I.J. Image, Graphics and Signal Processing, 2015, 12, 18-23
Published Online November 2015 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijigsp.2015.12.03

A New Heuristic Approach for DNA Sequences Alignment

M. I. Khalil
Nuclear Research Center, Atomic Energy Authority, Cairo, Egypt. Currently in a sabbatical leave as an Associate Prof. at Princess Nora Bint Abdurrahman University, Faculty of Computer and Information Sciences, Networking and Communication Dept., Riyadh, Kingdom of Saudi Arabia, Riyadh
Email: magdi_nrc@hotmail.com

Abstract—The problem of comparing DNA sequences is one of the most significant tasks in the field of computational biology. It helps locating the similarities and differences between pairs of DNA sequences. This task can be achieved by finding the longest common substrings between DNA sequences and consequently aligning them. The complexity of this task is due to the high computational power and huge space consuming. Comparing DNA sequences leads to infer the cause of a certain disease beside many significant biological applications. This paper introduces a new Heuristic Approach for DNA Sequences Alignment between two DNA sequences. The new approach is based on three processing phases: the first phase finds the multiple common substrings in the two sequences, the second one sorts the obtained common substrings descending according to their lengths, and the last phase generates the optimal two aligned sequences. The modules of the new approach have been implemented and tested in C# language under Windows platform. The obtained results manifest a reduction in both time of processing and memory requirements.

Index Terms—DNA similarity algorithms, DNA sequence comparison, DNA analysis, pattern recognition, Longest Common Substring, Longest Common Subsequence, DNA sequences alignment.

I. Introduction

DNA (deoxyribonucleic acid) and RNA (ribonucleic acid) play fundamental roles in carrying genetic information in all living things on Earth. These structures are double helixes that look like a twisted ladder in which the rungs of the ladder consist of match pairs of nucleobases [1, 16]. DNA and RNA have great chemical similarities. In their primary structures both are linear polymers (multiple chemical units) composed of monomers (single chemical units), called nucleotides. Cellular RNAs range in length from less than one hundred to many thousands of nucleotides. Cellular DNA molecules can be as long as several hundred million nucleotides [2] (Fig.1). The term DNA sequencing encompasses biochemical methods for determining the order of the nucleotide bases, adenine, guanine, cytosine, and thymine, in a DNA oligonucleotide. The sequence of DNA constitutes the heritable genetic information in nuclei, plasmids, mitochondria, and chloroplasts that forms the basis for the developmental programs of all living organisms. Determining the DNA sequence is therefore useful in basic research studying fundamental biological processes, as well as in applied fields such as diagnostic or forensic research. Because DNA is key to all living organisms, knowledge of the DNA sequence may be useful in almost any biological subject area. For example, in medicine it can be used to identify, diagnose and potentially develop treatments for genetic diseases. Similarly, genetic research into plant or animal pathogens may lead to treatments of various diseases caused by these pathogens [3].

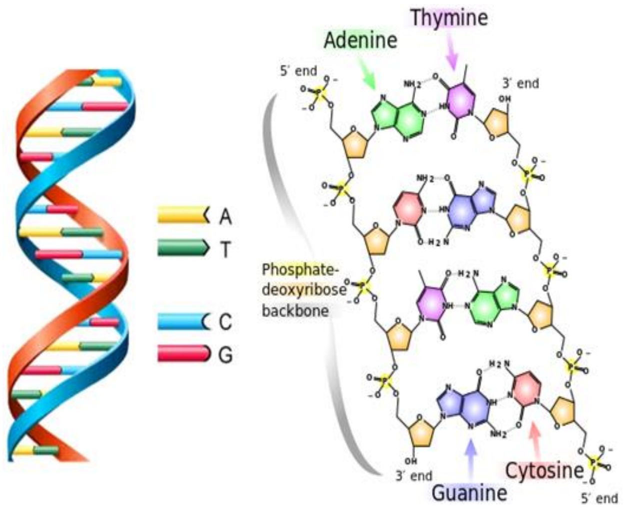


Fig.1. DNA Helix

In comparative genomics, comparing genome sequences is one of the main tasks because sequence similarities strongly reflect the evolutionary relationships between the corresponding species. In addition, with the introduction of next-generation sequencing technologies, the demand for rapid comparisons of massive amounts of long sequences has increased in recent years [4,5-11,15]. Improvements in the efficiency of DNA sequencing have both broadened the applications for sequencing and dramatically increased the size of sequencing datasets. For years, DNA comparison has been used in biology and forensics to discriminate and compares genes or genomes. Those tools vary in size, complexity and functionality based on several factors. Some small tools or websites are developed as free or open source for research or experimental purposes. Examples of such small size limited purpose tools or applications are: Double Act (http://www.hpabioinfotools.org.uk/pise/double_act.html), Genomatix (http://www.genomatix.de), Mobyle (http://mobyle.pasteur.fr), ALIGN, FASTA, etc. BLAST: (Basic Local Alignment Search Tool) [12,13] is an example of a larger scale. Most of these algorithms uses Smith–Waterman algorithm for performing sequence alignment. This algorithm which is also used in crimes' forensic investigation does not use full DNA to DNA sequence comparison. It rather selects several segments (e.g. eight segments) selected from the different locations of the DNA. BLAST uses also dynamic programming and "seeding" to find starts of possible matches. The goal is to accelerate the process of finding matches between DNA sequences as this can take a significant amount of time and resources. Another process that can be different from one tool to another is the ranking of the different matches. This can particularly occur when more than a match is in the same size [12,17].

The current paper addresses the problem of finding a sequence alignment between two DNA sequences, in which individual similar letters from each sequence are placed into correspondence. It is based on a heuristic approach including three processing phases: the first phase finds the longest common substrings in the two sequences, the second one sorts the obtained common substrings descending according to their lengths, and the last phase generates the optimal two aligned sequences.

The rest of the paper is organized as follows. Section II discusses the suggested algorithm. The implementation and experimental results are discussed in Section III. The work done in this paper has been concluded in Section IV.

II. The Suggested Approach

The aim of the suggested algorithm is to align homologous regions separately keeping their order in both DNA sequences.

Given two DNA sequences:

$$X = x\_(0)\ x\_(1)\ x\_(2) \ldots\ldots x\_(m-1) \qquad (1)$$

$$Y = y\_(0)\ y\_(1)\ y\_(2) \ldots\ldots y\_(n-1) \qquad (2)$$

Where X, Y are two sequences of length m-1, n-1 respectively, and x_i and y_i are chosen from a finite alphabet A, e.g. {A, C, G, T}:

$$x\_i \in \{A, C, G, T\}, \qquad y\_i \in \{A, C, G, T\} \qquad (3)$$

The goal is defining the distance between the sequences X and Y, or alternatively their similarity. Where similarity is defined with reference to a sequence alignment, in which individual letters from each sequence are placed into correspondence. Achieving this goal, the suggested approach has been divided into three consecutive modules:

Module I: An algorithm for finding all common substrings between two sequences and it has previously published [14, 18]. In this module, two DNA sequences are compared to find all possible identical matches between them. It is based on the convolution between the two sequences.

The data structure required for the suggested algorithm is shown in Fig.2. The major DNA sequence string is represented in the linked-list X while the minor one is represented in circular linked-list Y. Each element of the linked-list Y contains a data filed in addition to a single directional pointer to the next element in a circular manner. Each data field of Y holds one of the minor DNA sequence characters. Each element of the linked-list X consists of three fields described as follows. The first one is a data field holding one of the major DNA sequence characters. The second field is a pointer to an independent linked-list, where the collection of those independent linked lists are grouped and considered as array of linked-list Z. The array Z will be dedicated to hold the resultant of the matching algorithm as will be illustrated later. The third filed is simply a pointer to the next element of the linked-list X.

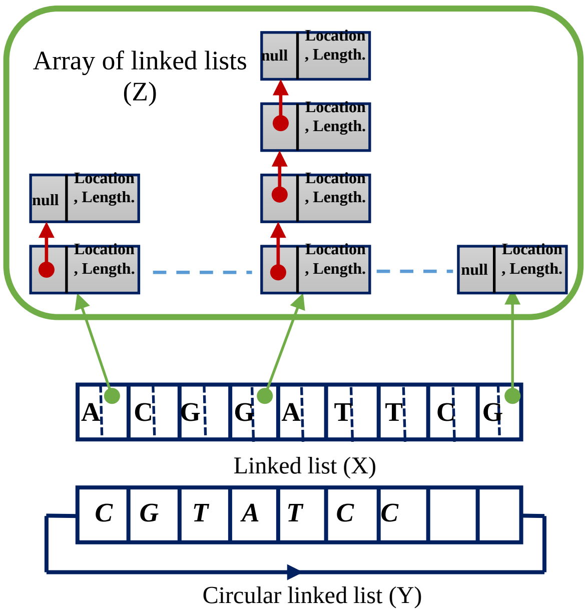


Fig.2. The Data Structures used in the Suggested Approach

The matching algorithm aims to determine both the location and length of all possible common substrings between the two DNA sequences represented in X and Y linked lists respectively. The main matching algorithm and its subroutines are listed in List. 1 through List.3.

Referring to List. 1, when the two variables x and y are equal, the process named *Get_substring_until_no_match* (List 2) will be performed continuously reading new pair of characters as long as the new pair of characters x and y are equal otherwise the process terminates. The length and location of the obtained substring are considered as input to the process named *Add_obtained_substring_to_Z* (List 3).

List.1: Pseudo code of ***Matching Algorithm***

```
//    Input: DNA1 and DNA2 sequences
Get Length_of_DNA1;
Get Length_of_DNA2;
For I = 0; I < Length_of_DNA1
    For j = 0; J < length_of_DNA2
       x = read character from DNA1[I]
       y = read character from DNA2[J]
       if ( x == y)
       {
         Perform Get_substring_until_no_match(I,J);
         // L = length of the obtained substring
         Add_ obtained_substring_to_Z(I,J, L);
        }
    Next J;
Next I;
```

List.2: Pseudo code ***Get_substring_until_no_match***

```
Input: I, J
m =I;
n = J;
Length =0;
do
{
     x  = read character from DNA1[m]
     y  = read character from DNA2[n]
     Length ++;
     m++;
     n++;
 }
 while (x == y)
Return Length
```

List.3: Pseudo code ***Add_obtained_substring_to_Z***

```
Input:  I, J, Length
m =I;
n = J;
l = Length;
if X[m].pointer == null
   {
      create new linked_list Z[m]
      add new cell with ( n, l )
       //   n = location of substring in DNA2 sequence
       //   l = length of substring
   }
   Else
      Add new cell to linked_list Z[m]
```

The ***Add_obtained_substring_to_Z*** process is responsible of adding information related to the obtained substring to the array of linked lists Z. Each character of the major DNA sequence, which is represented by X[I], points to a separate linked list in the array Z. The initialization process does not create the linked lists in array Z but leave this task for process **Add_obtained_substring_to_Z** to create only the actually needed linked lists to save the allocated memory space. Each separate linked list Z[I] should hold information about all common substrings between sequence X, starting at position I, and sequence Y. Accordingly, the process adds a new node to the corresponding linked list Z[I] writing both the length of the substring (Length**)**, and its location (J) in the second sequence Y to the corresponding fields of this node.

By the end of the matching process, some characters of the DNA sequence X have linked lists in the array Z and the others do not have.

The following example illustrates the matching process of the suggested algorithm:

Input: two sequences:
DNA-1 sequence = “ATCAGTTACGT”
DNA-2 sequence = “TATCATG”

The two sequences are placed in linked lists X and Y respectively. The matching process yields the array Z of the linked lists (Red boxes in Fig.3). For example, the first character at position 0 of the first sequence (“A”) has two matches with the second sequence at locations 1 and 4 with lengths 4 and 2 respectively. The first matching has length of 4 where the substring “ATCA” exists in both sequences. The second matching has length of 2 where the substring “AT” exists in both sequences.

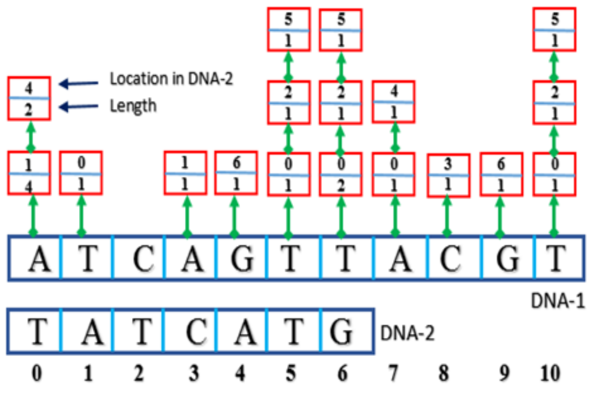


Fig.3. Illustration Example of the Matching Process

Inspecting the linked lists produced by the matching process leads to finding that the longest common substring is “ATCA” at location 0 of the first sequence and location 1 of the second sequence.

Module II: Module I generates a list of the common substrings between DNA sequences X and Y along with their lengths and their locations in both X and Y sequences. In Module II, this list is sorted descending according to the lengths of the obtained common substrings, location in X and location in Y respectively.

Module III: the algorithm of this module attempts to infer which positions within both sequences are homologous. The output of module II is passed as input to this module where individual letters from each DNA sequence are placed into correspondence. To explain the algorithm behind module III let us begin with the following example:

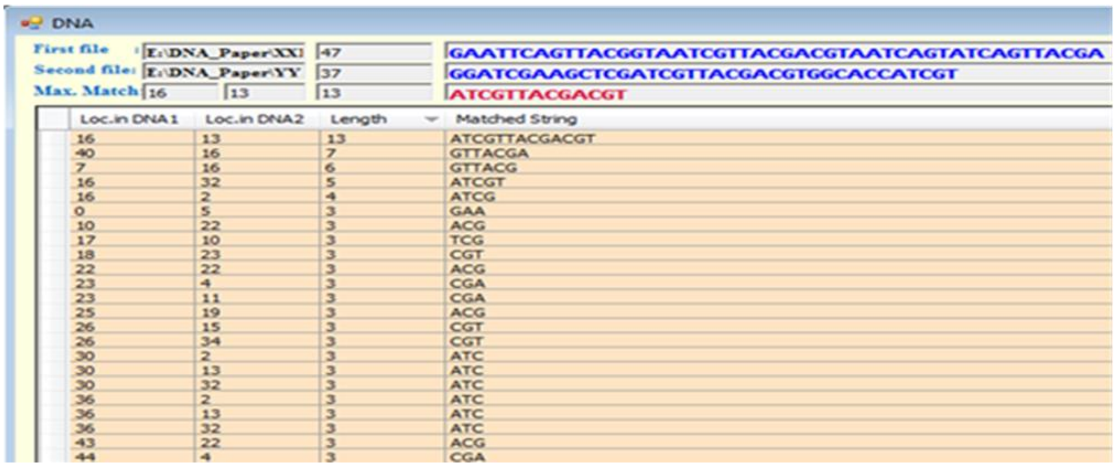
Sequence-1:

GAATTCAGTTACGGTAATCGTTACGACGTAATCAGTATCAGTTACGA

Sequence-2:

GGATCGAAGCTCGATCGTTACGACGTGGCACCATCGT

Applying module I to both sequences yields several common substrings and sorting these substrings (applying module II) yields the list partially shown in Fig.4.



DNA

First file : E:\DNA_Paper\XX 47 GAATTCAGTTACGGTAATCGTTACGACGTAATCAGTATCAGTTACGA

Second file: E:\DNA_Paper\YY 37 GGATCGAAGCTCGATCGTTACGACGTGGCACCATCGT

Max. Match 16 13 13 ATCGTTACGACGT

Loc.in DNA1	Loc.in DNA2	Length	Matched String
16	13	13	ATCGTTACGACGT
40	16	7	GTTACGA
7	16	6	GTTACG
16	32	5	ATCGT
16	2	4	ATCG
0	5	3	GAA
10	22	3	ACG
17	10	3	TCG
18	23	3	CGT
22	22	3	ACG
23	4	3	CGA
23	11	3	CGA
25	19	3	ACG
26	15	3	CGT
26	34	3	CGT
30	2	3	ATC
30	13	3	ATC
30	32	3	ATC
36	2	3	ATC
36	13	3	ATC
36	32	3	ATC
43	22	3	ACG
44	4	3	CGA

Fig.4. Sorted List of the Common Substrings According to Module II

We can geometrically (Fig.5) represent any common substring S with length l as:

$$S = \{(x1, y1), (x2, y2)\} \tag{4}$$

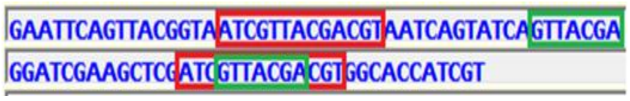
Where:
x1 = position of the substring in the first sequence X
y1 = position of the substring in the second sequence Y

$$x2 = x1 + \ell \tag{5}$$

$$y2 = y1 + \ell \tag{6}$$

Considering pair of the common substrings in the current example:

The first one is "*ATCGTTACGACG*" has length 13 and starts at position 16 in the first sequence and at position 13 in the second sequence. Geometrically, we can define this substring as: *S1{(16,13),(29,26)}*. The second one is "*CGTTACGA*" with length 7 and starts at position 40 in the first sequence and at position 16 in the second sequence. And can be defined geometrically as: *S2 {(40, 16), (47, 23)}*. So, We have two choices of alignment; either "ATCGTTACGACG" of the first sequence with its correspondence in the second sequence or "CGTTACGA" of the first sequence with its



GAATTCAGTTACGGTAATCGTTACGACGTAATCAGTATCAGTTACGA

GGATCGAAGCTCGATCGTTACGACGTGGCACCATCGT

correspondence in the second sequence. However, we cannot make alignment of both of them, as the second substring is included in the first substring of the second sequence. To avoid this conflict the choice should be subject to the following condition: if two substrings belong to the same sequence, we choose both of them if and only if the following four conditions are satisfied and assuming that *S1* is longer than *S2*:

$$S2..\ x1 \quad \notin [S1.x1, S1.x2], \tag{7}$$

$$S2..\ x2 \quad \notin [S1.x1, S1.x2], \tag{8}$$

$$S2..\ y1 \quad \notin [S1.y1, S1.y2], \tag{9}$$

$$S2..\ y2 \quad \notin [S1.y1, S1.y2] \tag{10}$$

Or simply omit all substring in zone A and zone B as illustrated in Figure.5.

Applying the last rule to the sorted list of common substrings in the current example yields the short list shown in Fig.6.

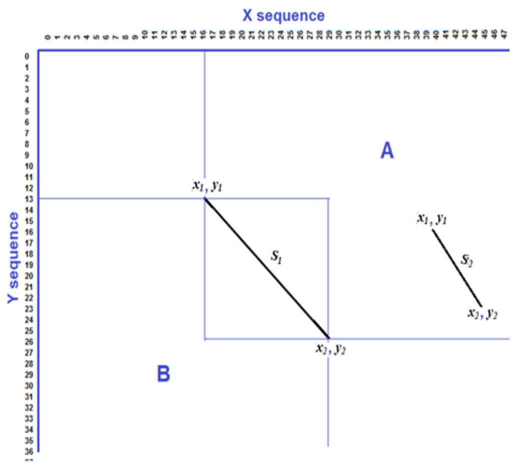
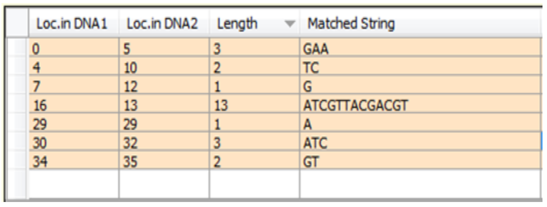


Fig.5. Geometrical Representation of Common Substrings

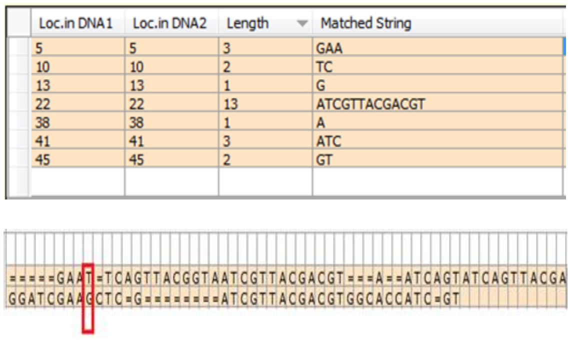


Loc.in DNA1	Loc.in DNA2	Length	Matched String
0	5	3	GAA
4	10	2	TC
7	12	1	G
16	13	13	ATCGTTACGACGT
29	29	1	A
30	32	3	ATC
34	35	2	GT

Fig.6. Short List of the Common Substrings

The first phase of the alignment algorithm is listed below.

The input to this phase is the short list previously obtained (Fig.6). It is based on comparing the positions of each common substring in both sequences. Getting the distance between the two positions, $n = S1.x1 - S1.x2$ then inserting number n of "=" in front of the substring with the little position in the hosting sequence until the two positions become equal. This action leads to increasing the position value of each subsequent substring in the short list. Fig.7 shows the two aligned sequences and the updated positions in the short list.



Loc.in DNA1	Loc.in DNA2	Length	Matched String
5	5	3	GAA
10	10	2	TC
13	13	1	G
22	22	13	ATCGTTACGACGT
38	38	1	A
41	41	3	ATC
45	45	2	GT

```
=====GAAT=TCAGTTACGGTAATCGTTACGACGT===A==ATCAGTATCAGTTACGA
GGATCGAAGCTC=G========ATCGTTACGACGTGGCACCATC=GT
```

Fig.7. The Preliminary Aligning of the two Sequences

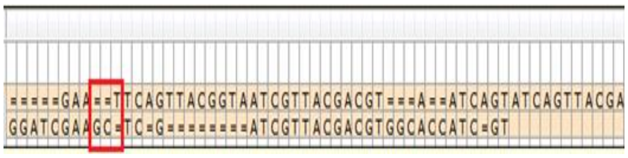
List.4: ***Alignment algorithm***

```
// input : short list of the common substring

S1.filler ="";
S2.filler ="";
foreach substring in the short list
{
        temp = "";
        x1  =  position of the substring in the major sequence ;
        y1  =  position of the substring in the minor sequence ;
        x =  x1   + S1.filler;
        y =  x1   + S2.filler;

         if (x < y)
        {
            S1.filler +=  y - x;
            for (int i = 0; i < y - x; i++)
                temp = temp + "=";
            S1= S1.Insert(x, temp);
            x =+ temp.Length;
         }
         else if (y < x)
         {
            S1.filler += x - y;
            for (int i = 0; i < x - y; i++)
                temp = temp + "=";
            S2= S2.Insert(y, temp);
            y =+ temp.Length;
         }
}
```

As shown in Fig.7 there is an exception case where the two characters "T" and "G" are unaligned. This case occurs when the sum of substring's position value plus the length of this substring is less than the position of the next substring in the same sequence. To overcome this situation, another procedure should be performed to detect the similar cases inserting number of "=" in the proper positions in both sequences (see Figure 8)

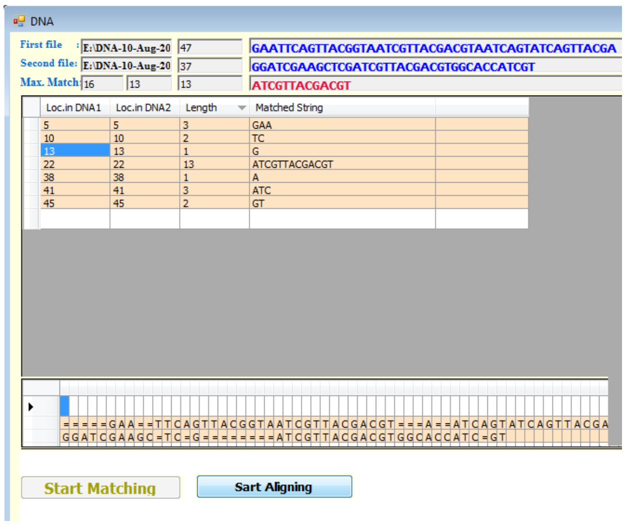


```
=====GAA==TTCAGTTACGGTAATCGTTACGACGT===A==ATCAGTATCAGTTACGA
GGATCGAAGC=TC=G=========ATCGTTACGACGTGGCACCATC=GT
```

Fig.8. The two Sequences After Complete Aligning

III. Implementation

The suggested system consists of two main algorithms, the matching algorithm and aligning algorithm respectively. The first one handles the process of finding the common substrings between two DNA sequences. It generates a list of common substrings along with their lengths and locations in both DNA sequences. The second algorithms processes the generated information from the first one yielding two new DNA sequences where all possible homologous substrings are aligned together (Fig.9). The system has been designed and implemented using C# language.



First file :	E:\DNA-10-Aug-20	47	GAATTCAGTTACGGTAATCGTTACGACGTAATCAGTATCAGTTACGA
Second file:	E:\DNA-10-Aug-20	37	GGATCGAAGCTCGATCGTTACGACGTGGCACCATCGT
Max. Match:	16	13	13	ATCGTTACGACGT

Loc.in DNA1	Loc.in DNA2	Length	Matched String
5	5	3	GAA
10	10	2	TC
13	13	1	G
22	22	13	ATCGTTACGACGT
38	38	1	A
41	41	3	ATC
45	45	2	GT

```
=====GAA==TTCAGTTACGGTAATCGTTACGACGT===A==ATCAGTATCAGTTACGA
GGATCGAAGC=TC=G=========ATCGTTACGACGTGGCACCATC=GT
```

Fig.9. Snapshot of the Program

IV. Conclusion

The algorithm proposed in this paper addresses the problem of locating the longest homologous substrings in two different sequences and consequently generates two new aligned DNA sequences. It is based on the convolution between the two DNA sequences (named major sequence *X* and minor one *Y*) and creating a node for each matched substring between the two sequences. If two or more matches share the same location in string *X*, the corresponding nodes will construct a single linked-list yielding a group of linked-lists containing nodes arranged in certain manner representing all possible matches between sequences *X* and *Y*. A list of the common substrings are then constructed, sorted and got rid of repeated ones. The short list of the common substrings are then more processed yielding the aligned sequences with the priority given to the longest common substring. The prposed algorithm could be more developed to locate and align the longest common strings between the major DNA sequence and several minor sequences. Moreover, the algorithm needs to be developed to be able to run in parallel processing manner to cope with the long time processing problem.

References

[1] Http://www.astrochem.org/sci/Nucleobases.php, retrieved on 8th, june 2015.

[2] Harvey Lodish, Arnold Berk, S Lawrence Zipursky, Paul Matsudaira, David Baltimore, and James Darnell., Molecular Cell Biology, 4th edition, New York: W. H. Freeman; 2000., ISBN-10: 0-7167-3136-3.

[3] S.Rajesh, S.Prathema and.L.S.S.Reddy, “Unusual Pattern Detection in DNA Database Using KMP Algorithm”, International Journal of Computer Applications (2010) (0975 - 8887) Vol. 1 – No. 22.

[4] Kyohei Yamaguchi and Satoshi Mizuta, “A New Graphical Representation of DNA Sequences Using Symmetrical Vector Assignment”, Review of Bioinformatics and Biometrics (RBB) Volume 3, 2014.

[5] O. Gotoh, “An Improved Algorithm for Matching Biological Sequences,” Journal of Molecular Biology, 162, pp: 705~708, 1982.J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68-73.

[6] S. Grier, “A tool that detects plagiarism in Pascal programs”, ACM SIGCSE Bulletin, vol. 13, no. 1, (1981), pp. 15-20.

[7] J. A. W. Faidhi and S. K. Robinson, “An empirical approach for detecting program similarity within a university programming environment”, Computers & Education, vol. 11, no. 1, (1987), pp. 11-19.

[8] U. Manber, “Finding similar files in a large file system[C/OL]”, In: Proceedings of the Winter USENIX Conference, (1994), pp. 1-10.

[9] BLAST, http://blast.ncbi.nlm.nih.gov/Blast.cgi, (2011) September.

[10] C. Yu, S.-Y. Cheng, R. L. He and S. S. -T. Yau, “Protein map: An alignment-free sequence comparison method based on various properties of amino acids”, Gene, vol. 486, (2011), pp. 110-118.

[11] Y. Guo and T. -m. Wang, “A new method to analyze the similarity of the DNA sequences”, Journal of Molecular Structure: THEOCHEM, vol. 853, (2008), pp. 62–67.

[12] Izzat Alsmadi and Maryam Nuser, “String Matching Evaluation Methods for DNA Comparison”, International Journal of Advanced Science and Technology, Vol. 47, October, 2012.

[13] BLAST, http://blast.ncbi.nlm.nih.gov/Blast.cgi, (2011) September.

[14] M.I.Khalil, Finding Longest Common Substrings in DNA Sequences, IJITCS, 2015.

[15] J. K. Me, M. R. Panigrahi, G. N. Dash and P. K. Meher, Wavelet Based Lossless DNA Sequence Compression for Faster Detection of Eukaryotic Protein Coding Regions, IJIGSP Vol.4, No.7, July 2012.

[16] Mohammed Abo-Zahhad, Sabah M. Ahmed and Shimaa A. Abd-Elrahman. A Novel Circular Mapping Technique for Spectral Classification of Exons and Introns in Human DNA Sequences, IJITCS Vol. 6, No. 4, March 2014, PP.19-29.

[17] G. Sethuraman,Kavitha Joseph, Star Coloring Problem: The DNA Solution, IJITCS Vol. 4, No. 3, April 2012, PP.31-37.

[18] M.I.Khalil, M.A.Hadi, Finding Longest Common Substrings in Documents, IJIGSP Vol. 7, No. 9, 2015, 9, 27-33.

Authors’ Profile

Dr. Magdi Ibrahim Khalil El-Sharkawy, Egyptian, male, has obtained his B.Sc degree in Computer and Automatic Control Engineering from Faculty of Engineering, Ain Shams University, Cairo, Egypt, in 1983, M.Sc degree in Computer Engineering from Faculty of Engineering, Tanta University, Tanta, Egypt, in 2003 and Ph.D degree in Computer Systems Engineering from Faculty of Engineering, Benha University, Cairo, Egypt, in 2005. He is currently working as Associate Professor in Department of Networking and Communication systems at the Faculty of Computer and Information Sciences, Princess Noura Bent Abdulrahman University, Riyadh, KSA. He has 15 years of previous experience at the Reactor Physics Department, Nuclear Research Center (NRC), Egyptian Atomic Energy Authority Cairo (EAEA), Egypt in the field of Data Acquisition and Interface Design. His main research interests focus on: Digital Signal Processing, Wireless Sensor Networks, Personal and Mobile Communications. So far, he has twelve years of teaching experience and has published more than twenty-five papers in repute journals and proceedings of conferences in fields of the data acquisition, digital signal processing, image processing and neural networks.