Toward tractable universal induction through recursive program learning

Arthur Franz

Independent researcher*

Abstract. Since universal induction is a central topic in artificial general intelligence (AGI), it is argued that compressing all sequences up to a complexity threshold should be the main thrust of AGI research. A measure for partial progress in AGI is suggested along these lines. By exhaustively executing all two and three state Turing machines a benchmark for low-complexity universal induction is constructed. Given the resulting binary sequences, programs are induced by recursively constructing a network of functions. The construction is guided by a breadth-first search departing only from leaves of the lowest entropy programs, making the detection of low entropy ("short") programs efficient. This way, all sequences (80% of the sequences) generated by two (three) state machines could be compressed back roughly to the size defined by their Kolmogorov complexity.

1 Introduction

What is intelligence? After compiling a large set of definitions in the literature Legg and Hutter [8] came up with a definition of intelligence that is consistent with most other definitions:

"Intelligence measures an agent's ability to achieve goals in a wide range of environments."

Based on that definition Marcus Hutter [5] has developed a mathematical formulation and theoretical solution to the universal AGI problem, called AIXI. Although it is not computable, approximations may lead to tractable solutions. AIXI is in turn essentially based on Solomonoff's theory of universal induction [15], that assigns the following universal prior to any sequence x:

$$M(x) := \sum_{p:U(p)=x*} 2^{-l(p)} \tag{1.1}$$

where p is a program of length $l(p)$ executed on a universal monotone Turing machine U. $U(p) = x*$ denotes that after executing program p, the machine U prints the sequence x without necessarily halting. Impressively, it can be shown [5] that after seeing the first t digits of any computable sequence this universal prior is able to predict the next digit with a probability converging to certainty:

* e-mail: franz@fias.uni-frankfurt.de

$\lim_{t\to\infty} M(x_t|x_1,\ldots,x_{t-1}) = 1$. Since most probability weight is assigned to short programs (Occam's razor) this proves that compressed representations lead to successful predictions of any computable environment. This realization makes it especially promising to try to construct an efficient algorithm for universal induction as a milestone, even cornerstone, of AGI.

A general but brute force approach is universal search. For example, Levin search [10] executes all possible programs, starting with the shortest, until one of them generates the required data sequence. Although general, it is not surprising that the approach is computationally costly and rarely applicable in practice.

On the other side of the spectrum, there are non-general but computationally tractable approaches. Specifically, inductive programming techniques are used to induce programs from data [6] and there are some approaches within the context of AGI as well [16, 14, 12, 3]. However, the reason why the generalization of many algorithms is impeded is the curse of dimensionality faced by all algorithms at some point. Considering the (algorithmic) complexity and diversity of tasks solved by today's typical algorithms, we observe that most if not all will be highly specific and many will be able to solve quite complex tasks (known as "narrow AI" [7]). Algorithms from the field of data compression are no exception. For example, the celebrated Lempel-Ziv compression algorithm (see e.g. [2]) handles stationary sequences but fails at compressing simple but non-stationary sequences efficiently. AI algorithms undoubtedly exhibit some intelligence, but when comparing them to humans, a striking difference comes to mind: the tasks solvable by humans seem to be much less complex albeit very diverse, while tasks solved by AI algorithms tend to be quite complex but narrowly defined (Fig. 1.1).

For this reason, we should not try to beat the curse of dimensionality mercilessly awaiting us at high complexities, but instead head for general algorithms at low complexity levels and fill the task cup from the bottom up.

2 A Measure for Partial Progress in AGI

One of the troubles of AGI research is the lack of a measure for partial progress. While the Turing test is widely accepted as a test for general intelligence, it is only able to give an all or none signal. In spite of all attempts, we did not yet have a way to tell whether we are half way or 10% through toward general intelligence. The reason for that disorientation is the fact that every algorithm having achieved partially intelligent behavior, has failed to generalize to a wider range of behaviors. Therefore, it is hard to tell whether research has progressed in the right direction or has been on the wrong track all along.

However, since making universal induction tractable seems to be a cornerstone for AGI, we can formalize partial progress toward AGI as the extent to which universal induction has been efficiently implemented. Additionally, if we start out with a *provably general* algorithm that works up to a complexity level, thereby solving all simple compression problems, the objection about its possible

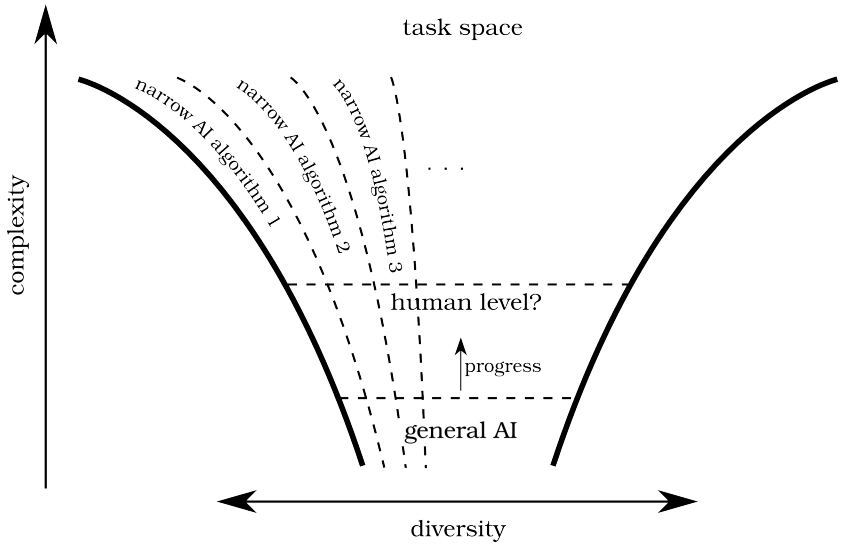

Fig. 1.1. Approach to artificial general intelligence. Instead of trying to solve complex but narrow tasks, AGI research should head for solving all simple tasks and only then expand toward more complexity.

non-generalizability is countered. The measure for partial progress then simply becomes the complexity level up to which the algorithm can solve all problems.

2.1 Related Work

This measure is reminiscent of existing intelligence tests based on algorithmic complexity. Hernandez-Orallo [4] has developed the C-test, that allows only sequences with unique induced explanations, of which a prefix leads to the same explanation and various other restrictions on the sequence set. However, since the pitfall of building yet another narrow AI system is lurking at every step, a measure of research progress in AGI (not so much of the intelligence of an agent) should make sure that *all* sequences below a complexity level are compressed successfully and can not afford to discard large subsets as is done in the C-test.

Legg and Veness [9] developed a measure that takes into account the performance of an agent in a reinforcement learning setting which includes an Occam bias decreasing exponentially with the complexity of the environment. They are correct to note that the solution to the important exploration-exploitation dilemma is neglected in a purely complexity-based measure. In that sense, universal induction is a necessary albeit not sufficient condition for intelligence. For our purposes, it is important to set up a measure for universal induction alone, as it seems to be a simpler problem than one of building complete intelligent agents.

Text based measures of intelligence follow the rationale that an agent can be considered intelligent if it is able to compress the information content of a text, like humanity's knowledge in the form of Wikipedia [1, 13]. However, this kind of compression requires large amounts of information not present in the text itself, like real world experience through the agent's senses. Therefore, the task is either ill-defined for agents not disposing of such external information or the agent has to be provided with such information extending texts to arbitrary data, which is equivalent to the compression of arbitrary sequences as proposed here.

2.2 Formalization

Suppose, we run binary programs on a universal monotone Turing machine U. U's possible input programs p_i can be ordered in a length-increasing lexicographic way: "" (empty program), "0", "1", "00", "01", "10", "11", "000", etc. up to a maximal complexity level L. We run all those programs until they halt or for a maximum of t time steps and read off their outputs x_i on the output tape. In contrast to Kolmogorov complexity[1], we use the time-bounded version – the Levin complexity – which is computable and includes a penalty term on computation time [11]:

$$Kt(x) = \min_p\{|p| + \log t : U(p) = x \text{ in } t \text{ steps}\} \tag{2.1}$$

Saving all the generated strings paired with their optimal programs (x_i, p_i^o) with $p_i^o(x_i) = \text{argmin}_p\{|p| + \log t : U(p) = x_i \text{ in } t \text{ steps}, \ |p| \leq L\}$, we have all we need for the progress measure. The goal of universal induction is to find all such optimal programs p_i^o for each of the x_i. If p_i is the actually found program, its performance can be measured by

$$r_i(L) = \frac{|p_i^o|}{|p_i|} \in (0, 1] \tag{2.2}$$

If not, there is no time-bounded solution to the compression problem. The overall performance R at complexity level L could be used as a measure for partial progress in universal induction and be given by averaging:

$$R(L) = \langle r_i(L) \rangle \tag{2.3}$$

One may object that the number of programs increases exponentially with their length such that an enumeration quickly becomes intractable. This is a weighty argument if the task is universal search – a general procedure for inversion problems. However, we suggest this procedure to play the mere role of a benchmark for an *efficient* universal induction algorithm, which will use completely different methods than universal search and will be described in Section

[1] The Kolmogorov complexity of a string is defined as the length of the shortest program able to generate that string on a Turing machine.

3. Therefore, using the set of simple programs as a benchmark may be enough to set the universal induction algorithm on the right track.

Note that only a small fraction of possible sequences can be generated this way. After all, it is well known that only exponentially few, $O(2^{n-m})$, sequences of length n can be compressed by m bits [11].

2.3 Implementation

Implementing this test does not require coding of a universal Turing machine (TM) since computers are already universal TMs. Instead, enumerating all transition functions of an n-state machine is sufficient. The machine used here has one bidirectional, two way infinite work tape and a unidirectional, one way infinite, write only output tape. Two symbols are used, $\mathbb{B} = \{0, 1\}$, the states taken from $Q = \{0, \ldots, n-1\}$. The transition map is then:

$$Q \times \mathbb{B} \to Q \times \{0, 1, L, R, N\} \times \{0, 1, N\} \tag{2.4}$$

where L, R, and N denote left, right and no motion of the head, respectively. The work tape can move in any direction while the output tape either writes 0 or 1 and moves to the right, or does not move at all (N). No halting or accepting states were utilized. The machine starts with both tapes filled with zeros. A finite sequence x is considered as generated by machine T given transition function (program) p, if it is at the left of the output head at some point: we write $T(p) = x*$. The transition table enumerated all possible combinations of state and work tape content, which amounts to $|Q| \cdot |\mathbb{B}| = 2n$. Therefore, there exist $|Q| \cdot 5 \cdot 3 = 15n$ different instructions and consequently $(15n)^{2n}$ different machines with n states. For $n = 2, 3$ this amounts to around 10^6 and 10^{10} machines. All those machines ($n = 1$ machines are trivial) were executed until 50 symbols were written on the output tape or the maximum number of 400 time steps was reached. All unique outputs were stored, amounting to 210 and 43295, for $n = 2, 3$, respectively, and paired with their respective programs.

Table 1 depicts a small sample of the outputs. It may be interjected that sequences generated by 2 and 3 state machines are not very "interesting". However, the present work is the just initial step. Moreover, it is interesting to note that even the 2 state machine shows non-repetitive patterns with an ever increasing number of 1's. In the 3 state machine patterns become quickly more involved

states	sample outputs
2	10101010101010101010101010101010101010101010101010
2	11011011011011011011011011011011011011011011011011
2	00010011001110011110011111001111110011111110011111
3	00101101001010001011010010100010110100101000101101
3	10111111001110111101001110101111010100111010101111
3	01011010110101110101101101011110101101101101011111

Table 1. Sample outputs of 2 and 3 state Turing machines

and require "intelligence" to detect the regularities in the patterns (try the last one!). Consistently with the reasoning in the introduction, could it be that the threshold complexity level of human intelligence is not far off from the sequence complexity of 3 state machines, especially when the data presentation is not comfortably tuned according to natural image statistics?

We suggest that these patterns paired with their respective programs constitute a benchmark for partial progress in artificial general intelligence. If an efficient algorithm can compress these patterns to small programs then it can be claimed to be moderately intelligent. Modern compression algorithms, such as Lempel-Ziv (on which the famous Zip compression is based), fail at compressing those sequences, since the packed file size increases with sequence length (ergo r_i gets arbitrary small) while the size of the TM transition table is always the same independently of sequence length.

3 Universal Induction of Low-Complexity Sequences

3.1 Methods

Having generated all strings printed by two and three state programs the task is to build an efficient algorithm compressing those strings back into a short representation, not necessarily the original one though, but having a similar size in terms of entropy.

As exemplified in Fig. 3.1 the present algorithm induces a recursive network of function primitives using a sequence generated by a three state Turing machine. Four function primitives were used that generate constant, alternating or incremental sequences or a single number:

$$C(s,n) = s, s, \ldots, s \,(n \text{ times}), \quad s \in \mathbb{Z} \cup \mathbb{S}, n \in \mathbb{N} \tag{3.1}$$

$$A(a,b,n) = a, b, a, b, \ldots a, b \,(n \text{ times}) \quad a, b \in \mathbb{Z} \cup \mathbb{S}, n \in \mathbb{N} \tag{3.2}$$

$$I(s,d,n) = s + 0 \cdot d, s + 1 \cdot d, \ldots, s + (n-1) \cdot d \quad s, d \in \mathbb{Z}, n \in \mathbb{N} \tag{3.3}$$

$$R(s) = s \quad s \in \mathbb{Z} \cup \mathbb{S} \tag{3.4}$$

where $\mathbb{Z}$ is the set of integers, $\mathbb{N}$ the set of non-negative integers and $\mathbb{S} = \{C, A, I, R\}$ is the set of arbitrary symbols (here function names).

The entropy of a given function network is computed as follows. Let $x_i \in \mathbb{Z} \cup \mathbb{S}$ denote the inputs to those functions without parent functions. The distribution $p(n) = 2^{-|n|}/3$ is imposed on integers $n \in \mathbb{Z}$. If $x_i \in \mathbb{Z}$ then its information content is given by $H(x_i) = -\log_2 p(x_i) = |x_i| + \log_2(3)$ bits[2] which we simplify to $|x_i| + 1$ bits. If $x_i \in \mathbb{S}$ then $H(x_i) = \log_2 |\mathbb{S}| = 2$ bits. The overall entropy of the network is the sum $H_{tot} = \sum_i H(x_i)$. It may be objected that according

[2] This coding is linear in the integer value. We could use Elias gamma or delta coding, which is logarithmic, however the algorithm has turned out to perform better with linear coding. This is work in progress and this issue shall be investigated in future work.

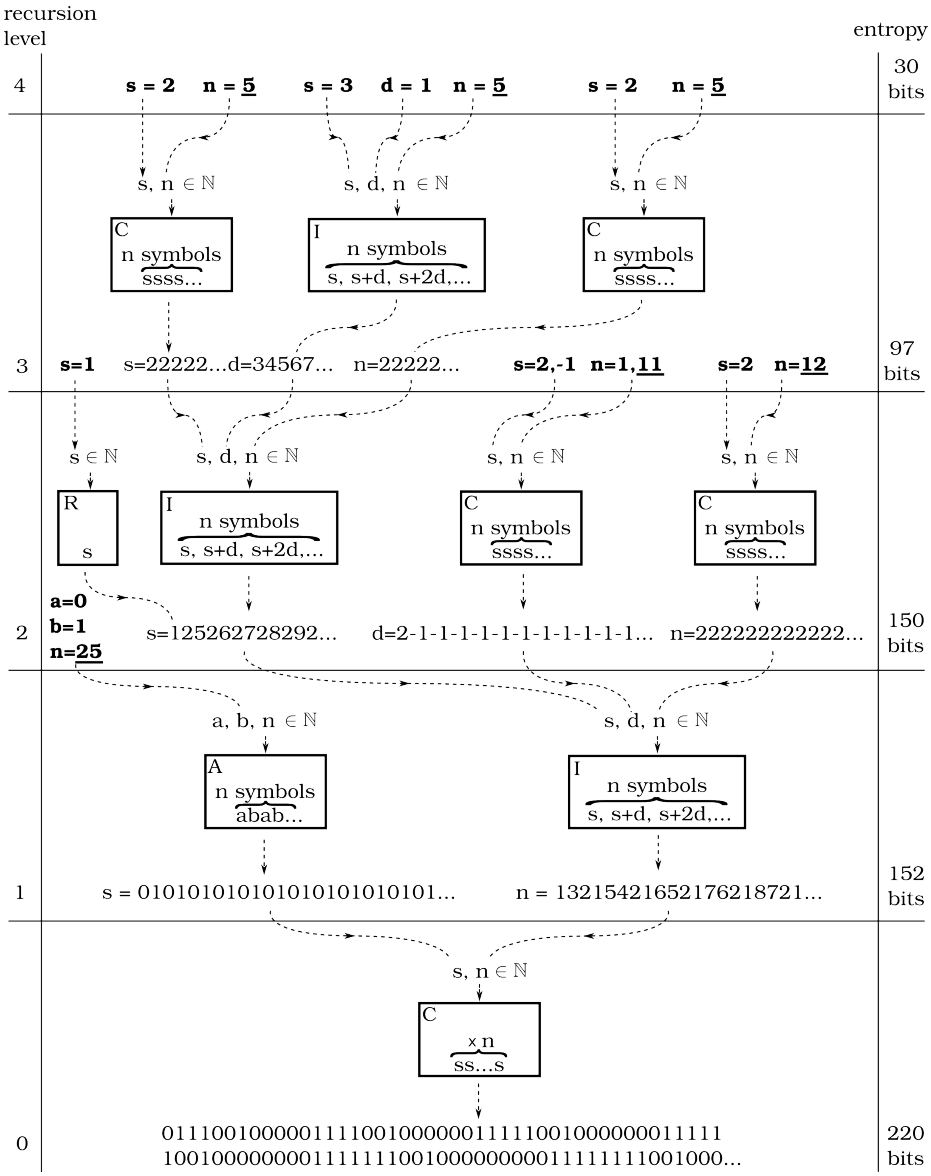

Fig. 3.1. Exemplifying recursive compression. A sequence is recursively transformed by a network of functions to an increasingly smaller representation. The original sequence takes up 220 bits of information, 129 bits for encoding the 0's and 1's plus the length of the sequence (91 bits). At the zeroth recursion level the sequence is parsed using a constant function (C) that prints n times the number s. At level 1 the sequences of function inputs are shown that recreate the original sequence. The original sequence is thereby transformed to two sequences of function inputs. Subsequently, an alternating function (A) explains the first sequence and an incremental function (I) explains the second one. This is done recursively, until the entropy can not be reduced any more. The bold inputs remain unexplained and amount to 96 bits. Note that the final number of inputs does not depend on the sequence length any more. If we remove those inputs that change with sequence length (bold and underlined) then the entropy decoding sequence structure is only 27 bits (only bold).

to the Minimum Description Length principle, the information contained in the algorithm itself has to be taken into account as well. After all, for any sequence x it is possible to define a universal Turing machine U' such that $Kt_{U'}(x) = 0$ thereby encoding all information about x in the design of U', making U' highly dependent on x. However, since both the present algorithm and the benchmark do not depend on x, their description length is a mere constant and can be neglected.

At each step of the algorithm a set of unexplained sequences is present, which are sequences of inputs to those functions without parent functions. For each such input sequence its entropy can be computed and the sequences ordered after decreasing entropy. Looping through that set starting with the sequence of highest entropy (requiring most explanation) the algorithm tries to generate a part of the sequence with one of the function primitives. For example, if the sequence $q = 3, 3, 3, 9, 9, 9, 9, 6, 6, 6, 6, 6$ is present, a sequence of inputs to the constant function is induced: $C(s = 3, 9, 6, n = 3, 4, 5)$. The entropy is reduced, in this case $H(q) = 87$ bits and its explanation takes only $H(s) + H(n) = 36$ bits. For each function primitive, such an entropy change is computed. If the entropy has been reduced, the function is accepted and added to the network. Otherwise, it is accepted only if its child (the function that receives its outputs) has been entropy reducing, allowing to overcome local minima in the entropy landscape to some extent.

In this fashion a breadth-first search is performed, while pruning away the least promising tree branches. Those are defined as programs having a higher total entropy than the 1.05 times the program with lowest entropy.[3]

3.2 Results

Since our fixed-size Turing machine programs can create sequences of arbitrary length, successful program induction is defined as induction of a program with a fixed number of inputs to the function network. Further, to establish a benchmark, the entropy of the Turing machine programs is computed as follows. There are $(15n)^{2n}$ machines with n states, hence the amount of information needed to specify a TM program with n states is

$$H_{\mathrm{TM}}(n) = 2n \log_2(15n) \tag{3.5}$$

which results in a program size of around 20 and 33 bits for two and three state TMs, respectively. Since the induced programs encode the length l of the target sequence and the TM programs do not, the information contained in the length has to be subtracted from the induced program entropy (the bold and underlined numbers in Fig. 3.1).

All sequences generated by all two state machines could be compressed successfully. The average induced program size is $\mu_2 = 22$ bits with a standard deviation of $\sigma_2 = 23$ bits. Because of the large number of three states sequences,

[3] Python code and string/program pairs are available upon request.

200 sequences were randomly sampled. This way, 80±4% of three state sequences could be compressed successfully, with $\mu_3 = 27$ bits and $\sigma_3 = 20$ bits. However, "unsuccessful" sequences could be compressed to some extent as well, although the resulting program size was not independent of sequence length. With sequences of length $l = 100$ the entropy statistics of "unsuccessful" sequences are $\mu'_3 = 112$ bits and $\sigma'_3 = 28$ bits. Given an average sequence entropy of 146 bits, this constitutes an average compression factor of 1.3.

It may seem surprising that the average entropy of the induced programs is even below the entropy of the TM programs (transition tables). However, since not all rows of a transition table are guaranteed to be used when executing a program, the actual shortest representation will not contain unused rows leading to a smaller program size than 20 or 33 bits. The most important result is that very short programs, with a size roughly around the Kolmogorov complexity, have indeed been found for most sequences.

4 Discussion

The present approach has shown that it is possible to both sensibly define a measure for partial progress toward AGI by measuring the complexity level up to which all sequences can be induced and to build an algorithm actually performing universal induction for most low complexity sequences. Our demonstrator has been able to compress all sequences generated by two state Turing machines and 80% of the sequences generated by three state Turing machines.

The current demonstrator presents work in progress and it is already fairly clear how to improve the algorithm such that the remaining 20% are also covered. For example, there is no unique partition of a sequence into a set of concatenated primitives. The way, those partitions are selected should also be guided by compressibility considerations, e.g. partition subsets of equal length should have a higher prior chance to be analyzed further. Currently, the partition is implemented in a non-principled way, which is one of the reasons for the algorithm to run into dead ends. Remarkably, all reasons for stagnation seem to be those aspects of the algorithm that are not yet guided by the compression principle. This observation leads to the conjecture that the further extension and generalization of the algorithm may not require any additional class of measures, but a "mere" persistent application of the compression principle.

One may object that the function primitives are hard-coded and may therefore constitute an obstacle for generalizability. However, those primitives can also be resolved into a combination of elementary operations, e.g. the incremental function can be constructed by adding a fixed number to the previous sequence element, hence be itself represented by a function network. Therefore, it is all a matter of flexible application and organization of the very same function network and thus lies within the scope of the present approach.

The hope of this approach is that it may lead us on a path finally scaling up universal induction to practically significant levels. It would be nice to backup this hope by a time complexity measure of the present algorithm, which not

available at present unfortunately, since this is work in progress. Further, it can not be excluded that a narrow algorithm is also able to solve all low-complexity problems. In fact, the present algorithm is narrow as well since there are numerous implicit assumptions about the composition of the sequence, e.g. the concatenation of outputs of several functions, no possibility to represent dependencies within a sequence, or regularities between different inputs etc. Nevertheless, since we represent general programs without specific a priori restrictions this setup seems to be general enough to tackle such questions which will hopefully result in a scalable system.

References

1. Hutter H-Prize. `http://prize.hutter1.net`, accessed: 2015-05-17
2. Cover, T.M., Thomas, J.A.: Elements of information theory. John Wiley & Sons (2012)
3. Friedlander, D., Franklin, S.: LIDA and a theory of mind. In: Artificial General Intelligence, 2008: Proceedings of the First AGI Conference. vol. 171, p. 137. IOS Press (2008)
4. Hernandez-Orallo, J.: Beyond the turing test. Journal of Logic, Language and Information 9(4), 447–466 (2000)
5. Hutter, M.: Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability. Springer, Berlin (2005), `http://www.hutter1.net/ai/uaibook.htm`, 300 pages
6. Kitzelmann, E.: Inductive programming: A survey of program synthesis techniques. In: Approaches and Applications of Inductive Programming, pp. 50–73. Springer (2010)
7. Kurzweil, R.: The singularity is near: When humans transcend biology. Penguin (2005)
8. Legg, S., Hutter, M.: A collection of definitions of intelligence. In: Goertzel, B., Wang, P. (eds.) Advances in Artificial General Intelligence: Concepts, Architectures and Algorithms. Frontiers in Artificial Intelligence and Applications, vol. 157, pp. 17–24. IOS Press, Amsterdam, NL (2007), `http://arxiv.org/abs/0706.3639`
9. Legg, S., Veness, J.: An approximation of the universal intelligence measure. In: Algorithmic Probability and Friends. Bayesian Prediction and Artificial Intelligence, pp. 236–249. Springer (2013)
10. Levin, L.A.: Universal sequential search problems. Problemy Peredachi Informatsii 9(3), 115–116 (1973)
11. Li, M., Vitányi, P.M.: An introduction to Kolmogorov complexity and its applications. Springer (2009)
12. Looks, M., Goertzel, B.: Program representation for general intelligence. In: Proc. of AGI. vol. 9 (2009)
13. Mahoney, M.V.: Text compression as a test for artificial intelligence. In: AAAI/IAAI. p. 970 (1999)
14. Potapov, A., Rodionov, S.: Universal induction with varying sets of combinators. In: Artificial General Intelligence, pp. 88–97. Springer (2013)
15. Solomonoff, R.J.: A formal theory of inductive inference. Part I. Information and control 7(1), 1–22 (1964)
16. Veness, J., Ng, K.S., Hutter, M., Uther, W., Silver, D.: A Monte-Carlo AIXI approximation. Journal of Artificial Intelligence Research 40(1), 95–142 (2011)