A Survey on Kolmogorov Complexity

Yuan Shuai
University of Illinois at Chicago
E-mail: syuan5@ece.uic.edu

Abstract

Kolmogorov complexity, which is also called algorithmic (descriptive) complexity is an object, such as a piece of text, to measure the computational resources needed, which are mostly the length of the shortest binary program to specify an object. Strings whose Kolmogorov complexity is small relative to the string’s size are not considered to be complex and easy to use a short program to specify it. Based on Kolmogorov complexity, one can easily tell how many and how fast a string can be compressed, which can be measured by resources in time and space such algorithms require. This is especially important nowadays because information is infinity, but process and storage capacity is finite. In this survey, we will get into the definition of Kolmogorov complexity and several applications in several aspects which Kolmogorov complexity is widely used, including efficiency of strategy optimization, security detection and protection and information distance.

I. INTRODUCTION

Before getting into the concept of Kolmogorov complexity, we can first check an example to know more intuitively. Let’s consider three strings shown below:

\[ \begin{align*}
010101010101010101010101 \\
100111011101011100100110 \\
110100110010110100101100
\end{align*} \]

All the strings are 24-bit binary strings, and can be considered as a result of 24 fair coin flips with 0 standing for head and 1 standing for tail. The first string can be described as "01" repeated 12 times. The second one seems to be no pattern to follow, and if we need to describe it, we have to recite its contents verbatim, which means we have to use a sequence as long as the original one. The third one seems to be no pattern to follow as first, but by carefully examine it, we can tell that if there are an odd number of 1’s in the binary expansion of position i, the ith position is set to be 1.

From the example above, we can tell that even following the same coin flip rule to generate strings, the difference in the complexity of describing each string is remarkable large. But what is a description? Let’s define the whole set of alphabet in binary, which is common in digital computation and computer program, that is \( \sum = \{0, 1\} \). Let \( f : \sum^* \rightarrow \sum^* \). Relative to \( f \), a description of a string \( \delta \) is some \( l(p) \) with \( f(\delta) = l(p) \).

Now we can understand the concept of Kolmogorov complexity. Kolmogorov complexity of finite strings was introduced by Andrei.N.Lolmogorov and Gregory.J.Chaitin independently to give precise computational meaning to define the randomness of a string, which also means the shortest unambiguous description of information.

Although "Kolmogorov complexity" is the commonly understood and widely accepted name, the theory also goes by names such as "Algorithmic Information theory", "Kcomplexity", "Kolmogorov-Chaitin randomness", "Algorithmic complexity theory", "Descriptive complexity", "Program-Size complexity", and "Solomonoff-Kolmogorov-Chaitin complexity", which would be the most appropriate name among all.

After the concept was introduced, many applications in mathematics, computer science and network have been revealed. In this paper, we will check several aspects in which Kolmogorov complexity has been proven to be extremely useful. In this survey, we will examine three aspects to see how Kolmogorov complexity plays its role in them, and here are the brief summary.

In strategy optimization aspect, Kolmogorov complexity helps strategy optimization with the constraint of limited memory space to find a strategy with the minimal cost and can be stored within the given memory space. The importance of considering the constraint of limited memory space in large scale practical systems is due to the fact that a strategy is generally a high dimensional function, which is difficult to store. Consider a strategy mapping an information space \( \mathbb{R} \) to an action space \( \kappa \), the strategy space \( \chi \) is \( \mathbb{R}^\kappa \) large. So it is of practical importance to study the constraint of limited memory space in strategy optimization.

In security detection and protection aspect, Kolmogorov complexity can be used to detect intrusion. According to Kolmogorov Complexity, a string is considered as no pattern, which is randomly generated if the shortest
Turing machine that can encode it is at least as long as the string itself. But a nonrandom string with patterns can be described by some Turing machine that is shorter than the string. By using certain functions, a so-called anomaly-based intrusion detection system can be build.

In information distance aspect, Kolmogorov complexity can find its usage in many ways. The first one is in data searching. Taking google search as an example. Usually users describe their information needs by typing a few keywords in their search boxes. Those keywords are likely to be different from the terms which users really look for in the documents. As a consequence, in many cases, the results returned by information retrieval system don’t match to the users’ need. This raises a fundamental problem of term mismatch in information retrieval. By comparing information distance, Kolmogorov complexity can enhance the precision of information retrieval system. Another usage would be program plagiarism detection. Based on Kolmogorov complexity, a metric can be established to measure the amount of shared information between two computer programs, to enable plagiarism detection.

II. DEFINITION OF KOLMOGOROV COMPLEXITY

Now we can define the Kolmogorov complexity of a string x as the minimal description length of x.

Definition The definition of Kolmogorov complexity $K_\mu(x)$ of a string x with respect to a universal computer $\mu$ is defined as

$$K_\mu(x) = \min_{p: \mu(p) = x} l(p)$$

the minimum length $l(p)$ over all programs that print x and halt. Thus we know $K_\mu(x)$ is the shortest description length of x over all descriptions interpreted by computer $\mu$.[4] [6] We can understand the Kolmogorov complexity by explaining it below if we find the concept is difficult to understand. If A can describe a sequence to B in a way that B can come up with an unambiguous computation of that sequence in a finite time, then the bits measuring the information in the communication is the upper bound on the Kolmogorov complexity. For example, $\pi$ is an irrational number, and we can build a program saying "Print out the first 1,857,469,874,123,547,854 bits of $\pi$." There are 59 symbols in this program, and by using ASCII to demonstrate this huge number, the actual Kolmogorov complexity is no more than $59 \times 8$ bits. Compared to 1,857,469,874,123,547,854 bits, we can tell how many space and how much time can be saved in processing this program.

A. Theorems and Properties of Kolmogorov Complexity with Some Examples

There are a lot of theorems, properties and lemmas of Kolmogorov complexity, and we will not look into them one by one. But it helps to learn some both important and interesting theorems before we get into the applications.

Theorem 1: (Conditional complexity is less than the length of the sequence.)

$$K(x|l(x)) \leq l(x) + c$$

Since the sequence x is given, the length of x is well known and need not to be described. Let’s assume sequence x is $\{x : x_1x_2 \cdots x_{l(x)}\}$, a program to describe the sequence can be as simple as "Print the following $l(x)$-bit sequence: $\{x : x_1x_2 \cdots x_{l(x)}\}$". The Kolmogorov complexity of the program is no more than $l(x) + c$, where c stands for the extra bits used to make the program understandable.

Theorem 2: (Upper bound on Kolmogorov complexity.)

$$K(x) \leq K(x|l(x)) + 2l\log l(x) + c$$

Theorem 1 gives the upper bound on conditional Kolmogorov complexity. The only difference between the Theorem 2 and 1 is that we have to use extra length to describe how long the sequence is. We can modify the program used in Theorem 1 by repeating each bit of the binary program twice with a "STOP SIGN" like 01 attached to the end, we can derive the bound.

However this is a loose bound. A tight upper bound will be given here, although we will not discuss it.
\[ K(x) \leq K(x|l(x)) + \log^*l(x) + c \]

**Theorem 3:** (Lower bound on Kolmogorov complexity.)

The number of strings \( x \) with complexity \( K(x) \prec k \) satisfies

\[ | \{ x \in \{0,1\}^* : K(x) \prec k \} | \prec 2^k \]

This theorem can be easily proven by listing all the possible programs of length less than \( k \). Programs include 1 null program, 2 1-bit programs, 4 2-bit programs till \( 2^{k-1} \) (k-1)-bit programs. The summation of all programs is \( 2^k - 1 \), which is less than \( 2^k \). It also means there are very few simple sequences with low Kolmogorov complexity, most of them are complicated.

After knowing what Kolmogorov complexity is and some of its properties, we now check some examples to understand Kolmogorov complexity more.

**Example 1:** (An integer \( n \))

If the program knows the number of bits to represent an integer, the only work we need to do is to fill the binary with values. So we use \( \log n \) bits to specify the integer \( n \), then we use \( \log \log n \) to specify the length \( \log n \), we can continue this iterate logarithm until the last positive term. The summation can be written as \( \log^*n \), and we know the Kolmogorov complexity of an integer is bounded by

\[ K(n) \leq \log^*n + c \]

**Example 2:** (A \( n \)-bit sequence with \( k \) ones)

At first we think we have to recover each value on each bits, so the whole sequence is incompressible and the Kolmogorov complexity is the length of the sequence. But by setting a program like "Generate all \( n \)-bit sequences with \( k \) ones, put them from the smallest to the largest in number, and pick the \( i \)th one", we can get the exact sequence that we want. The only variables are \( k \) and \( i \). Using the theorems above, we can get the Kolmogorov complexity of the program is

\[ K(p) = c + 2\log k + \log \left( \frac{n}{k} \right) \]

### B. Kolmogorov Complexity and Entropy

**Theorem 1:** (Relationship between Kolmogorov Complexity and Entropy.)

Without proving the theorem, we can derive the relationship between Kolmogorov complexity and entropy as follows:

\[ \frac{1}{n} \sum_{x^n} f(x^n)K(x^n|n) \leq H(X) + \frac{1}{n} \log n + \frac{\xi}{n} \]

for all \( n \). Thus

\[ E \left( \frac{1}{n} K(X^n|n) \right) \rightarrow H(X) \]

This theorem shows that with large number, Kolmogorov complexity is the entropy of the sequence.

### III. Strategy Optimization

When dealing programs and computers, the first constraint comes into the matter is limited memory space. Simply we can show that the binary strategy space mapping from information to action is \( 2^{2^n} \) \cite{5}, which increases even more than exponential functions and easily fill any memory space. So we focus on how to deal with the constraint of limited memory space, which is also the idea "how to sample simple strategies". The difficulty mainly attributes to: What strategies can be stored within a given memory space?
A. Problem formulation

As we can see from the examples above, Kolmogorov complexity helps to reduce the strategy space in this problem. The key idea is to scan all possible encoding/decoding schemes, and compare the code length of the strategy among all. Each encoding/decoding technique can be regarded as a high-dimensional function, which can be modeled by a Turing machine (TM). We can get the Kolmogorov complexity for a strategy $\gamma$ is that $K(\gamma) = \min \{ l(TM) + K(\gamma|TM) \}$, which is the summation of the length of the Turing machine and the Kolmogorov complexity of the strategy using such Turing machine. Given the limited memory constraint $C_0$ to fit the reality, we set $K(\gamma|TM)$ less than $C_0$. Now we can exam a strategy representation called OBDD. An OBDD figure is given below:

![Fig. 1. An example of OBDD](image)

In the figure, each node represents the variable, dotted and solid lines represent variable to be 0 or 1 respectively. Given this, we can start from the top node, which is also called root, then trace down certain line, which is also called branch to the leaves. To calculate the minimal code length to describe a strategy, we first simplify the OBDD of that strategy, and then propose a method to calculate the minimal code length to describe that OBDD. When none of the leaves are identical and the step number of the branch from root to leaves are the same of an OBDD, it is called POBDD. If we remove the nodes with identical low- and high-successors, and don’t care about the order of tests, then we obtain a reduced OBDD, which is ROBDD. Then we come up with an algorithm to calculate the lower bound to represent $K(\gamma|TM)$. Here is the brief steps of the algorithm.

**Step 1:** Set the number of nodes $b$ to 1 as a start, and specify an order of tests.

**Step 2:** Use $b$ nodes and the order to generate a POBDD.

**Step 3:** Use the POBDD to get the ROBDD.

**Step 4:** Calculate strategy of this ROBDD.

**Step 5:** Go through all the orders of tests by increasing $b$, repeat step 2-4 until each strategy is described by some ROBDD.

**Step 6:** Now we have the ROBDD $\gamma$ for each $\gamma$, then we output the lower bound of $K(\gamma|TM)$ to meet the constraint.[5]

IV. Security Detection and Protection

In the network communication, all we care about information is integrity, confidentiality and availability of a resource. An intrusion is a combination of actions trying to jeopardize them. Traditional Intrusion Detection System (IDS) has its disadvantage. It is always one step behind intrusions happened to learn the patterns to prevent the same intrusions again, and it requires a large detection database to store all the intrusion patterns. A new IDS called Anomaly-based Intrusion Detection System is developed applying Kolmogorov complexity to enable compact storage and efficient detection. This new IDS can protect against both old intrusions and new attacks.[1]
A. Rationale behind the scheme

So far there are several approaches that we can use. And we can examine advantages and disadvantages of each one.

- **N-Grams**: The behaviors of the program are learned from sequences during the executing processes. And by breaking sequences into blocks of N bits long, we can make our learning algorithms tractable. Each block is called N-Gram, and all of them are stored in a database. When during testing, a sliding window of the same size N goes through sequences and compares those blocks with N-Grams in the database. If there are mismatches, then the system would determine whether it is an intrusion or not. Though it is an easy implemented algorithm, there are several drawbacks we need to know. First of all, it can not detect anomalies if the anomalies are longer than N. Second, it can not “learn”, all the processes it accepts are those inputs during the training. Last but not least, it can't examine loops and branches, which happen to be the most common workflows in the programs.

- **Automata Modeling**: Thanks to the theoretical support of Kolmogorov complexity, we use automata to build systems, which can successively capture pattern in loops and branches. And more, compared to N-Gram, it is more compact and constant.

B. The ALERGIA Algorithm

Stochastic Finite Automata (SFAs) are constructs similar to finite automata, but with probabilities associated with all transitions. And the language accepted by one is called a Stochastic regular Language (SRL).[1] Two SRLs are said to be equivalent if they have the same strings with the same probabilities. If $L_1$ and $L_2$ are two SRLs and they are equivalent, we can derive the equation below with $p_1(w)$ and $p_2(w)$ the probabilities of $w$ in each SRL.

\[
L_1 \equiv L_2 \iff p_1(w) = p_2(w), \forall w \in A
\]

The ALERGIA Algorithm is to build a prefix-tree acceptor $T$. At each node in $T$, the frequencies of the arcs that leave the node are calculated and stored in $T$. By checking the associated transition probabilities, termination probabilities and destination states, the system can learn new patterns in the same SRL.

The new system can successfully capture attacks, including buffer overflows, server attacks, Trojan horses and login attacks with an acceptable false alarm rate. By applying Kolmogorov complexity to interpret and represent the data, we can use this system in a Hybrid IDS model. The system framework is given below: In

![Diagram of the Hybrid IDS system](image)

Fig. 2. Hybrid IDS system

In this model, we build misused-based component and anomaly-based component in it, and by using an appropriate feedback mechanisms, the two components could result in better detection accuracy and efficiency.[1]
V. INFORMATION DISTANCE

Among all the aspects related to Kolmogorov complexity, information distance gives one of the most fascinating notions. There are a lot of definitions of information distance, but normally we use the minimal number of edit operations from a fixed set required to transform one string in the other string to define information distance.[2]

A. Distance function and Algorithm information distance

Distance function can be defined as follows, which intuitively identical to the definition of distance which we are familiar with [2]:

A distance function $D$ is a nonnegative real value function, and the function is defined on a set $X$ for if every $x, y, z \in X$, we have

- $D(x, y) = 0$, iff $x = y$ (the identity axiom)
- $D(x, y) + D(y, z) \geq D(x, z)$ (the triangle inequality)
- $D(x, y) = D(y, x)$ (the symmetric axiom)

As we have already defined information distance, the next thing we care about is that what extent information we need to transform $y$ from $x$ and how much it can overlap with the information that transforms $x$ from $y$. In some simple cases, complete overlapping can be achieved, which also means the same number of operations to transform $x$ to $y$ and $y$ to $x$. As an example, if $x$ and $y$ are independent binary sequences with the same length, then their bitwise exclusive $x \oplus y$ is the shortest operation to serve both.

Since complete overlapping is not always the case, we need to find the maximal overlapping, which is also to find the shortest operation. We can find the conditional Kolmogorov complexity of $X$ given $Y$, which is $K_1(X|Y)$ and $Y$ given $X$, which is $K_2(Y|X)$. Without loss of generalization, let $K_1(X|Y) \leq K_2(Y|X)$, then $d = K_2(Y|X) - K_1(X|Y)$, so there exists a sequence $Z$ of length $K_1(X|Y) + K(K_1(X|Y), K_2(Y|X))$ and a string $Q$ of length $d$ that the shortest operation is $Z$ to transform $XQ$ to $Y$ and $Y$ to $XQ$. It means that the information to pass from $X$ to $Y$ can always be maximally correlated with the information to get from $Y$ to $X$.[2] So it will never happen when a lot of information is required to transform from $X$ to $Y$, but a lot of independent information is required to transform from $Y$ to $X$.

B. Program plagiarism detection

How can we determine two similar sentences, two similar programs, or say how can we detect plagiarism? This is a very practical problem for instructors in the university to secure the quality of education and programmers to protect his/her own intellectual property. A lot of plagiarism detection programs have been developed, and they can be roughly divided into two kinds: attribute-counting systems and structure metric systems. A attribute-counting system only counts the number of distinct operators, distinct operands, total number of operators of all types, and total number of operands of all types, and then constructs a profile using these statistics for each program.[3] We can give a simple attribute counting system as below:

- $\eta_1$ = number of distinct operators.
- $\eta_2$ = number of distinct operands.
- $N_1$ = total number of operators of all types.
- $N_2$ = total number of operands of all types.

Then we can determine the similarity of two programs by calculating two measure metrics.

- $V = (N_1 + N_2) \log_2(\eta_1 + \eta_2)$. 
\[ E = \frac{m_2(N_1 + N_2)\log_2(m_1 + m_2)}{2\eta_2}. \]

Though attribute counting system is easy to implement, it has its withdraws. The biggest problems is that it is incapable to detect sufficiently similar programs.

Structure metric system extracts the structures and compares the representation of each program, so it is a more precise system to detect the similarity. However, it has its withdraws as well, it is incapable to deal with transposed code segments.

A more useful and precise detection system can use information distance. Given two programs \( x \) and \( y \), we can build a information-based program distance model by the equation given below:

\[ D(x, y) = 1 - \frac{K(x) - K(x|y)}{K(xy)} \]

And since \( K(x) - K(x|y) \approx K(y) - K(y|x) \), we know \( D(x, y) \) is symmetric. And by simplify it we can have a normalization equation of information distance as below:

\[ D(x, y) \approx \frac{K(x|y) + K(y|x)}{K(x) + K(y)} \]

And we know the range of normalized \( D(x, y) \) is between 0 and 1, the smaller the value is, the more similar the two programs become. Though in principal, it is very difficult to get \( D(x, y) \) precisely, we can use Token Compress Algorithm (TCA) to get an approximation of it. Algorithm is given below: TCA is to compute

<table>
<thead>
<tr>
<th>Algorithm: TokenCompress</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> A token sequence ( s )</td>
</tr>
<tr>
<td>( i = 0; )</td>
</tr>
<tr>
<td>( \text{while } (i &lt;</td>
</tr>
<tr>
<td>( \text{if } (p, \text{compressProfit} &gt; 0) )</td>
</tr>
<tr>
<td>( i = i + p.length; )</td>
</tr>
<tr>
<td>( \text{else} )</td>
</tr>
<tr>
<td>( i++; )</td>
</tr>
<tr>
<td>( \text{return } )</td>
</tr>
<tr>
<td>( \text{and repeat pairs stored in } \text{repFile}. )</td>
</tr>
</tbody>
</table>

Fig. 3. Token Compress Algorithm

the distances between the token sequences pairs of the two programs generated by another source program. The pairs can be ranked by the similarity distance, and we can easily tell if the plagiarism happens or not.

VI. CONCLUSION

As we surveyed Kolmogorov complexity and several interesting aspects, we know that Kolmogorov complexity gives us a different view of information and some of its properties and applications are beyond expectation. Though all the survey we did are classical Kolmogorov complexity, still there are a lot of new interesting implementation in aspects like search engine design and soft testing. And new Kolmogorov complexity called quantum Kolmogorov complexity will be put into practice in the coming future.
REFERENCES


