Game Interpretation of Algorithmic Information Theory

Project Report : CS397

Akshay Kumar
Advisor : Prof. Satyadev Nandakumar
Department of Computer Science and Engineering
IIT Kanpur

April 18, 2013

Abstract

Algorithmic information theory is an area which tries to define concepts in information theory based on concepts in the theory of computing. The objective of our study is to prove certain results of AIT using Game Theoretic Arguments. In many cases, like Muchnik-Vyugin Theorem, Friedburg's Unique Numbering etc., Game Theoretic Arguments simplify the proof greatly.

We present a Game Theoretic proof of some other results, most notably Levin's Coding Theorem. The essence of the theorem is : "If there are many long program which output a string x, then there is a short program which outputs it." The essential algorithm of the proof is simulated using a game and we prove the analogue between the algorithm and the game strategy devised by us.

Algorithmic Information Theory

Algorithmic Information Theory (AIT) is the information theory of individual objects, using computer science, and concerns itself with the relationship between computation, information and randomness. This field is at the juncture of both information theory as well as computer science.

Quoting Gregory Chaitin :
It is "the result of putting Shannon's information theory and Turing's computability theory into a cocktail shaker and shaking vigorously."

Kolmogorov Complexity

An example

Which of the following strings requires more bits to be decribed?

1. 00000000001111111111
2. 10010101101111001011

The first string as be described more succintly as : "ten 0's followed by ten 1's". However, there's no inherent pattern in the second string. Hence, we say that complexity of second string is greater than complexity of the first string. Kolmogorov complexity formalizes this intuitive notion of complexity.

Kolmogorov Complexity

"Kolmogorov Complexity of a string x is the length of the shortest program whose output is x relative to some model of computation f."

$$C_f(x) = \min_p \{|p| : f(p) = x\}$$

Conditional Kolmogorov Complexity measures complexity relative to a string, say y.

$$C_f(x|y) = \min_p \{|p| : C_f(p, y) = x\}$$

Obviously,

$$C_f(x) = C_f(x|\epsilon)$$

Intuitively, $C_f(x|y)$ gives an idea of the information of string x contained in string y.

The Kolmogorov Complexity defined above refers to Plain Kolmogorov Complexity. Another more commonly used complexity is Prefix Kolmogorov Complexity.

The information content and degree of randomness of a string x is measured by $C_f(x)$. Is there a way to describe x independent of f?
Here comes the Invariance Theorem of Kolmogorov Complexity.

Invariance Theorem

Theorem: *Any complexity relative to some model of computation ψ can be tied to a universal description method ψ_0 is the following way:*

$$C_{\psi_0}(x) \leq C_\psi(x) + c$$

for some constant c depending on ψ_0 and ψ but independent of x.

Proof Idea: It essentially follows from the existence of a Universal Turing Machine.

An immediate corollary of this result is:

Corollary:
Given any two universal description method f and g :

$$|C_f(x) - C_g(x)| \leq c$$

for some c dependent only on f and g but independent of x.

Upper bound on $C(x)$

Theorem: *Complexity of a string can at most be length of the string plus a constant.*

$$C(x) \leq |x| + c$$

Proof Idea: A program which trivially outputs x

Structure and Complexity

Theorem: *The relationship between complexity of a string x and a string k-times x is*

$$C(x^k) \leq C(x) + c$$

Proof Idea: Take the program of $C(x)$ and loop it k times.

Non Computability of Kolmogorov Complexity

Theorem: *Kolmogorov Complexity $C(x)$ of a string x is incomputable.*

Proof: Consider the statement : "The smallest string x with $C(x) \geq n$". Devise a program which keeps computing complexity for every string in the enumeration order stopping only when it encounters the first string in the enumeration order whose complexity exceeds n and outputs it. The length of this program is $\log n + c$ ($\log n$ for string n and a constant memory for other description). Hence complexity $= \Omega(\log n)$. This would imply n is asymptotically smaller than $\log n$. This is a contradiction.
Hence, Kolmogorov Complexity of a string x is not computable.

The contradiction arrived in this theorem is very similar to Berry's paradox.

Berry's paradox: Consider the following statement : "The smallest number which can't be described with twelve words". If the words are picked up from a dictionary, there are finitely many combination of twelve words. Hence number of words describable in these twelve words are finitely many. The would immediately imply existence of a number adhering to the aforementioned condition. However, we have already described such a number by the aforementioned statement in less than 12 words (The statement comprises only 10 words). In this way, we arrive at a contradiction.

Incompressibility

1. A string x is said to be incompressible if $C(x) \geq |x|$
2. Similarly, if $C(x) \geq |x| - c$ for some c, then the string x is c-incompressible.

Theorem: *For all n, there exists an incompressible string of length n*

Proof: Number of strings of complexity k is at most 2^k. Hence, number of strings of complexity less than n is at most $\sum_{i=1}^{n-1} 2^n = 2^n - 1$. However, Number of strings of length $n = 2^n$. Therefore, by Pigeon Hole Principle, there must exist at least one string of complexity at least n. This is an incompressible string.

Theorem: *Number of strings of length n that are c-incompressible is at least* $2^n - 2^{n-c+1} + 1$

Proof: Can be proven in a way similar to the previous proof.

Subadditivity of Kolmogorov Complexity

Theorem: $C(x, y) = C(x) + C(y) + \Omega(\log(min(C(x), C(y))))$

Proof: Additional $2 \log C(n)$ are used so as to delimit the two strings. To differentiate the two strings, pad the length of the first string before it in the following fashion: repeat every bit of its length twice and then append 01 or 10 followed by the string. This way, the machine can recognize the end of a string.

Question: Is there a constant c s.t. $\forall x, y : C(x, y) \leq C(x) + C(y) + c$?

Answer: Unfortunately not.

Theorem: *Kolmogorov Complexity is not subadditive*

This is primarily because there's no possible way to recognize the end of a string unless you don't replicate each bit of a string because of which the complexity shoots up to atleast $2C(x)$.

Prefix Complexity

Self-Delimiting Strings

The Universal Turing Machine uses an extra code, say $\triangleright$, to read the end of the program. However. it increases the alphabet set from $\{0, 1\}$. To avoid this, we ideally want that the head reading the program should not run beyond the program. Here comes the notion of **self-delimiting** *i.e.* the Universal Turing Machine should itself know the end of a program without encountering $\triangleright$.

Prefix Set A set A is said to be prefix set (or prefix free) if no string $x \in A$ is prefix or extension of some other string $y \in B$.

No self-delimiting program is prefix of another. Hence, set of self delimiting programs forms a prefix set. One way to make a program self-delimiting is to duplicate every bit of its length, then mark the end of the length by 01/10 and prepend it to the original string.

Prefix Complexity

The minimum size of self-delimiting program that outputs x is called the prefix complexity of x. It is denoted by $K(x)$.

Some results on Prefix Complexity

- Prefix property makes Prefix complexity subadditive.
- Since now there's a way to recognize the end of program, $K(x, y) = K(y, x)$. Which string's description is first desn't matter.
- Conditional Prefix Complexity adheres to the properties of Conditional Plain Complexity. So, we can say that $K(x|y) \leq K(x) \leq K(x, y)$

The relationship between Prefix Complexity, $K(x)$ and Plain Complexity, $C(x)$ is

- $C(x|y) \leq K(x|y) + O(1)$
  When talking of Prefix Complexity, you are essentially removing some possible descriptions of x from the description set.
- $K(x|y) \leq C(x|y) + 2\log C(x|y) + O(1)$
  Pad the length of $C(x|y)$ in the fashion described apriori to get a self-delimiting program outputting $K(x|y)$.

Levin's Coding Theorem

Kraft's inequality

Theorem: Let A be any prefix set. Then

$$\sum_{x \in A} \frac{1}{2^{|x|}} \leq 1$$

Proof: Consider random tosses of a fair coin. What is the probablity of obtaining a string x which is a member of a prefix set A? Since A is a prefix set, no extension or prefix of x can occur in A if $x \in A$. Its probablity is 2^{-x}. Taking it over all possible values of $x \in A$, we get the desired result as the value is a probablity.

Extending Kraft's Inequality to a set of Prefix Turing Machines $\mathcal{M} = \{p_0, p_1, \ldots\}$, $\sum_{i \in 0}^{\infty} \frac{1}{2^{|p_i|}} \leq 1$.

Algorithmic Probablity

Let $x \in \{0, 1\}^*$. The algorithmic probability of x is defined as:

$$\boldsymbol{m}(x) = \sum_{\substack{p \in \mathcal{M} \\ U(p)=x}} \frac{1}{2^{|p|}}$$

- $\sum_{x\in\{0,1\}^*} \boldsymbol{m}(x) \leq 1$ as $\mathbf{m}$ is a subprobablity.
- $K(x) \geq -\log \boldsymbol{m}(x)$ since $2^{-K(x)}$ is one of the terms of the sum defining $K(x)$.

Coding Theorem (Levin)

$$K(x) = -\log \boldsymbol{m}(x) + O(1)$$

Indexed Enumerator A Turing Machine M which given any input $z \in \{0,1\}^*$ such that $M(z)$ enumerates a computably enumerable language is called an Indexed Enumerater.

Request Set A Request Set is a set L of pairs of binary strings (x, s_n) such that $\sum_{(x,s_n)\in L} \frac{1}{2^n} \leq 1$. Here, (x, s_n) is the request and n the request length.

Conditional Requestor An Indexed Enumerator R which, when given any input $z \in \{0,1\}^*$ enumerates a request set is called Conditional Requester.

Theorem: *Given a conditional requestor R fed with string $z \in \{0,1\}^*$, for all $(x, s_n) \in L(R(z))$, complexity of x conditioned on z is at most n plus a contant c_R.*

$$K(x|z) = n + c_R$$

A codeword c_i is assigned from a prefix set to each request (x_i, s_{n_i}) such that $|c_i| = n_i$. The Turing Machine described below outputs x_i on input c_i and z[4].

```
M(c_i, z)
1.  $C = \phi$                                   ▷ The set of codewords
2.  $j = 0$                                      ▷ index for the language $L(R(z))$
3.  for each $(x_j, s_{n_j})$ enumerated by $R(z)$ do
4.      $w$ = the first element in $\{0,1\}^{n_j} \backslash C$ with no proper prefix or extension already in $C$
5.      if $w = c_i$ then
6.          output $x_j$ and halt            ▷$c_i$ is the encoding of $(x_j, s_{n_j})$
7.      else $C[j] = w$, increment $j$ by 1        ▷ Add $w$ to the set of assigned code words
8.  done
```

The crux of the algorithm lies in Line 4. It is always possible to choose such a string w. The set C is a finite union of disjoint intervals which denotes the already allocated strings. The complement of C, C^c is a finite disjoint union of intervals with length strictly increasing to the right.

Game Theoretic Argument for the proof

Contruction of the Game

1. The game is played on an infinitely long perfectly balanced binary tree search tree.
2. As usual, there are two players : Alice (**A**) and Bob (**B**)
    - Bob has a set of pair of strings with him.

    - Alice has a single string with her.

3. All the edges have been given a cost of either 0 or 1. An edge connecting a node to its left child is given a cost 0 whereas an edge connecting a node to its right child is given a cost 1
4. Each node has a treasure value hidden in it which is revealed only when Alice visits the node.

Aim of the game

- **Alice's Aim:** She has to visit that node whose treasure value is associated with the string provided to her.
- **Bob's Aim:** To prevent Alice from visiting the node.

Constraints

- Cost incurred by Alice must be minimum.
- If there's a node whose treasure value has been checked once, then no node in the subtree rooted at this node or in the path from this node upto the root can subsequently reveal its treasure value.

Claim: If the algorithm mentioned previously terminates, Alice has a winning strategy for the game.

Strategy of the game

1. Alice requests pair of strings from Bob, say (a, b).
2. For each pair of string request, Alice travels a path of length ordinal number of b in the enumerated set keeping a minimum cost. Suppose Alice reaches node n.
3. Alice checks the treasure value of node n. If it matches the string provides to Alice, Alice wins and simply outputs a.
4. Else, Alice marks this as **explored** and restarts her journey right from Step 1.

Claim: If the algorithm terminates, the game described here also outputs the same string.

Similarity between the algorithm and the game

- $(a, b) \sim (x_j, s_{n_j})$
- String provided to Alice $\sim c_i$
- Treasure value of the explored node $\sim w$
- Minimum cost property ensures the first favourable element in $\{0,1\}^{n_j}$ is picked up.

- Restriction on exploring a node in the path of a node already explored ensures ensures no proper prefix or extension of already picked up w is chosen.

Hence, the above game correctly outputs a string similar to the one outputted by algorithm.

Conclusion and Future Work

The Game Theoretic Strategy presented here is more or less similar to the Algorithm used. We would like to devise games version differs significantly from the actual proof. Not only that, there are many more results in Algorithmic Information Theory whose proof can be formulated as a game between two players. We would also like to explore those proofs and devise a game strategy for the same.

References

[1] Andrej Muchnik, Alexander Shen, Mikhail Vyugin, *Game Arguments in Computability Theory and Algorithm Information Theory,* preprint,arxiv:1204.0198 v4 (2012).

[2] Alexander Shen, Algorithmic Information Theory and Kolmogorov Complexity, Technical Report, Uppsala University, TR2000-034
http://www.it.uu.se/research/publications/reports/2000-034/.

[3] Ming Li, Paul Vitanyi, *An Introduction to Kolmogorov Complexity and Its Applications,* Third Edition

[4] Satyadev Nandakumar, *CS687: Algorithmic Information Theory*, Lecture Notes

[4] Neil Conway, *Kolmogorov Complexity*, Notes