form:

$$M_i = (\{q_0\}, \{0, 1\}, \{0, 1, b\}, \delta_i, q_0, \{b\}, \{q_0\})$$

for which the start state and the accepting state are identical. Since they all start in the accepting state they never make a move so their observable behavior is identical. They differ only in the definition of the vacuous δ-function, of which there are:

$$(\texttt{\#output-symbols} \times \texttt{\#left-right-moves})^{\texttt{\#input-symbols}} = (3 \times 2)^3 = 216$$

different versions. Since the behavior of all these 216 machines is equivalent, an arbitrary machine can be chosen to be the first:

Definition 6. $M_1 = (\{q_0\}, \{0, 1\}, \{0, 1, b\}, \delta_1, q_0, \{b\}, \{q_0\})$ *is the first Turing Machine.*

Under any counting scheme that observes Occam's condition the first deterministic machine with two states is M_{217}.

Observation 2: Each binary string x of length k has a set of sequential print machines that simply write down the string:

Definition 7 (Sequential print machine). *Every string x can be generated by each member of a set of machines M_x with the following structure: M_x has $k + 1$ states and the delta function contains the following operations (only the relevant part of the function is given):*

$(\delta(q_s, b) = (q_2, x_1, r),$
$(...),$
$\delta(q_i, b) = (q_{i+1}, x_i, r),$
$(...),$
$\delta(q_k, b) = (q_a, x_k, r))$

i.e. M_x starts at the start state, at move i it is in state i, writes the i-th symbol x_i of x and goes in to state $i+1$, until it reaches the accepting state. The sequential print machines for string x have a state transition to a new state for every symbol in x.

Observe that these machines are maximal in the sense that that are no strings for which which automata with more states are necessary to produce them deterministically.

4 The Modeling Chasm

A simple counting argument proves that there are incompressible string for each length k. There are k^2 strings of length k and $k^2 - 1$ strings of length $k - 1$ or less. So there is at least one string x of length k for which there is no smaller program available. It is generally assumed that incompressibility and randomness are related in the sense that incompressible string have no regularities that can be exploited to find a shorter description [4], [12]. It is also assumed that any incompressible string x can be produced by a simple program of the form `PRINT("x")` with minimal overhead, i.e. a simple print program is the most efficient way to produce a random string. If this were the case then the following conjecture would be true:

Conjecture 1. With high probability a sequential print machine M_x generating a random string x is minimal in the sense that there are no Turing machines with fewer states producing x.

A proof of this conjecture would follow from another conjecture, implied in [9], that under Occam's condition for the class of Turing machines with one two-way infinite tape the density of the Halting set in the limit is 0. This can be proved for machines with one-way infinite tape. For our case the status of this conjecture is unclear. It is true, it has some non-trivial consequences. Amongst other things it implies that under Occam's condition the length of indexes of deterministic machines that produce random strings grows with a function that is super linear in the length of these strings:

Lemma 1. *Assuming Occam's principle and conjecture 1 there is no constant c such that for all strings x it is the case that $c|x| > |i_x|$, where T_{i_x} is the deterministic machine with the shortest index i_x that computes x on empty input.*

Proof: This is an immediate consequence of theorem 1. According to Conjecture 1 there are for every k strings x of length k for which the most efficient machine M_x is the sequential print machine, which is maximal and thus has $k+1$ states. According to theorem 1 the index length for such machines grows with a factor $k \log_2 k$ and thus cannot be bounded by a linear function ck.□

Under Conjecture 1 deterministic machines are very inefficient devices for storage of information: at least one state for each bit of information is needed. All a deterministic computational process can do to emulate a system of messages with maximal entropy is model each message explicitly.

The modeling chasm form the title of this paper is most apparent in the effects of the forcing operation of definition 2. Suppose x is a random string of length k. There is no input $\bar{i}y$ for a universal Turing machine U for which $U(\bar{i}y) = x$ with $|\bar{i}y| < |x|$. Under Occam's condition, the best model available is our first machine M_1, that 'does nothing', with a one bit code length. Given the formula $n + 2\log n + 1$ this only adds two bits to the code length. The best estimate is $|\bar{i}y| = |x| + 2$ with $\varphi(x) = 1$ and $\rho(x) = |y|$. Now, enter the forcing operation $K(x) = K(x|\varepsilon)$, that forces us to model x in terms of a deterministic computational process with empty input. Since x is random, under Conjecture 1, with high probability its minimal automaton is also maximal and by lemma 1 its index length is not bound by function linear in k. The chasm between $\varphi(x)$, the length of the model code under balanced two-part code, and $K(x)$, the forced one-part code complexity, is not bounded by any linear function in the limit.

The harm is even more apparent when a highly compressible string and a random string are concatenated. Assuming Conjecture 1 there are highly compressible strings that become 'incompressible' after the forcing operation: suppose x generated by $U(\bar{i}y) = x$ where i is the index of a machine that scans the input and then adds a string of l bits of 1-s and y is a random string of k bits. The length of this program is $|\bar{i}y| = c + \log_2 l + k$ where $|x| = k + l$, $\varphi(x) = \log l + c$ and $\rho(x) = |y|$. So x is compressible if the following holds:

$$c + \log_2 l < l \qquad (1)$$

Under the forcing operation j is the index of a new machine $U(\overline{j}\varepsilon) = x$ that produces x. According to the reasoning above, $|j|$ is at least $c + \log_2 l + k \log_2 k$ bits long, so x incompressible when $k + l < c + \log l + k \log_2 k$ or

$$k(\log_2 k - 1) > l - \log_2 l - c \quad (2)$$

It is easy to find values for l and k that satisfy both equations. For these values the corresponding string x is compressible before the forcing operation and incompressible afterwards.

Any theory using indexes satisfying Occam's constraint and the forcing operation will under conjecture 1 in the limit be confronted with an infinite chasm between the length of most strings and their shortest computational models. In such a context randomness and compressibility are two very distinct notions.

If conjecture 1 is not true the results are also non-trivial. In that case the intuition that the the most efficient way to generate random strings is the use of the print function is not true: every random string x longer than some constant c will have a complex program j with $|x| < |j| < |i|$, where i is the index of the sequential printing machine with $k \log_2 k < |i|$. One last possibility is the other extreme, implied by the standard interpretation of Kolmogorov complexity. i.e. that every random string it generated by a program of comparable length.

Lemma 2. *Under Occam's principle there is no constant $O(1)$ such that for every string x it is the case that $K(x) < |x| + O(1)$*

Suppose $\{0, 1\}^{\leq k}$ is the set of $2^{k+1} - 1$ binary strings of length $\leq k$. Suppose each string in this set is generated by a machine with index length of at most k bits. There $2^{k+1} - 1$ of these machines, so at best every string has exactly one machine. According to theorem 1 such a machine can have at most l states, where $l \log_2 l = k$, i.e. $l = \frac{k \log 2}{W(k \log 2)}$, where W is the product log function. But given the density of the Halting set a considerable part of these machines will not stop (at least 13% in the limit by a simple argument [9]). So some strings will by necessity be produced by an automaton with $l + 1$ states, with an index of at least $(l + 1) \log_2(l + 1) = (l \log_2(l + 1)) + \log_2(l + 1 > k + \log(l + 1)$ where $\log_2 l = \log_2 \frac{k \log 2}{W(k \log 2)}$ is growing linearly with $\log_2 k$ $\square$.

5 Discussion

The result of this analysis is that deterministic computational models are rather expensive. Moreover, the overwhelming majority of these models have no practical use. This insight does not affect the theory of Kolmogorov complexity itself: random strings stay random, compressible strings stay compressible. It destroys the intuition that computational models for random strings always have a complexity 'close' to the length of these strings. This might have consequences for all theories that try to balance length of data sets with their possible computational models: Kolmogorov structure function [13], facticity [1], effective complexity [6] and forms of MDL [8]. If models for random strings can be arbitrary longer than these strings, then it is natural to assume that compressible strings can have useful computational models that are longer than these data sets. This contradicts the intuition behind standard application of MDL and might explain why this principle performs sub-optimally in some conditions [7]. In order to analyze this situation properly we also need to take into account enumerations of non-deterministic Turing machines and this will be the subject of a future paper.

6 Acknowledgements

This research was partly supported by the Info-Metrics Institute of the American University in Washington, the Commit project and the ILLC and IvI of the University of Amsterdam.

Bibliography

[1] Adriaans , P.W. , (2009) Between Order and Chaos: The Quest for Meaningful Information, Theory of Computing Systems, Volume 45 , Issue 4 (July 2009), Special Issue: Computation and Logic in the Real World; Guest Editors: S. Barry Cooper, Elvira Mayordomo and Andrea Sorbi, 650-674.

[2] L. Antunes, L. Fortnow. D. Van Melkebeek and N. V. Vinodch, (2006) Computational depth: Concept and application, Theoretical Computer Science, volume, 354.

[3] C. H. Bennett, (1988) Logical depth and physical complexity. In R. Herken, editor, The Universal Turing Machine: A Half-Century Survey, pages 227-257. Oxford University Press.

[4] Cover T.M. and Thomas, J.A. (2006), Elements of Information theory, Wiley.

[5] Foley, D.K. and Oliver D., Notes on facticity and effective complexity, http://www.american.edu/cas/economics/info-metrics/pdf/upload/Oct-2011-Workshop-Paper-Foley-and-Oliver.pdf (retrieved 19-01-2014).

[6] Gell-Mann M. and S. Lloyd (2003) Effective complexity. In Murray Gell-Mann and Constantino Tsallis, eds.

[7] P.D. Grünwald P.D. and Langford J., Suboptimal behaviour of Bayes and MDL in classification under misspecification, COLT 2004

[8] Grünwald, P.D. (2007), The Minimum Description Length Principle. MIT Press.

[9] Hamkins, Joel David; Miasnikov, Alexei The halting problem is decidable on a set of asymptotic probability one. Notre Dame J. Formal Logic 47 (2006), no. 4, 515524.

[10] J.E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman, Introduction to Automata Theory, Languages, and Computation Second Edition. Addison-Wesley, 2001.

[11] P.M.B. Vitányi, (2006) Meaningful information, IEEE Trans. Inform. 52:10, 4617 - 4626.

[12] Li M., Vitányi P.M.B. (2008), An Introduction to Kolmogorov Complexity and Its Applications, 3rd ed., Springer-Verlag, New York.

[13] Vereshchagin N.K., Vitányi P.M.B., Kolmogorov's structure functions and model selection, IEEE Trans. Information Theory, vol. 50, nr. 12, 3265–3290, (2004)