Finite Inductive Sequences, Kolmogorov Complexity with Application to Genome Sequences

Paul S. Fisher¹, Jinsuk Baek¹, John O. Adeyeye², and Joao Setubal³

¹Department of Computer Science
Winston-Salem State University
Winston-Salem, NC 27110
{fisherp, baekj}@wssu.edu

²Department of Mathematics
Winston-Salem State University
Winston-Salem, NC 27110
adeyeyej@wssu.edu

³Virginia Bioinformatics Institute
Virginia Institute of Technology
Blacksburg, VA 24061
setubal@vbi.vt.edu

Abstract

We describe current work in examining the complexity of genomic sequences at the amino acid level. We have provided an original algorithm description and research results when considering this algorithm as a measure of Kolmogorov complexity of such candidate strings. The results can be used to estimate comparative string values, identifying substrings with comparable values, and determining regions of the gene where complexity is low or high to determine such things as similarity, potential promoter regions, mutation regions, or other features of interest in gene comparison. The algorithm applied to this problem allows the sequence to be processed into a structure that provides the fundamental component rules for disassembling the string. As a result, we provide this structure which can be used directly for the applications described above. The algorithm uses a mathematical concept of finite induction, similar to Markov Models, but with unique features to determine the syntax of the string, or strings being examined.

1. Introduction

Suppose we consider a string of irregular composition such as one generated by the amino acids found in the expression of a gene. The Kolmogorov complexity of such a string is defined to be the symbols used to describe, compute, or represent the string. For example, consider the strings:

\[ \text{actgactgactgactgactgactgac} \]
\[ \text{tgactgactgactgactgactgac} \] (1)
\[ \text{accaagactggactaacgcacaagtttaaacatggcgca} \]
\[ \text{atcccttacgtacgtatatctatac} \] (2)

Both strings have the same length, but the first string can be represented by the symbols “actg 16 times”, while the second string requires a definition equivalent to the string length, as there is no obvious repetition of symbols that would allow for a more concise definition. The first string has a complexity of 13. That is, there are 13 symbols in the representation “actg 16 times”, while the second string has complexity of 64 as this is the length of the string. Thus, the complexity of a string is the length of the representation of that string in some descriptive language. Since there is no description for the second string, the complexity of the string is just the string itself. The upper bound of the complexity of a string would be the string itself. This limits the ability to compare such complex strings, as the measure of comparison of such strings just becomes their length, and that provides almost no useful information.

We propose a new method for dealing with such strings whose complexity is only measured by the length of the string. This technique, called finite inductive sequences (FI) [3], and it provides a way to characterize such strings. A string that is finitely inductive is a string where the determination of any symbol in the string can be uniquely specified by some number \( n \) of previous symbols, for some fixed \( n \). A string with this property, and every finite string has this property as does many infinite strings, can be further manipulated under particular circumstances to have the value of \( n \) altered to any a priori value smaller than \( n \). In this paper we describe the FI formulation and apply it to complexity measures of strings in the sense of Kolmogorov.

In the next section we describe FI, Section 3 deals with some issues relating to sequences from Kolmogorov theory. Section 4 relates the material in Sections 2 and 3 showing how FI can be to resolve some issues when looking for potential string similarities in terms of symbol complexity, or string comparison or substring comparison when dealing with measures of similarity. Section 5 provides a relationship between FI and Kolmogorov complexity, and Section 6 is the conclusion.

2. Finite Inductive Sequences

In this section we describe a kind of sequence composed of symbols coming from some alphabet with certain mathematical properties. Sequences of symbols
are said to be **finitely inductive (FI)** over a finite alphabet if the choice of any symbol at any particular position in a sequence depends immediately on only the choices of symbols at the previous $n$ points. The least such $n$ is called the **inductive base** of the sequence. Given an FI sequence, an **implicant** is a pair $(w, p)$ consisting of a word $w$ over the alphabet and a symbol $p$ of the alphabet such that $w$ occurs at least once as a substring of the sequence; and whenever $w$ occurs as a substring of the sequence there is a succeeding entry and it is $p$. This $w$ is called the **antecedent** while $p$ is called the **consequent**. An implicant is said to be in **reduced form** if no proper terminal segment of its antecedent is the antecedent of another implicant.

We note the following without proof [3]:

a. Every finite sequence is finitely inductive.

b. For any finitely inductive sequence, the inductive base is the maximal length of the antecedents when considering all of its reduced form implicants.

c. If an FI sequence has inductive base $n$ and an alphabet of $k$ symbols, then $k^n$ is an upper bound for the number of its reduced form implicants.

d. If there exists one implicant that has inductive base less than the value of $n$, then the inductive base of all implicants can be changed to any a priori value less than $n$.

A **Ruling** is a function table resulting from an FI sequence(s) consisting of a structure with $k$ levels. The rules are defined from the string according to a push up formulation of the symbols. In order to represent a string in a Ruling, we select an inductive base, and all implicants that can’t be formulated with a unique antecedent of length less than or equal to $n$ will have their consequent pushed out to the next level. As the inductive base is reduced, the number of levels will increase. The number of rules remains nearly invariant as the number of levels change.

The levels are simple, and typically do not exceed ten, and yet the structure can exhibit significant complexity. Consider the string (2) as given in Section 1. This is:

```
accaagactgacgactgaacgacgttaaaac
atgagatcactctgaagtatctatatctac
```

The process of building the ruling is called **Factoring.** The actual algorithm is very efficient and can be implemented in $O(n)$ time [4]. If we set the inductive base to 5, we get a ruling for this string consisting of two levels with 51 implicants in the first level and 1 implicant in the second level. Thus, the above string is represented by 52 implicants, no implicant longer than six characters, five in the antecedent and one consequent. In fact the ruling contains implicants of inductive bases:

<table>
<thead>
<tr>
<th>Antecedent Length</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Num of Implicants</td>
<td>1</td>
<td>2</td>
<td>18</td>
<td>23</td>
<td>8</td>
</tr>
</tbody>
</table>

A few implicants are: $ga \rightarrow c$, $tg \rightarrow g$, $aaac \rightarrow a$, $aaca \rightarrow t$, $acgca \rightarrow c$, and $agact \rightarrow g$.

If we reduce the inductive base to 4, then the resulting ruling has the configuration: 45 rules in Level 0, 5 rules in Level 1. This ruling contains implicants of IB lengths:

<table>
<thead>
<tr>
<th>Antecedent Length</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Num of Implicants</td>
<td>2</td>
<td>3</td>
<td>20</td>
<td>25</td>
</tr>
</tbody>
</table>

If we again reduce the inductive base to 3, we obtain the following for the 46 rules in the ruling, which now requires three levels:

<table>
<thead>
<tr>
<th>Antecedent Length</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Num of Implicants</td>
<td>2</td>
<td>11</td>
<td>33</td>
</tr>
</tbody>
</table>

From this, one can easily see that the number of symbols required for the representation of the string is:

<table>
<thead>
<tr>
<th>Inductive Base</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Symbols</td>
<td>251</td>
<td>218</td>
<td>169</td>
<td>112</td>
</tr>
<tr>
<td>Number of levels</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>9</td>
</tr>
</tbody>
</table>

If we now consider string (1) from Section 1:

```
actgacgactgacgactgactgactgactgactgactgactgactgactg
```

The optimal factoring of this string produces a ruling of six rules in two levels, and there are two rules of antecedent length 2, and 4 rules of antecedent length 2. This produces a ruling measure of 16 symbols for the complexity.

Using this standard form of sequence representation provides a normative measure for comparing the two sequences. We also note that the two sequences are wildly different from a complexity measure. We violate the guidance for Kolmogorov complexity [6], [7] in that the representation of more complex strings is bounded by the $IB^*|s|$ where $IB$ is the inductive base and $|s|$ is string length. For most cases this upper limit on string complexity will not be reached unless the string is pseudo random, and it is difficult to construct such a string.

### 2.1 Pseudo Random Strings

Immediately the nature of strings [3] that come into consideration are strings with little differentiation between the symbols making up the finite string. For example, long runs of a single symbol then a unique symbol followed by another long run of the same symbols are such cases. Consider the example consisting of 64 symbols:
Using an inductive base of 4, this produces a ruling of 12 levels and each level is identical consisting of four implicants:
\[
\begin{align*}
g &\rightarrow a & ga &\rightarrow a & gaa &\rightarrow a \\
\& t &\rightarrow a & ta &\rightarrow a & taa &\rightarrow a
\end{align*}
\]

The only exception is the last level and it only contains the first three implicants. The initial sixteen symbols can’t be factored, as the antecedent is limited to four, and this would require a total of 16 symbols in the antecedent. However, the run length string equivalent to this consisting of the description: ‘16 a 1 g 47 a’ It would be factorable so, with a simple transform one can still compare these strings, but this is generally not useful information.

The second class of finite strings that are non-factorable are those that are pseudo random, such as the expansion of a transcendental number such as PI. In general one can generate pseudo random strings as follows:

Consider four symbols a, c, t, g. These are represented by the binary values 00, 01, 10 and 11 respectively. If we perform one shift and complement the lowest order bit as two steps, we get a mapping back to the original symbols, and this is shown in Table 1 and Figure 1.

<table>
<thead>
<tr>
<th>Shift</th>
<th>Complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>00</td>
</tr>
<tr>
<td>c</td>
<td>01</td>
</tr>
<tr>
<td>t</td>
<td>10</td>
</tr>
<tr>
<td>g</td>
<td>11</td>
</tr>
</tbody>
</table>

Figure 1. Directed Graph of Mapping with Labeled Edges where 0 is Down and 1 Else

There are two simple paths (a simple path is a path through each vertex in the graph, with no vertex being visited more than once) in Figure 1: actga and ctgac. If we replace these paths with the binary number of the edge connecting the vertices, then we have two strings: 0011 and 0110. If we consider the path as a repeating cycle, and then factor the longer resulting strings, we obtain the following implicants for the two strings:
\[
\begin{align*}
00 &\rightarrow 1 & 01 &\rightarrow 1 & 11 &\rightarrow 0 & 10 &\rightarrow 1 \\
01 &\rightarrow 1 & 11 &\rightarrow 0 & 10 &\rightarrow 0 & 00 &\rightarrow 1
\end{align*}
\]

Substituting the original symbols in the vertices, we can get identical implicants for both paths:
\[
\begin{align*}
a &\rightarrow c & c &\rightarrow t & t &\rightarrow g & g &\rightarrow a
\end{align*}
\]

Observations: If S is a finite sequence, and if the implicants of S have uniform antecedent length, then the string S is pseudo random, and there is no effective factorization for the string S. By default, the complexity of such pseudo random sequences is the upper limit of complexity which was stated as IB*|s|.

3. Kolmogorov Complexity – A Review

The immediate question now becomes the actual language used to describe the string. One can use a simple representation, or a computer program, or a finite state machine generating the string. The complexity of such representations would then be the length of the program or the length of the specification for the finite state machine. In measuring length one can consider for each constant the number of bits required to represent the constant, or count only the number of digits or alpha characters, as long as the counting process is consistent. One would expect that somehow in the description language adopted, there would be a minimal representation, which in the case of a program would be the minimal, non-redundant representation from the family of programs that generate the string.

The Kolmogorov complexity of an object, such as a piece of text, is a measure of the computational resources needed to specify the object.

More formally, the complexity of a string is the length of the strings shortest description in some fixed universal description language. The sensitivity of complexity relative to the choice of description language is discussed below. It can be shown that the Kolmogorov complexity of any string cannot be too much larger than the length of the string itself. Strings whose Kolmogorov complexity is small relative to the string’s size are not considered to be complex. The idea is that K(x) gives us a way to describe the randomness of x.
3.1 Definition

To define Kolmogorov complexity, we must first specify a description language for strings. Such a description language can be based on any programming language, such as Lisp, Pascal, or Java. If \( P \) is a program which outputs a string \( x \), then \( P \) is a description of \( x \). The length of the description is just the length of \( P \) as a character string. In determining the length of \( P \), the lengths of any subroutines used in \( P \) must be accounted for. The length of any integer constant \( n \) which occurs in the program \( P \) is the number of bits required to represent \( n \), that is (roughly) \( \log_2 n \).

Any string \( s \) has at least one description [6], [7], namely the program:

```plaintext
function GenerateFixedString()
    return s
```

If a description of \( s \), \( d(s) \), is of minimal length, i.e. it uses the fewest number of characters, then it is called a minimal description of \( s \). The length of \( d(s) \) is of minimal length – i.e. it uses the minimal number of characters in the description of \( s \). Then the length of \( d(s) \) – i.e. the number of characters in the description – is the Kolmogorov complexity of \( s \), written \( K(s) \). Symbolically,

\[
K(s) = |d(s)|
\]

In the following, we will fix one definition and simply write \( K(s) \) for the complexity of the string \( s \).

The effect of the choice of description language on the value of \( K \) is given by:

**Theorem:** If \( K_1 \) and \( K_2 \) are complexity functions relative to description languages \( L_1 \) and \( L_2 \) then there is a constant \( c \) (which depends only on the languages) such that for every \( s \):

\[
|K_1(s) - K_2(s)| \leq c
\]

3.2 Basic Results

As stated \( K(s) \) will be the complexity of the string \( s \). It is not hard to see that the minimal description of a string cannot be too much larger than the string itself: The program above named GenerateFixedString that outputs \( s \) is a fixed amount larger than \( s \).

**Theorem 3.2.1.** There is a constant \( c \) such that for every \( s \) \( K(s) \leq |s| + c \).

The first result is that there is no way to effectively compute \( K \).

**Theorem 3.2.2.** \( K \) is not a computable function.

Remark: In other words, there is no program which takes a string \( s \) as input and produces the integer \( K(s) \) as output.

3.3 Chain Rule for Kolmogorov Complexity

Here are some properties of \( K(x) \).

1. \( K(x) \leq |x| + c \) for all \( x \). (We can see intuitively that there is a program \( P \) that just prints out its input, and that \( K(x) \leq K_P(x) + c \).)
2. \( K(xx) \leq K(x) + c \) for all \( x \). (We can take a program for \( x \) and put it in a loop, increasing the program size by only a constant.)
3. For any partial computable \( h \), \( K(h(x)) \leq K(x) + ch \), where \( ch \) is the length of a description of \( h \).
4. \( K(y|x) \leq K(y) + c \). (At worst, \( x \) is of no benefit in describing \( y \).)

The chain rule for Kolmogorov complexity states that:

\[
K(x, y) = K(x) + K(y|x) + O(\log(K(x, y)))
\]

This states that the shortest program that produces \( x \) and \( y \) is no more than a logarithmic term larger than a program to reproduce \( x \) and a program to reproduce \( y \) given \( x \).

4. FI and String Complexity

Consider the following three nucleotide strings:

\[
\begin{align*}
TCATCCCTGTCATGGATTTG & \quad (5) \\
AACTGAGATTTGCTGAATTAC & \quad (6) \\
TCAAAAGGCAAATAGCCATGAA & \quad (7)
\end{align*}
\]

These strings represent a portion of an early stage breast cancer gene. If we consider these three sequences with an inductive base of 4, and two levels in the ruling for each string, we obtain the following comparative results:

<table>
<thead>
<tr>
<th>Level</th>
<th>IB</th>
<th>String (5)</th>
<th>String (6)</th>
<th>String (7)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>22</td>
<td>23</td>
<td>13</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>8</td>
<td>14</td>
<td>16</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Total Implicants</td>
<td>34</td>
<td>39</td>
<td>38</td>
<td></td>
</tr>
<tr>
<td>Total Implicants</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>
If we allow the inductive base IB to be as large as it needs to be for these three examples to build a ruling with only one level then we have the following results:

**Table 3: Factoring Strings (5), (6), (7) with System Selected IB**

<table>
<thead>
<tr>
<th>Level</th>
<th>IB</th>
<th>String (5)</th>
<th>String (6)</th>
<th>String (7)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>5</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>19</td>
<td>21</td>
<td>13</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>4</td>
<td>13</td>
<td>16</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>6</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>0</td>
<td>6</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Total Implicants</td>
<td>36</td>
<td>38</td>
<td>42</td>
<td></td>
</tr>
</tbody>
</table>

Table 2 suggests that the string (7) is a lot more irregular than the other two strings since it has several implicants of IB two. String (6) is the most regular of the three followed by (5) and then by (7). Now from Table 3 we see string (5) has two common subsequences of six symbols in two places. String (6) has two common substrings of length 5. String (7) has six common substrings of length four. Again string (7) is the most irregular as it still has the most implicants of IB equal to two. It may be that these common substrings are used several times in the string, but we do know that they appear at least twice because of the way they are constructed.

The last thing we point out concerns the long IB implicants. If we concatenate the three strings (5), (6), (7) together and then process them, we get some longer substrings where all but the first symbol in the antecedent is identical as shown in Table 4. This gives a powerful technique to identify common substrings [2], [8], [9]. We do not discuss the issue or mutations, but the software system will provide common substrings under mutation.

Table 4 shows the long implicants, and as can be seen, the initial symbols must be different so that the implicants are unique, but the remainder of the antecedent is identical in these pairs [1]. This initial symbol and the consequent may be due to single nucleotide polymorphisms (SNP’s).

**Table 4: Common Substrings Pairs Found Among (5), (6) and (7)**

| CACAGAG | G | AATTTGC | C |
| GACAGAG | A | GATTTGC | T |
| ACACTG  | C | AGAGATT | A |
| CCACTG  | A | TGAGATT | T |
| TGATTG  | G | AGATTTG | C |

We can now compare the strings (5), (6) and (7) with three strings generated randomly. These are (8), (9) and (10).

- **ATAATTGTGTCGCCATCTAACCTCG**
- **AGTTGCTAGTTTCCAGGCGCTTTCCGT** (8)
- **TCGTCCTCTGTAGGGGTCTACTTT**
- **CCCAGGCTGACCTGTCCCCTGCCCCTA** (9)
- **GGAGATTTGTATCTTGCGCAGCCG**
- **GCTCATGCTCGGTTGACATTTCCGCC** (10)

If we concatenate these strings together to get a string of 156 symbols, we can now compare the two long strings. We point out, that in short strings of few symbols, the difference between nearly random and actual strings is difficult to discern. Figure 2 provides a graphical comparison between these two concatenated strings.

**Figure 2: Comparison of Two Strings of 156 Symbols, the First Random and the Second Actual Data**

There are some interesting results portrayed in Figure 2. Both strings evidence few short implicants, but when the implicants get longer allowing more of the symbols to be determined, then they diverge significantly. The random string differs from the known string in count at IB = 4 by 18, which implies that the known string has smoothed out at this level while the random string is still pretty choppy. Also at the long IB of seven, the random string has almost no implicants at this level (a value of 2) while the known string has 10 such common long subsequences, again indicating a certain level of regularity or smoothness. The random string does not exhibit common subsequences at longer IB’s.

5. K Complexity and FI Strings
Now consider the requirements of Kolmogorov complexity and a Ruling as a measure of such complexity. Since every string that has a reducible IB from the original or prime implicants can be represented in a ruling with IB minimized, we have decided to use as a standard for comparative measure an IB of 2. This yields a ruling with the highest number of levels and the shortest length implicants. This implies that the measure $K(s)$ for any string will be:

$$K(s) \leq O((IB+1)|s|) \quad (11)$$

Since we have defined the complexity as the number of symbols within the ruling, the question of an effective computable function is altered. This does not alter Theorem 3.2.2, since we have relaxed the length representation making now Theorem 3.2.1 no longer true.

Within Section 3.3, the chain rule properties are changed. Rule 1 is no longer true, and is replaced by (11). Rule 2 is in fact true for a duplicated string. Rule 3 does not apply since FI is not a partially computable function as long as the string is finite. For infinite strings, FI becomes a partially computable function. See [3] for conditions under which the FI algorithm is not computable, essentially if the infinite string eventually becomes recursive, then it is computable. One can divide the infinite string up into two parts, some initial substring that is finite and the remaining portion which is infinite, then Rule 3 holds true. Rule 4 is also true. Given a ruling containing string $x$, then it may have no impact upon the ruling for string $y$.

The stated chain rule of Section 3.3 which is:

$$K(x, y) = K(x) + K(y|x) + O(\log(K(x, y)))$$

When considering Rulings as the complexity, then we can state that the complexity of two strings $x$ and $y$ found in a single ruling where the strings are factored simultaneously [3], [5] then the length of the new ruling is going to be in the worst case $K(x)+K(y)$ if the two strings $x$ and $y$ are dissimilar and in the best case $K(x) + K(y|x)$ where $K(y|x)$ will be smaller than $K(y)$ because certain of the implicants found in the ruling for $x$ will also be found in the ruling for $y$, and thus will not be duplicated within the same ruling. So we can rewrite this as:

$$K(x, y) = K(x) + K(y|x) \leq K(x) + K(y)$$

6. Conclusion

We have described a technique for obtaining the complexity of genome sequences with potential application into comparative applications, common substrings, etc. using a mathematical technique called Finite Inductive Sequences. We have shown a way to obtain a complexity number requiring the relaxation of the standard definition of Kolmogorov Complexity. By such a relaxation, we obtain a way to compare strings which are long and have no other means for comparison except using the string itself. We have also provided some examples to illustrate this application.

7. References


