(Thesis)

Faster and efficient algorithm for sequence alignment

Nusaiba Islam

09201032

nusaiba.islam046@gmail.com

Supporting body:

Abu Mohammad Hammad Ali (Supervisor)

Farzana Rashid (Co-supervisor)

Department of Computer Science & Engineering

BRAC University

DECLARATION

I hereby declare that this thesis is based on the results found by myself. Materials of work found by other researcher are mentioned by reference. This thesis, neither in whole nor in part, has been previously submitted for any degree.

____

Nusaiba Islam

Id: 09201032

Dept: CSE

Supervisor:

____

Abu Mohammad Hammad Ali

Lecturer – III

hammad@bracu.ac.bd

BRAC University, Dhaka

Co – supervisor:

____

Farzana Rashid

Lecturer – I

BRAC University, Dhaka

ACKNOWLEDGMENT

First of all, I would thank our almighty ALLAH for everything.

Next, I would like to thank my supervisor, Abu Mohammad Hammad Ali, and co – supervisor, Farzana Rashid, for their guidance and full support throughout my work.

Contents:

ABSTRACT

In bioinformatics, a sequence alignment is a way of arranging the sequences of DNA, RNA, or protein to identify regions of similarity if two sequences in an alignment share a common ancestor, mismatches can be interpreted as point mutations and gaps as indels. The goal of this paper is to explore the computational approaches to sequence alignment in a faster and optimal way. Two techniques that have been studied are global alignment and local alignment. In this paper, I have used the idea of both the alignment techniques separately. Each technique follows an algorithm (Needleman – Wunsch algorithm for global alignment and Smith – Waterman algorithm for local alignment) which helps in generating proper optimal alignment accordingly. Multiple DNA sequences are read and according to alignment type, the sequences are matched.

Introduction:

Life on Earth originated and then evolved from a universal common ancestor approximately 3.8 billion years ago. Repeated speciation and the divergence of life has occurred throughout this time due to shared sets of biochemical and morphological traits, or by shared DNA sequences. These homologous traits and sequences are more similar among species that share a more recent common ancestor, and can be used to reconstruct evolutionary histories, using both existing species and the fossil record. Existing patterns of biodiversity have been shaped both by speciation and by extinction. These similarities were mostly done by the help of sequence alignment [2].

In bioinformatics, sequence alignment deals with the comparison of two or more DNA, RNA and protein sequences with each other. The comparison is done to identify regions of similarity that may be a consequence of functional, structural, or evolutionary relationships between the sequences. i.e. the similarity shows that the sequences share a common ancestral sequence, as a result similar sequences have similar functionality. Such sequences are said to be homologous [1].

Sequence alignment turns out to be helpful while detecting and identifying known genes or unknown genes, all sorts of mutations (insertion, deletion) i.e. detect the DNA nucleotides responsible for the changes. Comparison against other known protein sequences will help to understand the changed functionality and structural arrangement. Sequence alignment plays an important role in drug design, its help in detecting any abnormal changes in protein sequences to which the drug is subjected. As a result proper drug can be designed with reduced side effects [3].

Literature overview:

Information in [1] [3] helped me to understand the importance of sequence alignment. Information in [1] shows the type of alignment method were used and being used. [7] I learned the basic strategies used for aligning and finding similarity. [9] Helped by highlighting the major factors to concentrate on while aligning. [7][9][12] Explained the workings of both local and global alignment algorithm. [8][10][12][13] Helped me learn the heuristic method FASTA algorithm. By the help of these papers I have learned and built this system.

Types of sequence alignment:

There are several sequence alignment practiced in bioinformatics. The alignments dealing with sequence arrangement of DNA or protein (primary structure) are:

1. **Pairwise sequence alignment**
2. **Multiple sequence alignment**

Pairwise sequence alignment methods are used to find the best-matching piecewise alignments of two query sequences. Pairwise alignments can only be used between two sequences at a time, but they are efficient to calculate and are often used for methods that do not require extreme precision (such as searching a database for sequences with high similarity to a query).

The three primary methods of producing pairwise alignments are **dot-matrix methods, dynamic programming,** and **word methods.**

Multiple sequence alignment is an extension of pairwise alignment to incorporate more than two sequences at a time. Multiple alignment methods try to align all of the sequences in a given query set.

<u>**Methods and algorithms for pairwise sequence alignment**</u>:

Dot-matrix method

Dot-matrix representation is efficient to give an overall picture of discrete and/or repeated regions of similarity between sequences. The sequences are arranged in 2 dimensional matrix and a dot is placed in any region of similarity. This method is appropriate for comparing large sequences over 1000 residues. Also, sequences of unknown similarity can be presented. The dot-matrix approach is qualitative and conceptually simple, though time-consuming to analyze on a large scale. In the absence of noise, it can be easy to visually identify certain sequence features—such as insertions, deletions, repeats, or inverted repeats—from a dot-matrix plot. The dot plots of very closely related sequences will appear as a single line along the matrix's main diagonal.

Problems with dot plots:

Dot matrix doesn’t function well due to problems like noise, lack of clarity, non-intuitiveness, difficulty extracting match summary statistics and match positions on the two sequences. There is also much wasted space where the match data is inherently duplicated across the diagonal and most of the actual area of the plot is taken up by either empty space or noise, and, finally, dot-plots are limited to two sequences.

Dynamic programming

Dynamic programming techniques are used in Needleman-Wunsch algorithm which produces results of global alignment, and in Smith-Waterman algorithm which produces results for local alignment.

- **Global alignments** attempt to align every residue in every sequence, and are most useful when the sequences in the query set are similar and of roughly equal size.
- **Local alignments** are more useful for dissimilar sequences that are suspected to contain regions of similarity or similar sequence segment within their larger sequence context.

Protein alignments use a substitution matrix to assign scores to amino-acid matches or mismatches, and a gap penalty for matching an amino acid in one sequence to a gap in the other. DNA and RNA alignments may use a scoring matrix, but in practice often simply assign a positive match score, a negative mismatch score, and a negative gap penalty.

Dynamic programming is efficient in aligning nucleotides or protein sequences that may also contain frame shift mutation (mainly insertion or deletion). Its ability to evaluate frame shifts offset by an arbitrary number of nucleotides makes the method useful for sequences containing large numbers of indels (insertion or deletion), which can be very difficult to align with more efficient heuristic methods. The dynamic programming method is guaranteed to find an optimal alignment given a particular scoring function; however, identifying a good scoring function is often an empirical rather than a theoretical matter. Although dynamic programming is extensible to more than two sequences, but its functionality is prohibitively slow for large numbers of or extremely long sequences.

Word methods

Word methods, also known as *k*-tuple methods, are heuristic methods that are not guaranteed to find an optimal alignment solution, but are significantly more efficient than dynamic programming. These methods are especially useful in large-scale database searches where it is understood that a large proportion of the candidate sequences will have essentially no significant match with the query sequence.

Sequence alignment (Pairwise):

The basic idea behind aligning two sequences (which could be of different length) is placing one sequence on top of the other and break them by inserting spaces in one or the other sequences such that the identical subsequences are aligned in one-to –one correspondence. The spaces are not inserted in the same positions of the sequence at the same time. Eventually the sequences end up with the same size. For example aligning sequence A = ACAAGACAGCGT

With sequence B = AGAACAAGGCGT.

```
A = A C A A G A C A G - C G T
    |   | |   |   | |   | | |
B = A G A A C A - A G G C G T
```

The sequences are examined for all possible matches, there are total 9 matches, mismatches occurs where the nucleotides don’t seem to match. In this example the mismatches are marked green. The gaps are inserted so that the most matching subsequences are in one-to-one correspondence.

Sequence A can be transformed to sequence B by following three steps:

1. **Substitution**: the mismatch in regarded as substitution.
2. **Insertion**: the gap in sequence A is the insertion of character from sequence B.
3. **Deletion**: the gap in sequence B is the deletion of character from sequence A.

Each of these steps is assigned with a score. After completing the alignment the overall score is calculated. Matches are rewarded positively where as mismatch and gaps are assigned with negative value. The similarity of the sequences can be defined as the best score among all possible the alignment. In this example the score would be 9.(1) + 2.(0) + 2.(-1) = 7.

DNA nucleotide matches are assigned with fixed score but scoring system is different from protein. Each amino acid matches or mismatches are scored differently. This scoring system is known as alphabet - weight scoring system and is implemented by substitution matrix.

<u>Major algorithms</u>:

Needleman-Wunsch algorithm

The standard global alignment algorithm, referred to as Needleman-Wunsch after its original authors. The algorithm computes the similarity between two sequences A and B of lengths m and n, respectively, using a dynamic programming approach. Dynamic programming is a strategy of building a solution gradually using simple recursion. The key observation for the alignment problem is that the similarity between sequences A [1..n] and B[1..m] can be computed by taking the maximum of the three following values:

• The similarity of A[1..n −1] and B[1..m −1] plus the score of substituting A[n] for B[m];

• The similarity of A[1..n −1] and B[1..m] plus the score of deleting aligning A[n]; i.e. adding a gap to A.

• The similarity of A[1..n] and B[1..m −1] plus the score of inserting B[m]. i.e. adding a gap to B.

Smith-Waterman

Local alignment is defined as the problem of finding the best alignment between substrings of both sequences. In 1981, T. F. Smith and M. S. Waterman showed that a local alignment can be computed using essentially the same idea employed by Needleman and Wunsch. The main difference is that M [i, j] contains the similarity between *suffixes* of A[1..i] and B[1..j]. As a result, the recurrence relation is slightly altered because an empty string is a suffix of any sequence and, therefore, a score of zero is always possible. The formula for computing M [i, j] becomes:

M [i, j] = max {0;

M [i −1, j −1] + sub (A[i], B[j]);

M [i −1, j] + del (A[i]);

M [i, j −1] + ins (B[j]) }

Another important distinction is that the score of the best local alignment is the highest value found *anywhere* in the matrix. This position is the starting point for retrieving an optimal alignment using the same procedure described for the global alignment case. The path ends, however, as soon an entry with score zero is reached. It is trivial to see that the Smith-Waterman algorithm has the same time and space complexity as the Needleman-Wunsch.

System details:

This system is based on pairwise sequence alignment. The system will take multiple inputs of sequences at once and align each combination of sequences. The DNA sequences are recorded in a csv file. This file is read as inputs and the outputs are generated accordingly.

The system depends on few major factors listed below:

1. **Type of sequence**: input sequences are DNA sequences only.
2. **Alignment** : the sequences will be matched in one –to-one manner such that each character in a pair is associated with a single character or a gap
3. **Alignment score:** it's a numerical value that will describe the overall quality of an alignment. The alignment with highest score is the alignment with highest similarity.
4. **Match value:** the homologous character in a pair is rewarded with match value. In case of DNA sequence match value is set as 1.
5. **Mismatch value:** the pair of different characters in a homologous position is stated as a mismatch and a mismatch value is added instead. In case of DNA sequence mismatch value is set as 0.
6. **Gap:** absence of a homologous character in either sequence could be a reason of deletion in the comparing sequence or an insertion in the sequence to which it is compared to. In either case a gap is added and a value of -1 is added to the score.

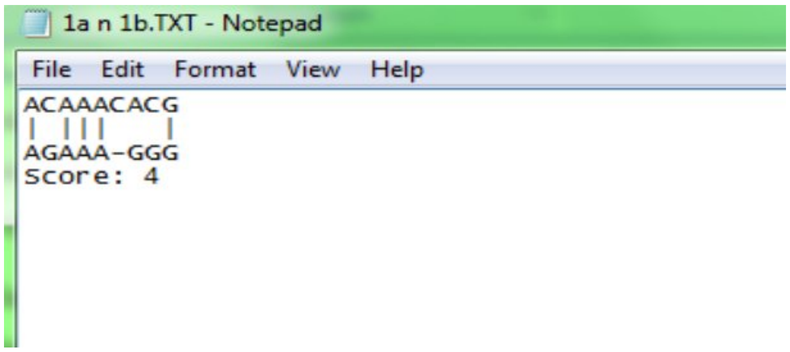
The alignment of the sequences are done by using two aligning method.

Global alignment

This method aligns the pair of sequences from end to end. The entire length of the sequence is taken into account. An optimal score is calculated from the matrix formed using the maximum similarity of each character using match, mismatch and gap penalty values of the sequences. The optimal alignment is achieved by trace back of the matrix.

Input sequences: ACAAACACG and AGAAAGGG (two sequence of DNA taken from e.coli bacteria)

Global alignment output :



```
ACAAACACG
|  |||    |
AGAAA-GGG
Score: 4
```

Fig.1

Local alignment

This method aligns only a subsequence of the given pair of sequences. The region with highest match density is given priority than the rest of the sequences. The alignment can start and end in any region of the sequence and only the best matching portion is aligned. A matrix is filled using the maximum similarity score for each pair of character. The resultant subsequences are generated by tracing back of the matrix starting from the maximum score till it reaches zero.

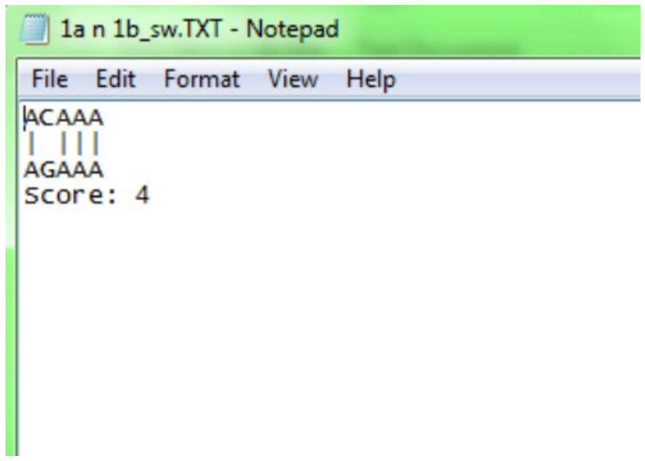
The same sequences tested for global alignment is used for local alignment.

Input: seq1 = ACAAACACG

Seq2 = AGAAAGGG

Highest match density

Output:



```
ACAAA
| |||
AGAAA
Score: 4
```

Fig .2

Working with global alignment:

The global alignment is achieved by using dynamic programming algorithm designed by Needleman – Wunsch.

The following recurrence is derived from the Needleman-Wunsch algorithm description:

$$\mathrm{sim}(A[1..i], B[1..j]) = \max \begin{cases} \mathrm{sim}(A[1..i-1], B[1..j-1]) + \mathrm{sub}(A[i], B[j]); \\ \mathrm{sim}(A[1..i-1], B[1..j]) + \mathrm{del}(A[i]); \\ \mathrm{sim}(A[1..i], B[1..j-1]) + \mathrm{ins}(B[j]) \end{cases}$$

- sim(A,B) : gives the similarity of sequence A and B.
- sub (A,B): gives the match or mismatch score.
- del (A,B): gives the score of adding a gap for a deleted character.
- ins (A,B): gives the score of a gap when a character is inserted.

This recurrence generates sets of values. To store this value and work with the recursion, a matrix is created of size (m+1) * (n+1).

Both row and column length is incremented by 1 because of an extra gap (-) character. Initially the matrix is filled with zeroes.

E.g.

	-	A	T	C	G
-	0	0	0	0	0
A	0	0	0	0	0
T	0	0	0	0	0
G	0	0	0	0	0
C	0	0	0	0	0

Then a scoring value is set for characters by comparing them with the gap index.

Gap = -1

The first row is filled with j*gap and the first column is filled with i*gap value.

Eg.

	-	A	T	C	G
-	0	-1	-2	-3	-4
A	-1	0	0	0	0
T	-2	0	0	0	0
C	-3	0	0	0	0
G	-4	0	0	0	0

The gap score is added to the first row and column to show that there are possibilities for the all the characters to be replaced by gaps. The gaps may appear consecutively in either sequence and the scores represent the number of added gaps.

The system reads multiple inputs and takes two sequences for comparison at a time. Considering the following sequence of DNA taken from E.coli bacteria as inputs:

Sequence 1 = ACAAACACG   length=9

Sequence 2 =AGAAAGGG   length = 8

The system creates a matrix of size (9 + 1) * (8 + 1). As shown above the matrix is first filled with zeroes and then the first row and column is filled with gap scores.

The rest of the cells are filled by the following recurrence:

M [i.j] = max {M [i −1, j −1] + sub (seq1[i], seq2[j]),

M [i −1, j] + gap, ***deletion from sequence 1*

M [i, j −1] + gap} ***insertion in sequence 2*

Scoring matrix filled using the above recurrence.

	-	A 1	G 2	A 3	A 4	A 5	G 6	G 7	G 8
-	0	-1	-2	-3	-4	-5	-6	-7	-8
A	-1	1	0	-1	-2	-3	-4	-5	-6
C	-2	0	1	0	-1	-2	-3	-4	-5
A	-3	-1	0	2	1	0	-1	-2	-3
A	-4	-2	-1	1	3	2	1	0	-1
A	-5	-3	-2	0	2	4	3	2	1
C	-6	-4	-3	-1	1	3	4	3	2
A	-7	-5	-4	-2	0	2	3	4	3
C	-8	-6	-5	-3	-1	1	2	3	4
G	-9	-7	-5	-4	-2	0	2	3	4

The diagonal area of the matrix carries the optimal alignment result. Once the matrix is formed, the trace back starts from the right most end corner, which is also the maximum optimal alignment score.

The backward trace direction depends on:

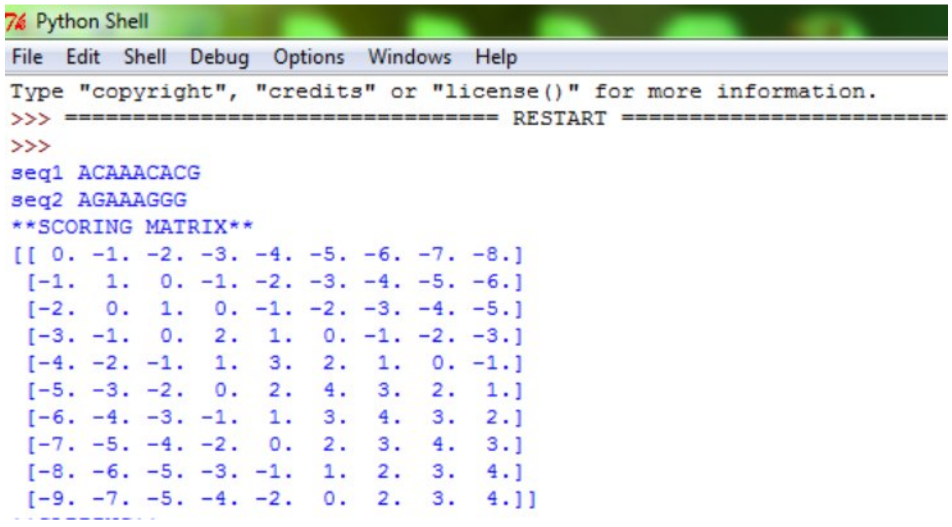
- Score of the current cell i.e. M [i , j] where i and j are the lengths of the sequences.

- Score of diagonal cell i.e. M [i-1 , j-1]. (match or mismatch)
- Score of up cell i.e. M[i-1, j] (deletion in sequence 1)
- Score of left cell i.e. M[i , j-1] (insertion in sequence 2)

As shown previously, each cell is filled by the maximum of the three conditions i.e. substitution, deletion, insertion. Therefore, the current score is the maximum of the three conditions. As a result:

1. If score is equal to diagonal cell plus substitution score of the current characters, then the arrow moves diagonally.
2. If the score is equal to up cell plus a gap, the arrow moves up and a gap is placed in sequence 2
3. If the score is equal to left cell plus a gap, the arrow moves to left and a gap is added to sequence 1.

System output:

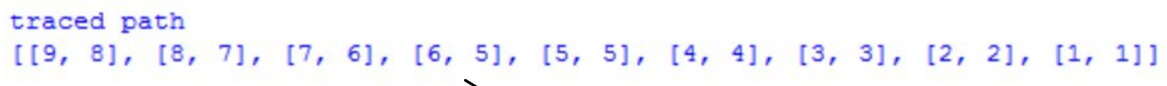


```
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ========================
>>>
seq1 ACAAACACG
seq2 AGAAAGGG
**SCORING MATRIX**
[[ 0. -1. -2. -3. -4. -5. -6. -7. -8.]
 [-1.  1.  0. -1. -2. -3. -4. -5. -6.]
 [-2.  0.  1.  0. -1. -2. -3. -4. -5.]
 [-3. -1.  0.  2.  1.  0. -1. -2. -3.]
 [-4. -2. -1.  1.  3.  2.  1.  0. -1.]
 [-5. -3. -2.  0.  2.  4.  3.  2.  1.]
 [-6. -4. -3. -1.  1.  3.  4.  3.  2.]
 [-7. -5. -4. -2.  0.  2.  3.  4.  3.]
 [-8. -6. -5. -3. -1.  1.  2.  3.  4.]
 [-9. -7. -5. -4. -2.  0.  2.  3.  4.]]
```

Fig.3

The sequences are read and a matrix is created. The scores are found using the Needleman – Wunsch recurrence.

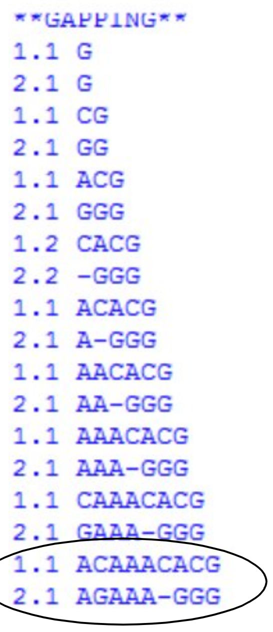
The matrix is traced starting from the maximum score.



```
traced path
[[9, 8], [8, 7], [7, 6], [6, 5], [5, 5], [4, 4], [3, 3], [2, 2], [1, 1]]
```

Fig.4

The traced path shows the indices of the matrix that is traced. Along with the traced indices, gapping of both the sequences is generated at every traced cell.

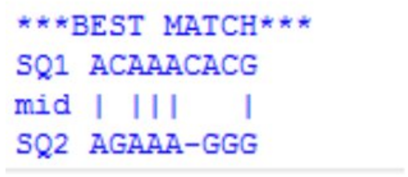


```
**GAPPING**
1.1 G
2.1 G
1.1 CG
2.1 GG
1.1 ACG
2.1 GGG
1.2 CACG
2.2 -GGG
1.1 ACACG
2.1 A-GGG
1.1 AACACG
2.1 AA-GGG
1.1 AAACACG
2.1 AAA-GGG
1.1 CAAACACG
2.1 GAAA-GGG
1.1 ACAAACACG
2.1 AGAAA-GGG
```

Aligned sequences are generated with appropriate gaps.

Fig.5

Finally the sequences are represented as shown below:



```
***BEST MATCH***
SQ1 ACAAACACG
mid | |||   |
SQ2 AGAAA-GGG
```

Fig .6

Working with Local Alignment:

Local alignment corresponds to find the best global alignment within subsequences of the input sequences. Only the alignments with score zero and higher are considered. Local alignment is achieved by using Smith – Waterman algorithm. Similar to the global alignment, the matrix is filled by maximum score of substitution, insertion, deletion and zero.

Smith – Waterman algorithm:

```
M [ i , j] = max { 0,
                 M [ i −1, j −1]  + sub ( seq1[i], seq2[j] ),
                 M [i −1, j] + gap, **deletion from sequence 1
                 M [i, j −1] + gap} **insertion in sequence 2
```

The system takes in the sequences and creates a matrix just like global alignment. Similarly, the matrix is first filled with zero. Since local alignment deals with scores that is zero and higher, unlike global alignment the first Row and column is not filled with gap value.

Input sequences:

Sequence 1 = ACAAACACG

Sequence 2 = AGAAAGGG

Once the matrix is filled, it is searched for the maximum score. The corresponding indices of the maximum score are used for the trace back. A working matrix along with the scores for the given sequences is shown below.

Matrix representation:

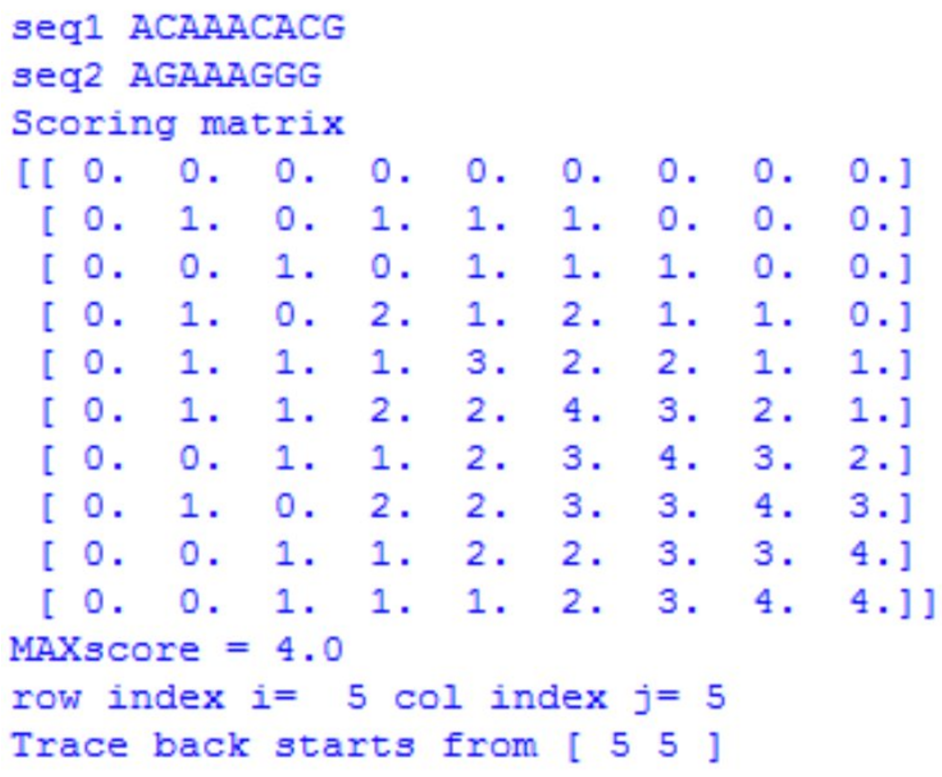
	-	A	G	A	A	A	G	G	G
-	0	0	0	0	0	0	0	0	0
A	0	1	0	1	1	1	0	0	0
C	0	0	1	0	1	1	1	0	0
A	0	1	0	2	1	2	1	1	0
A	0	1	1	1	3	2	2	1	1
A	0	1	1	2	2	4	3	2	1
C	0	0	1	1	2	3	4	3	2
A	0	1	0	2	2	3	3	4	3
C	0	0	1	1	2	2	3	3	4
G	0	0	1	1	1	2	3	4	4

The matrix is searched till it reaches to last cell that carries the highest score. In this case the max score is 4 and its indices are [5 , 5]. The trace back start from the given indices and the best tracked path is calculated using the Smith – Waterman recurrence, similar to Needleman – Wunsch.

Traced path is exactly diagonal, showing that the subsequences are matched and no gaps are added. The system output will give a better explanation.

System output:

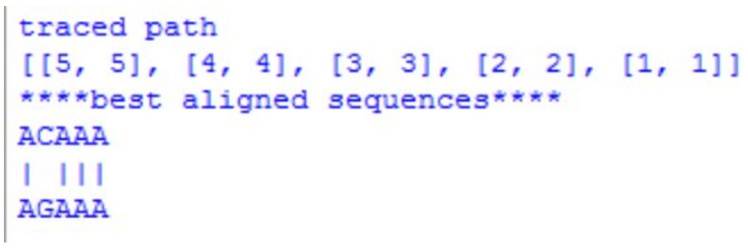
The system reads the sequences and generates a matrix using the Smith – Waterman algorithm.



```
seq1 ACAAACACG
seq2 AGAAAGGG
Scoring matrix
[[ 0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  1.  0.  1.  1.  1.  0.  0.  0.]
 [ 0.  0.  1.  0.  1.  1.  1.  0.  0.]
 [ 0.  1.  0.  2.  1.  2.  1.  1.  0.]
 [ 0.  1.  1.  1.  3.  2.  2.  1.  1.]
 [ 0.  1.  1.  2.  2.  4.  3.  2.  1.]
 [ 0.  0.  1.  1.  2.  3.  4.  3.  2.]
 [ 0.  1.  0.  2.  2.  3.  3.  4.  3.]
 [ 0.  0.  1.  1.  2.  2.  3.  3.  4.]
 [ 0.  0.  1.  1.  1.  2.  3.  4.  4.]]
MAXscore = 4.0
row index i=  5 col index j= 5
Trace back starts from [ 5 5 ]
```

Fig .7

The trace back start from the given indices and the subsequences of highest match density is generated.



```
traced path
[[5, 5], [4, 4], [3, 3], [2, 2], [1, 1]]
****best aligned sequences****
ACAAA
| |||
AGAAA
```

Fig .8

Results:

The systems are tested by using multiple inputs of DNA sequences of E.coli bacteria. The original alignment data of the sequences were collected using NEOBIO. This original data helped me to compare the system outputs and check the accuracy.

The input sequences are as follows:

ACAAACACG
AGAAAGGG
AGAAAGGC
AGAAAGGGA
AAAAATAATAAAA
ATAAACTTAATAATAAAA

Global alignment		
Sequences	System output	Original data
seq1 ACAAACACG seq2 AGAAAGGG	ACAAACACG | ||| | AGAAA-GGG score 4.0	ACAAACACG | ||| | AGAAA-GGG Score: 4
seq1 ACAAACACG seq2 AGAAAGGC	ACAAACACG | ||| | AGAAAGGC- score 4.0	ACAAACACG | ||| | AGAAAGGC- Score: 4
seq1 ACAAACACG seq2 AGAAAGGGA	ACAAACACG | ||| AGAAAGGGA score 4.0	ACAAACACG | ||| AGAAAGGGA Score: 4
seq1 ACAAACACG seq2 AAAAATAATAAAA	--ACA-AA-CACG | | || | AAAAATAATAAAA score 1.0	ACAAACACG---- | ||| | AAAAATAATAAAA Score: 1
seq1 ACAAACACG seq2 ATAAACTTAATAATAAAA	----AC---A-AA-CACG || | || | ATAAACTTAATAATAAAA score -3.0	ACAAAC--ACG------- | |||| | ATAAACTTAATAATAAAA Score: -3

```
seq1 AGAAAGGG             AGAAAGGG                AGAAAGGG
seq2 AGAAAGGC             |||||||                 |||||||
                          AGAAAGGC                AGAAAGGC
                          score 7.0               Score: 7
```

```
seq1 AGAAAGGG             AGAAAGGG-               AGAAAGGG-
seq2 AGAAAGGGA            ||||||||                ||||||||
                          AGAAAGGGA               AGAAAGGGA
                          score 7.0               Score: 7
```

```
seq1 AGAAAGGG             ----AGAA-AGGG           AGAAAGGG-----
seq2 AAAAATAATAAAA            | || |              | |||
                          AAAAATAATAAAA           AAAAATAATAAAA
                          score -1.0              Score: -1
```

```
seq1 AGAAAGGG             ---------AGAA-AGGG      AGAAAGGG----------
seq2 ATAAACTTAATAATAAAA            | || |         | |||
                          ATAAACTTAATAATAAAA      ATAAACTTAATAATAAAA
                          score -6.0              Score: -6
```

```
seq1 AGAAAGGC             AGAAA-GGC               AGAAAGGC-
seq2 AGAAAGGGA            ||||| ||                |||||||
                          AGAAAGGGA               AGAAAGGGA
                          score 6.0               Score: 6
```

```
seq1 AGAAAGGC             ----AGAA-AGGC           AGAAAGGC-----
seq2 AAAAATAATAAAA            | || |              | |||
                          AAAAATAATAAAA           AAAAATAATAAAA
                          score -1.0              Score: -1
```

```
seq1 AGAAAGGC             ---------AGAA-AGGC      AGAAAGGC----------
seq2 ATAAACTTAATAATAAAA            | || |         | |||
                          ATAAACTTAATAATAAAA      ATAAACTTAATAATAAAA
                          score -6.0              Score: -6
```

```
seq1 AGAAAGGGA            --AGA-AA-GGGA           AGAAAGGG-A---
seq2 AAAAATAATAAAA          | | ||     |          | |||    |
                          AAAAATAATAAAA           AAAAATAATAAAA
                          score 1.0               Score: 1
```

```
seq1 AGAAAGGGA            ----A---GA-AA-GGGA      AGAAAGGGA---------
seq2 ATAAACTTAATAATAAAA       |    | ||    |      | |||   |
                          ATAAACTTAATAATAAAA      ATAAACTTAATAATAAAA
                          score -4.0              Score: -4
```

```
seq1 AAAAATAATAAAA        --AAA---AATAATAAAA      A-AAA---A-TAATAAAA
seq2 ATAAACTTAATAATAAAA     |||   ||||||||||      | |||   | ||||||||
                          ATAAACTTAATAATAAAA      ATAAACTTAATAATAAAA
                          score 8.0               Score: 8
```

Score: All the scores of aligned sequences done by the system are same as the original aligned scores.

Match and mismatch: number of match and mismatches in tested outputs are same as those of original data.

Number of gaps: Numbers of gaps added are also same to that of original data.

****Position of gaps:** 10 out of 15 outputs acquired by the system show different gap position, although the score and number of gaps are same as that of original data.

The global alignment system outputs show different position of gaps in most of the sequences. This could be due to trace back values. Since the gap addition depends on tracing path and also on the score values, the score calculated may have differed from that of the original data.

Local alignment		
Sequences	System output	Original data
seq1 ACAAACACG seq2 AGAAAGGG	MAXscore = 4.0 ACAAA | ||| AGAAA	ACAAA | ||| AGAAA Score: 4
seq1 ACAAACACG seq2 AGAAAGGC	MAXscore = 5.0 ACAAACAC | ||| | AGAAAGGC	ACAAACAC | ||| | AGAAAGGC Score: 5
seq1 ACAAACACG seq2 AGAAAGGGA	MAXscore = 4.0 ACAAA | ||| AGAAA	ACAAA | ||| AGAAA Score: 4
seq1 ACAAACACG seq2 AAAAATAATAAAA	MAXscore = 5.0 ACAAACA | ||| | AAAAATA	ACAAACA | ||| | AAAAATA Score: 5
seq1 ACAAACACG seq2 ATAAACTTAATAATAAAA	MAXscore = 5.0 ACAAAC | |||| ATAAAC	ACAAAC | |||| ATAAAC Score: 5
seq1 AGAAAGGG seq2 AGAAAGGC	MAXscore = 7.0 AGAAAGG ||||||| AGAAAGG	AGAAAGGG ||||||| AGAAAGGC Score: 7
seq1 AGAAAGGG seq2 AGAAAGGGA	MAXscore = 8.0 AGAAAGGG |||||||| AGAAAGGG	AGAAAGGG |||||||| AGAAAGGG Score: 8

```
seq1 AGAAAGGG                 MAXscore = 4.0        AGAAA
seq2 AAAAATAATAAAA            AGAAA                 | |||
                              | |||                 AAAAA
                              AAAAA                 Score: 4

seq1 AGAAAGGG                 MAXscore = 4.0        AGAAA
seq2 ATAAACTTAATAATAAAA       AGAAA                 | |||
                              | |||                 ATAAA
                              ATAAA                 Score: 4

seq1 AGAAAGGC                 MAXscore = 7.0        AGAAAGG
seq2 AGAAAGGGA                AGAAAGG               |||||||
                              |||||||               AGAAAGG
                              AGAAAGG               Score: 7

seq1 AGAAAGGC                 MAXscore = 4.0        AGAAA
seq2 AAAAATAATAAAA            AGAAA                 | |||
                              | |||                 AAAAA
                              AAAAA                 Score: 4

seq1 AGAAAGGC                 MAXscore = 4.0        AGAAA
seq2 ATAAACTTAATAATAAAA       AGAAA                 | |||
                              | |||                 ATAAA
                              ATAAA                 Score: 4

seq1 AGAAAGGGA                MAXscore = 4.0        AGAAA
seq2 AAAAATAATAAAA            AGAAA                 | |||
                              | |||                 AAAAA
                              AAAAA                 Score: 4

seq1 AGAAAGGGA                MAXscore = 5.0        AGAAAGGGA
seq2 ATAAACTTAATAATAAAA       AGAAAGGGA             | |||   |
                              | |||   |             ATAAACTTA
                              ATAAACTTA             Score: 5

seq1 AAAAATAATAAAA            MAXscore = 10.0       AAAAATAATAAAA
seq2 ATAAACTTAATAATAAAA       AAAAATAATAAAA         |||  |||||| |
                              |||  |||||| |         AAACTTAATAATA
                              AAACTTAATAATA         Score: 10
```

Scores: the generated scores of the system match all the scores of original data.

Subsequence: the length and characters of the subsequences generated by the system is same as the original data.

Gap: similar to the original value. No gaps were added to the subsequences.

Match and mismatch: number mismatch and matches are same as the original value.

The above data explains that this system gives the same output as given in original data. We can say that the local alignment system is working correctly and is able to generate outputs with 100% accuracy.

Output feedback:

The original data was generated by NEOBIO software which is designed by using JAVA, but I have used PYTHON to work with these algorithms. Although in performance, JAVA works faster than PYTHON, but it is much easier to develop in PYTHON. My system showed no delay to generate the same output as that through NEOBIO. It is successful in reading multiple lengths of DNA sequences and generating optimal alignment efficiently.

Difficulties:

I have used the basic match values instead of substitution matrices mentioned in several papers. I tried using BLOSUM62 substitution matrix which contains the different match values for different nucleotide alignment as well as protein alignment. My system was unable to read this matrix; if it would have worked then the system could have taken protein sequences as inputs. I wanted to write the outputs to a text file also, but every time the system was run it generated blank text output, although the system was running correctly.

There was always an issue about choosing an efficient medium or optimal medium. As both efficiency and optimality doesn't work side by side, there is always a choice to be made between these two factors when it comes to large sequence query. After Needleman – Wunsch and Smith – Waterman algorithms, FASTA and BLAST algorithms were introduced. These algorithms are intended to give a faster result but it may lack in optimality.

FASTA algorithm:

Several studies and researches show that a large amount of sequences would require more time to generate optimal output. This issue is handled by FASTA algorithm.

In FASTA algorithm, a fixed k length of substrings of highest similarity is chosen. Each query substrings are then matched with the substrings with highest similarity. These substrings of highest similarity would give a diagonal match when represented in a matrix. FASTA creates a hash table of k tuples and stores the location of all k – tuples into the table. This helps in retrieving the location of the matching substrings and marks its position in matrix. The algorithm then identifies ten highest diagonal by the help of the marked position and calculate the diagonal score by adding a positive value for each marked match. The rest of the sequences are discarded. These highest scoring segments are scored again using a substitution matrix or basic match techniques. The segments with score less than threshold are discarded again.

Finally, these segments are placed in the matrix and local alignment algorithm is run. The characters missing between the segments are replaced by gaps and gap penalty is added. This results in a final optimally aligned sequence.

My future work:

I have worked with the techniques of Needleman – Wunsch and Smith – Waterman algorithm which gives an optimal output for specific length of DNA sequences. In future I will try to use heuristic method (FASTA) along with this algorithm which would give optimal output for large amount of sequences efficiently.

Conclusion:

In this thesis, the methods of aligning DNA sequences optimally and relatively efficiently are studied. It shows that, by the help of dynamic programming and its techniques several DNA sequences with defects or vague identity could be recognized by aligning with existing genetic data. In bioinformatics, sequence alignment of such type is greatly required for generating correct outputs. As I have mentioned, aligning plays an important role in drug design, forensics, DNA defects etc. The demand for faster and optimizing algorithm would also be at high peak for bioinformatics due to increasing need of better drugs and treatment. This proposed system is built using the algorithms proposed in early 80's and many software are using these algorithm for alignment. It takes in several sequences of DNA and runs an alignment test, gives the output along with the matrices and traced path. For both global and local alignment the output score matches and number of gaps are correct as the original data, only few sequences show different gap positions. In totality, this system successfully generated properly aligned sequences efficiently.

References:

1. http://en.wikipedia.org/wiki/Sequence_alignment#Structural_alignment
2. http://en.wikipedia.org/wiki/Evolution
3. http://en.wikipedia.org/wiki/Bioinformatics
4. http://ksvi.mff.cuni.cz/~mraz/bioinf/BioAlg10-8.pdf
5. http://www.cs.helsinki.fi/bioinformatiikka/mbi/courses/07-08/itb/slides/itb0708_slides_83-116.pdf
6. http://intl-bib.oxfordjournals.org/content/11/5/473.full.pdf
7. http://iic.arizona.edu/static/resources/2010/07/02/SequenceAlignment.pdf
8. http://bioinformatics.oxfordjournals.org/content/14/1/25.abstract
9. http://www.people.usi.ch/baludam/projects/ALiBio/Thesis_Baluda_Mauro.pdf
10. http://www.cmb.usc.edu/papers/msw_papers/msw-053.pdf
11. http://neobio.sourceforge.net/
12. http://biochem218.stanford.edu/Projects%202004/Chan.pdf
13. http://www.cs.tau.ac.il/~rshamir/algmb/98/scribe/pdf/lec03.pdf
14. http://www.itu.dk/people/sestoft/bsa.html
15. http://blogs.infoecho.net/echo/2011/04/10/how-to-implement-the-needleman%E2%80%93wunsch-alignment-algorithm-without-using-a-single-loop-in-python/
16. http://www.codesofmylife.com/2011/04/10/needleman-wunsch-algorithm-for-global-sequence-alignment-in-python/
17. http://code.google.com/p/gal2009/source/browse/trunk/app/GalServer/src/main/java/gal/needleman/wunsch/NeedlemanWunsch.java?r=22
18. http://www.seas.gwu.edu/~rhyspj/fall09cs144/lab3/lab34.html
19. http://www.ibm.com/developerworks/java/library/j-seqalign/index.html

tools

20.http://bioinformatics.igc.gulbenkian.pt/resources/tools/sequenceanalysis/
21.http://hari.hyderabadatoz.com/Bioinf_tools_soft.html
22.http://dna.engr.uconn.edu/?page_id=110
23.http://www.bioinformatics.babraham.ac.uk/projects/seqmonk/
24.http://www.mybiosoftware.com/alignment/2732

python

25.http://www.ibm.com/developerworks/java/library/j-seqalign/index.html
26.http://docs.python.org/2/tutorial/appetite.html
27.http://www.google.com/search?client=opera&q=waterman+smith+algorithm+for+local+alignment+in+python&sourceid=opera&ie=utf-8&oe=utf-8&channel=suggest
28.http://docs.python.org/2/library/index.html#library-index
29.http://stackoverflow.com/questions/12666494/how-do-i-decide-which-way-to-backtrack-in-the-smithwaterman-algorithm
30.http://www.codesofmylife.com/2011/05/13/smith-waterman-algorithm-for-local-alignment-in-python/
31.http://en.wikipedia.org/wiki/Hidden_Markov_model
32.http://en.wikipedia.org/wiki/Gap_penalty
33.http://home.wlu.edu/~lambertk/pythontojava/
34.http://docs.python.org/2/reference/index.html#reference-index
35.https://github.com/kevinakwok/bioinfo/blob/master/Smith-Waterman/smith-waterman.py