NCICIT 2013: 1st National Conference on Intelligent Computing and Information Technology, November 21, CUET, Chittagong-4349, Bangladesh

A Sequence Alignment Algorithm and Tools for Molecular Replacement

Mohammad Ibrahim Khan, Mehtanin Kabir Rashikh
Department of Computer Science & Engineering, Chittagong University of Engineering & Technology, Bangladesh
Email: muhammad_ikhancuet@yahoo.com; the.rashikh@gmail.com

Abstract—**This paper describes a new genetic alignment algorithm and software tool for sequences that can be used for determination of deletions and substitutions. Sequence alignment is one of the most active ongoing research problems in the field of computational molecular biology. Sequence alignment is important because it allows scientists to analyze protein strands (such as DNA and RNA) and determine where there are overlaps. This overlaps can show commonalities in evolution and they also allow scientists to better prepare vaccines against viruses, which are made of protein strands. The algorithm provides several solutions out of which the best one can be chosen on the basis of minimization of gaps or other considerations. The algorithm does not use similarity tables and it performs aspects of both global and local alignment. It is also compared with other sequence alignment algorithms.**

Keywords— Sequence Alignment; Genetic Algorithms; Computational Biology; Algorithm Complexity

I. INTRODUCTION

Sequence alignment refers to the problem of optimally aligning sequences of symbols with or without intersecting gaps between the symbols. The objective is to maximize the number of matching symbols between the sequences and also use only minimum gap intersection, if gaps are permitted. The sequence characters in bioinformatics can be any genic (gene sequence protein sequence), structural (morphological) or behavioral features of an organism. The DNA (deoxyribonucleic acid) sequence is a string of adenine (A), guanine (G), cytosine (C) and thiamine (T) bases and there are several approaches to the solution to the alignment problem of two DNA sequences. In general, the alignment we seek is not for all the bases of the fragment but rather of various parts of it at different locations. We may find different solutions where the parts of the fragment have different gaps associated with them [1]. Given any number of sequences, finding out the degree of similarity between them is highly helpful in the field of bioinformatics, because similar regions help in deciding structural and evolutionary relationship between the sequences.

When searching in database by strings, the problem can be of errors in spellings or variant spellings or even systematic errors due to misalignment of hands with the keyboard. In such cases a similarity matrix between the characters needs to be considered. Likewise, there can be alternate spellings and representation of heard utterances in a variety of ways. In diachronic examination of and language the question of alignment must deal with changes [2]. In biological memory, fragments may be matched by means of indices [3] or by shape [4], which are aspects of alignment that go beyond string matching topics.

At present, there are various sequence alignment algorithms to find the best alignment between two sequences. In general these algorithms perform either global or local alignment or a combination of the two. Global alignment is generally performed if the sequence lengths are comparable whereas local alignment is when one is looking for the best fit within the larger sequence of the small sequence. The Needleman-Wunsch [5] and the Smith-Waterman [6] algorithms are well known algorithms for finding the best alignment between sequences. These algorithms make use of dynamic programming to find the best alignment and they use similarity tables. Needleman-Wunsch performs global optimization, whereas Smith-Waterman calculates local alignments. For sequences of length m and *n*, the complexity of the two algorithms in their basic forms is O(mn).

II. METHOD

A. Alignment Algorithms

The best alignment between any two given sequences easily be found by brute force if their lengths are small. If the sequence lengths are large, we must develop a strategy to minimize the comparisons.

Basically, alignment methods perform global and local alignments. We will explain global alignment with an example consisting of two sequences X and Y.

```
X: GLKATKDNCKSSEBSEFDN
   |      | | | |    |
Y: GHGFLERNCKSLMRLEDAH
```

The alignment is stretched over entire sequence lengths to match as many matches possible. Although, NCKS is the biggest match that is possible between X and Y, G and E are also considered because of the match occurrence.

Local Alignment is an alignment that searches for segments of two sequences that match really well. Local alignment stops at the end of regions of similarity. It does not take the entire sequence into consideration. It just looks for regions that have more similar segments by neglecting regions with less similar segments.

Example for Local Alignment:

```
 X: ----------NCKS----------
              | | | |
Y: ----------NCKS----------
```

B. *Reliable Alignment*

Examination of structural alignments shows that the more similar the proteins, the larger the proportion of the structure that can be aligned. It follows rather obviously that if one only has sequences, then the more similar the sequences, the more reliable any alignment of those sequences is likely to be. This relationship between alignment reliability and sequence similarity has been quantified.

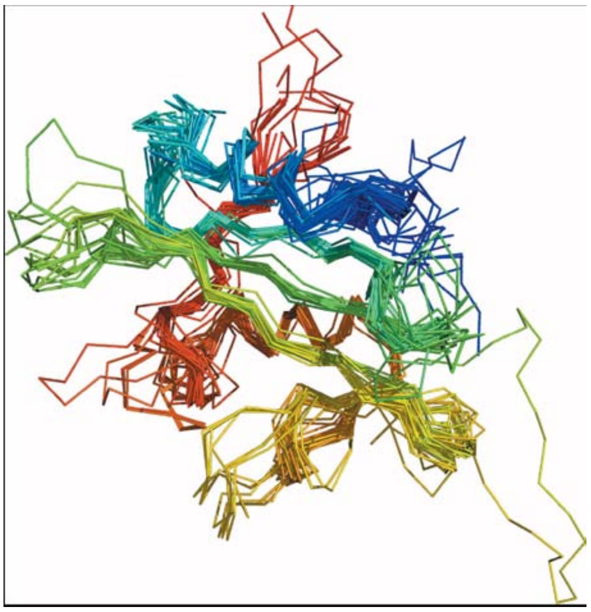


Figure 1. C_ representation of 27 SH2 Domain Structures Aligned Using the Program STAMP (Russell & Barton, 1992).

C. *Proposed Algorithm*

Our algorithm can be explained by taking two sequences into consideration. We represent our first sequence as X and second sequence as Y. Let the length of X be m and length of Y be n, In general $n \leq m$. In our example, we take Y to be shorter than X.

The algorithm works as follows:

```
Step 1: Set a=0
Step 2: Set ct1=0 and ct2=0
Step 3: If n-a ≠ 1, goto Step 4, Else goto Step 9
Step 4: Compare (n-a) size substring of Z with (m-(n-a)) substring of X
Step 5: If match found, record the positions of Y and X where match has occurred
Step 6: If (n-a) is less than n, Increment ct2, else set ct2=0 and goto Step 7
Step 7: If (m-(n-a)) is less than m, Increment ct1, else set ct1=0 and goto Step 8
Step 8: Increment 'a' by one and goto Step 3
Step 9: Print similarity regions recorded when match occurred and rest are filled with gaps(-).
```

D. *Example 1*

Let us consider X = CTFTALILLAGAG and Y = FTALLAAG. The length m of X is 13 and the length n of Y is 8 (where $m \geq n$). Since Y is shorter, we perform comparison of sequence Y with sequence X.

These are the comparisons that occur in the first iteration of our program:

FTALLAAG is compared with CTFTALIL, TFTALILL, FTALILLA, TALILLAG, ALILLAGA and LILLAGAG.

If match does not occur, a is incremented by one, so we perform (n-a) comparison of all possible strings in Y with all possible (m-(n-a)) length strings in X.

FTALLAA and TALLAAG in sequence Y are compared with CTFTALI, TFTALIL, FTALILL, TALILLA, ALILLAG, LILLAGA and ILLAGAG in sequence X.

We perform such comparisons in sequence till (n-a) becomes 1 or all character in shorter sequence Y gets match with sequence X.

Finally, we obtain alignment as:

```
X: C T F T A L I L L A G A G
Z:  - - F T A L - - L A – A G
```

The sequences as presented to us may have errors. These errors could be of different kind: substitution errors, extra characters, dropped characters. Clearly, one must correct for errors based on knowledge of the database from where the strings have come. The matching process can point to likely places of substitution, extra characters, and dropped characters.

Let us assume we have an error in sequence X. Let the actual value of X = CTFTALILLAAG obtained after error correction. Now, if we perform alignment between X and Y, we might get better alignment than what we got before. Likewise there could have been an error in the sequence Y. If errors are known to have occurred it is important to correct these errors.

In the above example, if we consider X after error correction then we get a better result than that of the alignment that we got before.

X_{new} = C T F T A L I L L A A G

Z_{new} = - - F T A L - - L A A G

Z_{new} has better alignment than Z after considering error correction.

If there exists more than one match between any two sequences, we must select an alignment that has minimal gap amongst them or use other considerations related to similarity between characters. Here we propose the following method to choose the better alignment: Generate all possible alignments between sequences and calculate the mean and variance for the gaps for each of those alignments. An alignment with less gap mean and smaller variance, if the means are the same, is the optimal alignment.

E. Example 2

Let us consider X= SNARSENAGCATQRABCRTLJT and Y= ARAGCATR. The possible alignments Z that we could get are given below:

X: S N A R S E N A G C A T Q R A B C R T L J T
Z: - - A R - - - A G C A T - R - - - - - - - -
- - A R - - - A G C A T - - - - - R - - - -

In the above example, we have 2 possible alignments for Z. The gap variance is computed after finding the mean of the gaps which are: 2 and 4 respectively. The variance values for the gaps are: 1 and 1 respectively. But still first one has minimum number of gaps, Therefore, the first alignment is the best one.

Some words in English dictionary, which are frequently confused with words that sound same, but are spelled differently. For example effected and affected, to and too, sent and cent, here and hear and many more. These words are called as homonyms. Same names are spelled in different ways in different regions or they are pronounced differently. The English “a” has the pronunciation of “ai” in many words. If we are able to derive a matrix that tells us which letters could be substituted with existing letters that could minimize the gaps between sequences, and our algorithm could then work significantly better.

F. Example 3

X=CGTCTAACTAGGTACAGTAGAG and Z=TACTAGGAG, so that m= 22 and n=9. Here are the possible alignments our algorithm generates.

X =C G T C T AA C T A G G T A C A G T A G A G
Z1= - - T - - - A C T A G G - - - A G - - - - -
Z2= - - - -T - A C T A G G - - - - - - - - - A G
Z3= - - - -T- A C T A G G - A - - G - - - - -
Z4= - - - - T- A C T A G G - - - - - - A G- -

There are many possible alignments that can be generated, Likewise, our algorithm generates all possible alignments and computes variance for all alignments. An alignment with less variance could be chosen as best alignment.

In the above example, mean of Z1, Z2, Z3 and Z4 is 4.67, 6.33, 3 and 4 respectively. Variances are 5.56, 14.89, 8.5 and 9.5. As alignment string Z1 has minimum variance, Z1 is considered as best alignment amongst Z1, Z2 Z3 and Z4. This strategy defines a new way of selecting optimal alignment between given any number of sequences. When we perform alignment between any sequences, our main objective is to minimize gaps between them. So we have to select optimal alignments by discarding all non-optimal alignments that would create more gaps. When given sequences are long, we need to use dynamic programming to calculate best possible alignment. But in the beginning, it is also necessary to prune bad alignments.

G. Flow Chart of Software Tool

The software tool that is developed by Java language by using the algorithm, works by following this process flow chart.

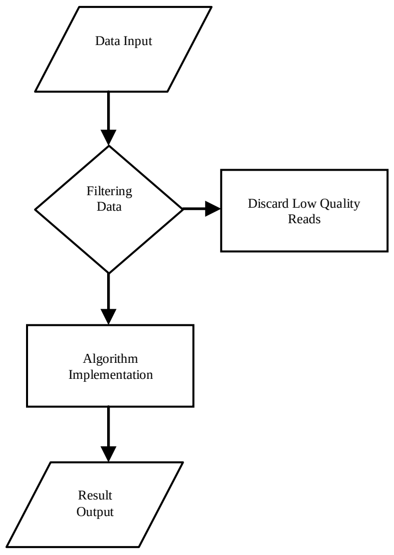


Figure 2. Flow Chart of the Process of Developed Software Tool.

III. Complexity Analysis

For sequences X and Y of length m and n respectively, the proposed algorithm performs following number of comparisons to calculate alignment between them where k is an integer that ranges from 1 to n, so the complexity of our algorithm can be derived by the derivation of following,

$$\sum_{k=0}^{n}(m-(n-k))*(n-k)$$

=(m-n)*n+((m-n-1)*(n-1))+((m-n-2)*(n-2))+-----+mn

=mn+(n*n)+mn-m-(n*n)-n-n+1+-------------------+mn

As the higher order terms are m and n, after expansion we get complexity of O(mn).

IV. COMPARISON WITH EXISTING ALGORITHM

If we consider two sequences X and Y, where X = GGCCTAACTAGAATACCGTACAGTACGAAG and Y = CACTAGGAA.

Alignment using Needleman Wunsch Algorithm:

```
GGCCTAACTAGAATACCGTACAGTACGAAG
    | |   | | | |   |                        |
   CA -CTAGGA -  - -  - - - -  - - - -  - - - - - -A
```

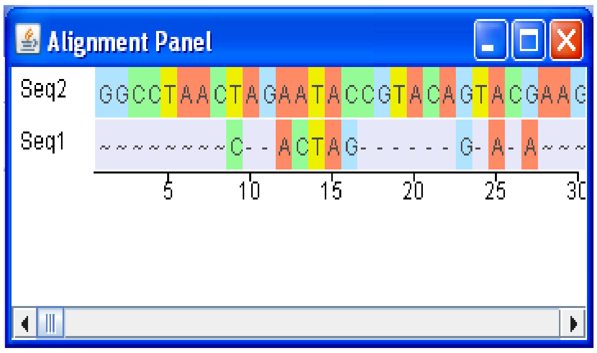
Alignment using Smith Waterman Algorithm:

```
GGCCTAACTAGAATACCGTACAGTACGAAG
                    | |   | | | | |
-  - -  - CA - CTAGGAA  - -  - - -  - -  - - -  - - -  - - -
```

Alignment generated by our algorithm:

```
GGCCTAACTAGAATACCGTACAGTACGAAG
            |   | | | | |             |   |   | |
        - - - - C - ACTAG - - - - - - G - A - A
```

The developed software tool gives the output as the following:



```
Seq2  GGCCTAACTAGAATACCGTACAGTACGAAG
Seq1  ~~~~~~~~C--ACTAG------G-A-A~~~
          5    10   15   20   25   30
```

Figure 3. The output GUI of software tool developed in Java.

V. CONCLUSION

In conclusion, the goal of this paper is to describe a new alignment algorithm and software tool developed in Java for sequences that can be used for determination of deletions and substitutions. Insertions may also be handled if the order of the comparison sequences is switched but it would make sense only if the size of the two sequences is comparable. Our algorithm provides several solutions out of which the best one can be chosen on the basis of minimization of gaps or other considerations. Statistical consideration related to alignment solutions can be studied in a manner similar to those for other alignment algorithms. The algorithm and the application generates good alignment by finding maximum length matches between given sequences. Its complexity is of the same order as the Needleman Wunsch and Smith Waterman algorithms. The basic algorithm can be refined by adding further constraints related to character similarity or gap constraint that will also make it more efficient.

ACKNOWLEDGMENT

Grace to almighty at first for enabling us to complete this work. We would like to express our gratitude to all who have assistance in our work, analysis completion and paper making. We would like to thank all the faculty members of department of Computer Science and Engineering, CUET, without whom nothing these would be possible. We also want to be grateful to all of our friends and well-wishers for being with us this long time. Finally, we would not like to express thanks to someone, rather we would like to say they owe us everything we have, they are our parents and family.

REFERENCES

[1] D. Gusfield, “Algorithms on strings, trees, and sequences”, *computer science and computational Biology*, Cambridge University Press, 1997.

[2] S. Kak, “A frequency analysis of the Indus script”, Cryptologia 12:129-143, 1988.

[3] S. Kak, “The three languages of the brain: quantum, reorganizational, and associative”, In: K. Pribram and J. King (editors), Learning as Self-Organization. Lawrence Erlbaum, Mahwah, 185 –219, 1996.

[4] L.S. Swan and L.J. Goldberg, “Biosymbols: symbols in life and mind”, Biosemiotics 3: 17-31, 2010.

[5] S.B. Needleman and C.D. Wunsch, “A general method applicable to the search for similarities in the amino acid sequence of two proteins”, *Journal of Molecular Biology* 48: 443–53, 1970.

[6] T.F. Smith and M.S. Waterman, “Identification of common molecular subsequences”, *Journal of Molecular Biology* 147: 195-197, 1981.

[7] D. Garg and N. Singla, “String Matching Algorithms and their Applicability in various Applications”, *International Journal of Soft Computing and Engineering*, Vol-1, Issue- 6, January 2012.

[8] Ziad A. A., Alqadil M. Aqel, and El Emary I. M. M., “Multiple Skip Multiple Pattern Matching Algorithm (MSMPMA)”, *IAENG International Journal of Computer Science*, September, 2007.