Efficient Sequence Comparison Using Binary Codes

Hossein Kamel Rahimi

Abstract—In this paper, we propose an efficient way of finding the exact distance in sequence comparison by using Huffman coding method for alphabets with uniform symbol probabilities. The approach is proposed as a refinement for word pair comparison in D2 statistics, though it can readily be generalised. Two given sequences with identical lengths are encoded to Huffman binary codes by which we are able to calculate Hamming Distance using binary operations efficiently. This method is applied on D2 statistics to compare k-tuples faster than its original version. The evaluation on empirical sequences showed that the method is faster than original D2; especially, when re-using the encoded sequences which resulted in better performance.

Index Terms—Performance, experimentation, string and sequence processing, huffman encoding, hamming distance, D2 statistics, string comparison, binary codes, bitwise operation.

I. INTRODUCTION

Finding similarities and differences of bio-sequences such as DNA and amino acids is a fundamental problem in biological science. Vinga and Almeida argue that “Sequence analysis is a discipline that grew enormously in recent years in response to the overwhelming burst in data generated by molecular biology initiatives” [1]. As Bioinformatics is an essential ingredient for the success of human genome projects, powerful computer systems and sophisticated algorithms are required [2]. Sequence comparison has played a crucial role in better performance.

Among the more important aspects of sequence comparison is the determination of sequences which ‘almost match’ the query, those which lie within some neighbourhood of the perfect match. This paper aims to propose a heuristic software solution for rapid determination of approximate matches between word pairs in D2 statistics. By encoding the base symbols through fixed codes according to the application, we allow the comparison to be computed via bitwise operations. The library sequence to be search against and the query is encoded only on one run and the rest of the computation approximates similar matches based on the encoded sequence as many times as needed with variant queries.

This paper is organized as follow paragraphs. In the next section, we introduce the main ideas in Section II before introducing the method formally in Section III and Section IV. Results are presented in Section V, and conclusions in Section VI.

II. HUFFMAN CODING

The proposed approach relies on the well-known entropy encoding method Huffman coding which is a common technique for data compression [10]. Various Huffman applications have been proposed and the method is still widely used. Mcintyre and Pechura applied Huffman to test data over various source programs in four different languages where the data was read from files [11]. Nourani and Tehranipour used Huffman technique to reduce test data volume, power dissipation and pattern delivery time in the field of Integrated Circuits [12]. Also, Tang utilised Huffman encoding methods for a proposed method of multimedia encryption for vast amounts of MPEG video data [13]. Since efficiency and effectiveness are the key aims in bio-sequence comparison algorithms, the aim of the proposed method is to utilize Huffman Binary Encoding as a mechanism of rapidly identifying exact and approximate matches of two bio-sequences and comparing pairs of bio-sequences much quicker than previous approaches. Encoded in Huffman binary codes, a bio-sequence could be compared rapidly regardless of the sequence’s actual symbols because the core of the proposed algorithm, being Huffman Binary code, could be operated at machine bit level. Therefore, this research topic is seeking an accurate, fast, and efficient way of bio-sequence comparison which has been crucial in bioinformatics science for many years.

III. HUFFMAN BINARY ENCODING

According to Huffman, the symbol or sequence symbol associated with a given message is called the message code [10]. In Bioinformatics, sequence k-mers are considered the message code which are the members of a set of alphabetic characters

\[ \Sigma = \{A, C, G, T\} \]

Let \( P(i) \) be the probability of \( i \)-th message, then

\[ \sum_{i=1}^{N} P(i) = 1 \quad (1) \]
The basic type of the Huffman method is the ‘static type’, where the frequencies of message codes are constant; while in ‘dynamic type’, the frequencies change. Knuth [14] describes that “a Huffman tree with nonnegative integer weights can be represented in such a way that any weight can be increased or decreased”. In this preliminary study, it is assumed that occurrence probabilities of the k-mers throughout the sequence are identical in such a way that satisfies equation (1) as equation (2):

\[ P(A) = P(C) = P(G) = P(T) = \frac{1}{N}, \]

an assumption which is only mildly in error for most sequence data, and one that may be modified for high GC content sequences.

Having two bio-sequences \( S_1 \) and \( S_2 \) over the alphabet set \( \Sigma = \{A, C, G, T\} \), we assign static binary codes 00,01,10 and 11 to A,C,G and T respectively:

\[ S_1 = a_1a_2a_3...a_n \]
\[ S_2 = b_1b_2b_3...b_n \]
\[ ai \in \Sigma \] and \( bi \in \Sigma \]
\[ 1 \leq i \leq w \]

Message codes are

\[ A=00, C=01, G=10, T=11 \]

At this point, we are able to compute the Huffman code for a given sequence by substituting any k-mer to its correspondent message code. For instance, given sequence \( S_1 = \text{TCATG} \), the Huffman Binary code is \( H_1=1101001110 \).

### A. D2 Statistics

**Definition:** The D2 statistic is defined as the number of words of pre-specified length k between two given sequences \( A=(a_1,...,a_n) \) and \( B=(b_1,...,b_m) \) where \( A_i \) and \( B_i \) belonging to the alphabet set \( \Sigma \) of size \( |\Sigma| = N \) and also let \( f_a \) be the occurrence probability of letter ‘a’, though in this work uniform probability for each letter is considered as \( f_a = \frac{1}{N} \).

Also, it is assumed that each sequence is generated independently and identically (i.i.d) of the letters in alphabet \( \Sigma \) [8]. Let \( Y(i,j) \) be the k-word match indicator variable indicating if the words starting at position i in A and word at position j in B match [5],[16].

\[ D2(A,B) = \sum_{(i,j) \in I} Y(i,j) \]

where index set \( I = \{(i,j): 1 \leq i \leq n-k+1, 1 \leq j \leq m-k+1 \} \). Similarly, the approximate word match statistic

\[ D2(t) = \sum_{(i,j) \in I} Y(t)(i,j) \]

where \( Y_{ij,0} \) if \( 0 < t < k \) is the k-word match indicator variable allowing up to \( t \) mismatches.

### IV. Method

The fundamental aspect of the method is the encoding function in which each base is substituted with its corresponding message code. Bradford [16] used the idea of fixed binary codes to measure Levenshien [17] edit distance between a set of encoded strings with the respect to a given super-sequence, though the method is only reliable to the existence of the super-sequence. Although Bradford’s work does not address the performance, in particular, for empirical sequences, its concept is promising to be studied for wider string comparison projects. In this paper, the idea of encoding strings to binary codes is developed to compare sequence pairs in respect to the facts that

1. Bitwise operations are highly efficient by which total comparison between sequence pairs is performed fast and (see section V-A)

2. Since most of the computation cost is paid as sequences are encoded, and needs to be done only once, the comparison process could be repeated comparatively rapidly with little computation [16]. (see Section V-B)

Given two sequences with identical length \( N \), we aim to find the Hamming Distance [18] \( H_d \) between two Huffman Binary Codes associated to the sequences.

\[ s_1 = a_1a_2a_3...a_N \]
\[ s_2 = b_1b_2b_3...b_N \]

Let \( h_1 \) and \( h_2 \), the Huffman binary bit vectors of \( s_1 \) and \( s_2 \) and \( H_d \) (equation 3) is the Hamming Distance bit vector calculated from Exclusive OR operation of \( h_1 \) and \( h_2 \) and represents those positions where there are mismatches comparing \( h_1 \) and \( h_2 \).

\[ H_d = \text{XOR}(h_1, h_2) \]  

Despite true bits in \( H_d \) could be pronounced as distances between \( h_1 \) and \( h_2 \) bit vectors, it does not demonstrate Hamming Distance between \( s_1 \) and \( s_2 \), as each k-mer has been substituted with 2 bits and some substitutions involve more bits than others. In order to deal with this issue, we introduced normalized \( D \) as \( D_z \) (equation 4). We apply an OR operation on each adjacent bit-pair representing one k-mer

\[ Dz(i) = \text{OR}(D(i), D(i+1)) \text{ when } i = 2k \]

For computing \( D_z \) in an efficient way, bitwise operation is again required. Let \( D_z \) be the shifted version of \( D \) by one to the left and let \( Z = \text{OR}(D, D_z) \). Due to the differences between \( D \) and \( D_z \), then it could be claimed that

\[ Z(i) = \text{OR}(D(i), D(i+1)) \text{ where } 1 \leq i \leq N-1 \]

To satisfy equation (4), bits at indexes \( 2k+1 \) where \( 0 \leq k \leq N-1 \) positions in \( Z \) should be eliminated which here we call them noise bits; let \( \Pi \) a finite sequence of “10”s with the length equal to \( Z \).

\[ \Pi = 1010101010... \]

where

\[ |\Pi| = |Z| \]

and
\[ D_z = \text{AND}(Z, \top) \]

By counting true bits in \( D_z \), Hamming distance \( H_d \) between \( S_1 \) and \( S_2 \) is revealed:

\[ H_d(s_1, s_2) = \text{cardinality}(D_z) \]

In equation (4), cardinality \( D_z \) is the number of true bits found in \( D_z \). For example, considering \( S_1=\text{ACGTC} \) and \( S_2=\text{GCATG} \), we then have

\[
\begin{align*}
  h_1 & = 0001101101 \\
  h_2 & = 1001001110
\end{align*}
\]

Then

\[ D = \text{XOR}(h_1, h_2) = 1000100011. \]

and \( D \) is shifted to the left by one as

\[ D_s = 0001000110 \]

\[ Z = \text{OR}(D, D_s) = 1001100111 \]

Since the length of \( Z \) is 10, thus is defined as a finite sequence of “10”s with the identical length of \( Z \).

\[ \top = 1010101010 \]

To eliminate noise bits at odd index position

\[ D_z = \text{AND}(1001100111, 1010101010) = 1000100010 \]

Finally Hamming distance between two sequences \( s_1 \) and \( s_2 \)

\[ H_d(s_1, s_2) = \text{cardinality}(D_z) \]

By replacing their corresponding values

\[ H_d(\text{ACGTC,GCATG}) = \text{cardinality}(1000100010) = 3 \]

Hamming Distance of two sequences is 3 which was computed using bitwise operations.

A. Algorithm

**Definition:** Given a genome \( g=\{s_1, s_2, ..., s_n\} \) on an alphabet \( \sum = \{A, C, G, T\} \), \( \sum = 4 \), \( s \) is a sub-sequence of \( g \) with the length \( |N| \) where \( 1 \leq i \leq n \) and

\[ P(A) = P(C) = P(G) = P(T) = \frac{1}{4} \]

and let bit vector \( h \) as Huffman code where each alphabet in \( \sum \) is encoded to binary codes as \( A \leftarrow 00 \), \( C \leftarrow 01 \), \( G \leftarrow 10 \) and \( T \leftarrow 11 \). Also, let \( D \) and \( D_z \) be bit vectors - the ‘Distance Vector’ and ‘Normalized Distance Vector’ respectively.

**HuffmanEncoder**(Sequence \( s \))

\[ k \leftarrow 1; i \leftarrow 1; N \leftarrow \text{Length of } s; \]

**While** \( k \leq N \)

\[ D(i) \leftarrow D(i+1) \leftarrow \text{binary Code of } s(k); \]

\[ k \leftarrow k+1; i \leftarrow i+2; \]

**EndWhile**;

**EndEncode**;

\[ h_s \leftarrow \text{HuffmanEncoder}(s_s); \]

\[ h_1 \leftarrow \text{HuffmanEncoder}(s_1); \]

\[ D \leftarrow \text{XOR}(h_s, h_1); \]

\[ i \leftarrow 1; j \leftarrow 1; N \leftarrow |D|; \]

**While** \( i \leq N \)

\[ D(i, j) \leftarrow \text{OR}(D(i), D(i+1)); \]

\[ i \leftarrow i+2; j \leftarrow j+1; \]

**EndWhile**

\[ k \leftarrow 0; \]

**While** \( k \leq N \)

\[ \top(k) \leftarrow ^\prime1^\prime; \]

\[ \top(k+1) \leftarrow 0^\prime; \]

**EndWhile**

\[ D_s \leftarrow \text{AND}(D, \top); \]

**Return** \( D_s \).cardinality();

V. EXPERIMENTS

Experimental evaluations were performed using custom implementations to evaluate the speed of the proposed approach. Given the library sequence to be searched against and the query to be searched for, let \( \lambda(s) \) and \( \lambda(q) \) be the elapsed CPU time for encoding \( s \) and \( q \) to fixed binary codes respectively. Furthermore, let \( \gamma(s, q) \) be the required CPU time to search the encoded query \( q \) over the encoded library sequence \( s \), then denote total execution time

\[ \Delta = \lambda(s) + \lambda(q) + \gamma(s, q) \tag{5} \]

In this section, \( k \)-tuple comparison is performed using string comparison and the proposed approach to compute D2 total execution time approach (section A). Then CPU time will be recorded for the proposed comparison approach over encoded library sequence and the result is evaluated with section B. Finally, the effect of various \( k \) over CPU time and accuracy will be demonstrated (section C).

A. D2 String Comparison vs. Proposed Approach

In this section, the D2 value between the encoded library sequence \( s \) with the length \( 50 \leq N \leq 50,000 \) and query with the length 300 and word size \( k = 3 \) are computed to examine the efficiency of the proposed approach by capturing total execution time \( \Delta \). One common approach to find the dissimilarity between \( k \)-tuples in D2 statistics is to take the minimum of all window distances for each pair \( W = (W(L),W(Q)) \), where \( W(L) \) and \( W(Q) \) are the \( k \)-tuples in library sequence and query sequence respectively [19]. In Fig. 1, we used the proposed binary method and string comparison to compare \( k \)-tuples and recorded total execution time for each sequence lengths in regard to equation (5).

B. Efficiency over Encoded Library Sequences
Several query searches could be performed over a given library sequence which has been already encoded in section A to address reusability aspect of this approach rather than running the encoding procedure several times over the same sequence. Hence, in equation (5), let $\lambda(s) = 0$ and denote new total execution time

$$\Delta^* = \lambda(q) + \gamma(s, q)$$  \hspace{1cm} (6)$$

Equation (6) demonstrates that encoding time for queries $\lambda(q)$ should be included as queries could be different. In this experiment, different queries with identical length 50 have been selected in order to have the same impact on query encoding time $\lambda(q)$ in (6) with respect to the total time $\Delta$ analysis with uniform data characteristics. In regard to Table I, the figures show that this approach could be even more efficient while re-using the encoded library sequence over various query searches.

<table>
<thead>
<tr>
<th>TABLE I: THE PROPOSED APPROACH TOTAL EXECUTION TIME (MS) BEFORE ENCODING $\Delta$ AND AFTER ENCODING $\Delta^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N=50$</td>
</tr>
<tr>
<td>$\Delta$</td>
</tr>
<tr>
<td>$\Delta^*$</td>
</tr>
</tbody>
</table>

Comparing Fig. 1 and Table I, the performance of the approach is affected whether encoder is included (Fig. 1) or excluded (Table II) despite the fact that both scenarios are faster than the regular approach. Table II also demonstrated that reusing the encoded library sequence could decrease the computation cost dramatically.

### C. K-Selection

<table>
<thead>
<tr>
<th>TABLE II: $N=1000$ AND $Q=300$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K=3$</td>
</tr>
<tr>
<td>$\Delta^*$</td>
</tr>
<tr>
<td>$D2$</td>
</tr>
<tr>
<td>INDEX</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TABLE III: $N=10,000$ AND $Q=1000$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K=3$</td>
</tr>
<tr>
<td>$\Delta^*$</td>
</tr>
<tr>
<td>$D2$</td>
</tr>
<tr>
<td>INDEX</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TABLE IV: $N=100,000$ AND $Q=3000$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K=3$</td>
</tr>
<tr>
<td>$\Delta^*$</td>
</tr>
<tr>
<td>$D2$</td>
</tr>
<tr>
<td>INDEX</td>
</tr>
</tbody>
</table>

### VI. Conclusions

In this paper, an efficient method for sequence pair comparison using Huffman coding was proposed. The genome sequences were viewed as text documents and applying fixed binary codes to the base symbols allowed the comparison to be performed with the help of bitwise operations, which enhances the performance. This simple approach appears to perform well over a wide range of sequences and conditions and offers superior performance to commonly used methods. It was also shown that the exact distance degree of sequence pairs could be achieved using this approach. The experiments showed that applying this approach could speed up original D2 statistics. In addition, re-using encoded sequences would decrease execution time dramatically so that the total computation focuses on the comparison of binary codes rather than time-consuming encoding procedure.

Several attempts might be done to improve the core of the proposed approach in future. Firstly, storing and re-using the encoded sequences is the major step which helps to decrease total execution time; instead of parsing the actual sequences.
Since the encoded sequences are in binary format, less space is required for storage, yet an efficient methodology for storing and retrieving the binary codes is needed. Moreover, this approach only works for alphabet sets with uniform symbol probabilities. Since many bio-sequence comparisons are performed with different symbol probabilities, the encoding procedure is required to be improved for further applications.

ACKNOWLEDGMENTS

The authors would like to thank Dr. James M. Hogan for the comments and providing sample data.

REFERENCES


Hossein Kamel Rahimi finished his bachelor degree in software engineering in Azad University of Iran in September 2006 and he graduated with the master degree of IT with the major of software architecture at Queensland University of Technology in Brisbane, Australia in December 2013. During his studies, he was awarded “Certificate of Dean’s List for Excellent Academic Performance” in July 2013 and December 2013. He was also recognized as the top student for his academic performance from Golden Key International Honour Society in July 2013. He is interested in software optimization and bioinformatics, particularly, in sequencing and sequence comparisons of genomes.