SEQUENCE ALIGNMENT FOR GENOME ASSEMBLY

INTELLIGENCE OF BIOLOGICAL SYSTEMS - 1 (19BIO112)

A PROJECT REPORT

Submitted by

Venkata Ramana Murthy. Polisetty	-	CH.EN.U4AIE20077
Chereddy. Spandana	-	CH.EN.U4AIE20010
Priyadharshini. AR	-	CH.EN.U4AIE20051
Bachu Ganesh	-	CH.EN.U4AIE20003
Krithika. S	-	CH.EN.U4AIE20034

Under the Guidance of

Dr.Oviya

In partial fulfillment for the award of the degree

Of

BACHELORS OF TECHNOLOGY

In

COMPUTER SCIENCE ENGINEERING

Amrita Vishwa Vidyapeetham

Chennai – 601 103, Tamil Nadu, India.

July 2021

Acknowledgment

I offer my sincere pranams at the lotus feet of Universal guru, **MATA AMRITANANDAMAYI DEVI** who blessed me with her grace to make this a successful project.

I express my deep sense of gratitude to **Dr. P Shankar,** Principal, Amrita School of Engineering, Chennai for his kind support. I would like to extend my gratitude and heartfelt thanks to **Dr. A Manikandan,** Chairperson, Department of Electrical and Electronics Engineering, for his constant help, suggestions and inspiring guidance. I am grateful to my guide **Dr.Oviya** , Department of Biology , for his invaluable support and guidance during the course of the major project work. I am also indebted to **Dr. Prasanna Kumar**, Chairperson of Department of AI, ASE, Chennai for his guidance.

I specially thank our director **Sri I B Manikantan** who supported us in every possible manner.

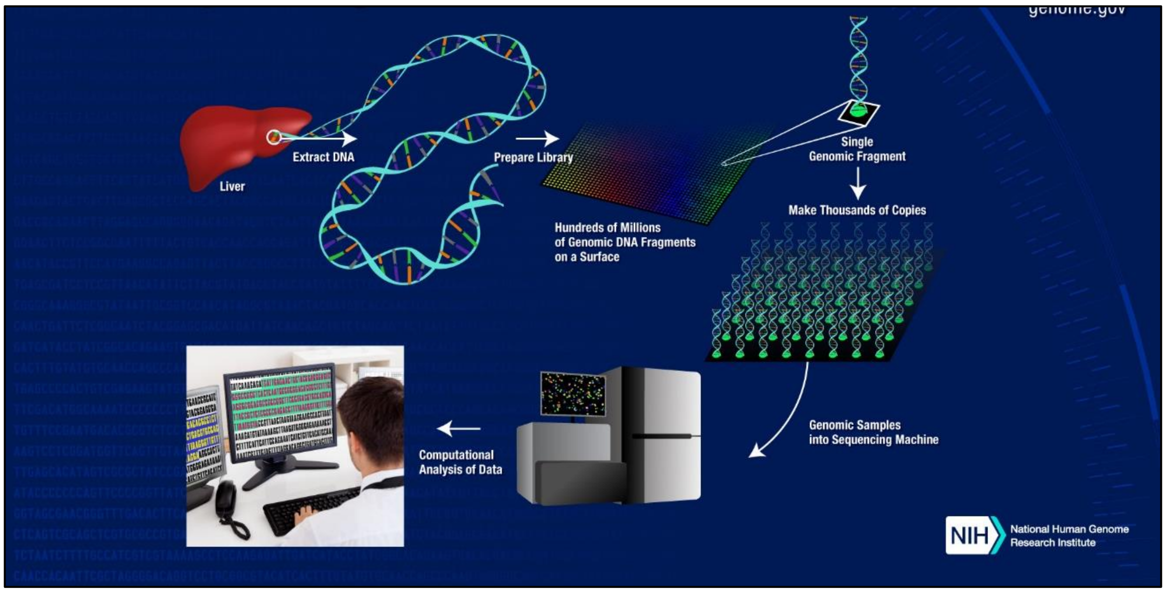
ABSTRACT

One of the most complex and computationally intensive tasks of genome sequence analysis is genome assembly. Even today, few centers have the resources, in both software and hardware, to assemble a genome from the thousands or millions of individual sequences generated in a whole-genome shotgun sequencing project. With the rapid growth in the number of sequenced genomes has come an increase in the number of organisms for which two or more closely related species have been sequenced. This has created the possibility of building a comparative genome assembly algorithm, which can assemble a newly sequencedgenome by mapping it onto a reference genome.

INTRODUCTION

DNA sequencing technologies have revolutionized biology.

The genomes of more than 800 bacteria and 100 eukaryotes have been sequenced, including the genomes of several human individuals. Wealth of data has resulted in numerous biological discoveries and led to a betterunderstanding of the fundamental principles of life. The dramatic impact of sequencing as a key component of modern biological research is, at first glance, surprising due to limitations in the length of DNA fragments that can be sequenced with current technologies. Modern sequencing instruments can only ‘read’ DNA fragments of up to ∼2000 bp (commonly just 600–1000 bp), orders of magnitude shorter than the genomes of most living organisms. Throughout the years, in fact, many thoughts that such limitations would prevent the sequencing of large genomes.


genome.gov
Liver
Extract DNA
Prepare Library
Hundreds of Millions of Genomic DNA Fragments on a Surface
Single Genomic Fragment
Make Thousands of Copies
Genomic Samples into Sequencing Machine
Computational Analysis of Data
NIH National Human Genome Research Institute

Today, however, the sequencing of bacteria (millions of base pairs in length) is done routinely and the sequencing of 1000 human genomes (3 billion bp in length, each) is considered possible within the next 3 years. Process involves shearing the genome of an organism into multiple small fragments, each of which being then sequenced separately.

The resulting DNA segments are combined into a reconstruction of the original genome using computer programs called genome assemblers. Similarly, the difficulty of the assembly problem is dependent on the number of reads being assembled—large genomes and/or short DNA fragments posing specific challenges. To overcome such challenges, sophisticated computational algorithms have been developed over the years, resulting in genome assemblers capable of reconstructing large mammalian genomes. these programs have been critical to the success of many recent genome projects including mammals, plants and worms. Despite such successes, genome assembly is far from being a solved problem. New challenges are posed by recent advances in genome sequencing technologies, both due to the sheer size of the data that need to be processed (a single sequencing instrument can provide throughput similar to what was previously only achievable at a genome center), and due to the characteristics of the new data: shorter sequencing reads and new types of sequencing errors. Furthermore, assembly software must adapt to new applications of DNA sequencing in biological research. Perhaps the biggest new challenge is posed by the large-scale sequencing of entire microbial communities.

GENOME ASSEMBLY

Aiming at reconstructing genomes that are similar to any organismspreviously sequenced, and comparative (re-sequencing) approaches that use the sequence of a closely related organism as a guide during the assembly process. comparative assembly is a much easier task—it is essentially sufficient to align the set of reads to the reference genomein order to characterize a newly sequenced organism. The characteristics of the sequencing technology being used also restrict thechoice of assembly strategy.

GENOME ASSEMBLY IN COMPUTATIONAL BIOLOGY

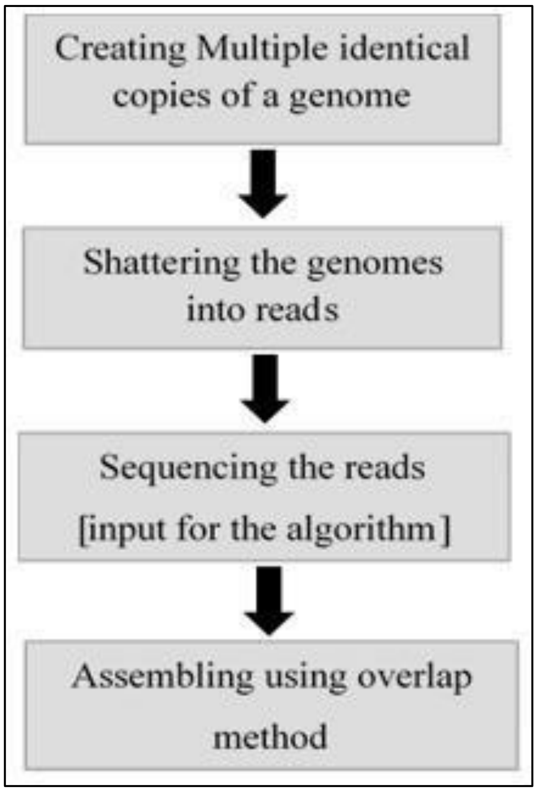
Genome assembly is the process of reconstructing fragments of the DNA sequence to obtain the original sequence. This is required as it iscomputationally difficult to read a complete sequence all at once.
Genome is divided into smaller fragments called reads. These reads consist of 20 to 30,000 bases. Genome assembly can be interpreted as solving a jigsaw puzzle. In a jigsaw puzzle, we connect numerous numbers of small pieces in order to get the complete figure. In spite ofhaving the complete figure, which is the prior information that discloses how every single individual piece associate with each other. Here we have a vast number of individual pieces, called reads, and wedo not know how to connect them altogether since we do not know thefinal product, the sequence.

In order to obtain the pieces or the reads, we take an analogy from a newspaper problem. In this problem, multiple copies of the same newspaper are exploded. And there are chances of many missing pieces. This is the exact reason why we don't use a single newspaper. Before modelling the problem there are some set of assumptions, we consider in order to reduce complications. To begin with, DNA is a double-strand, it is difficult to determine whether to use a read or its reverse complement when assembling a particular strand of a genome.Therefore, we assume that all reads are from a single strand. Secondly,we assume that all reads are of the same size.

An overlap-based assembler is one method for reconstruction. This algorithm finds overlaps between the reads in order to build a string graph which will be the input for the assembly. Reads of k length aretermed as k-mer.

The problem of finding overlaps between each ordered pair of k-mer is commonly called the all-pairs suffix-prefix problem, that is if the prefix of one k-mer matches

with the suffix of another k-mer then the two
k-mers are joined. The flow graph is depicted below-


Creating Multiple identical copies of a genome
Shattering the genomes into reads
Sequencing the reads [input for the algorithm]
Assembling using overlap method

Any computer program for overlap sequencing, even though it may work well for a limited number of inputs, will not work sufficiently for all inputs. The challenges increase when the length of the string increases and when there is repetition in the k-mer. When there are duplicates, the choices of the next k-merbecome more than one which might lead to the wrong sequence and the sequence may not be completed.

Repeated substrings in the genome are not a serious problem when we have a lesser number of reads, but with millions of reads, repeats make it much more difficult to look ahead and construct the correct assembly.Also finding the starting k-mer becomes difficult due to the presence of duplicates. Overlap assembly methods have time complexity O(N^2), where N is number of k-mers.

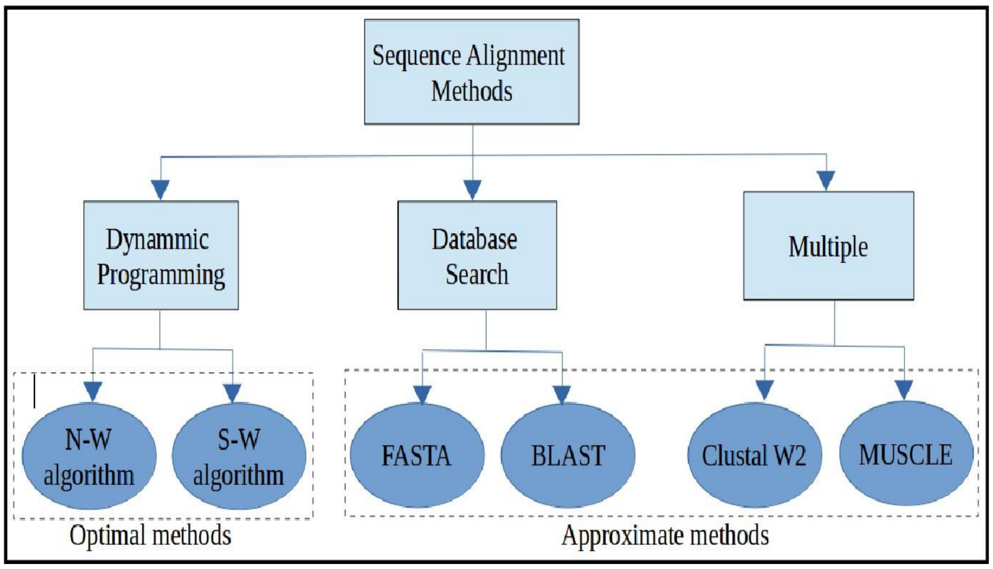
To overcome this problem a huge network is created by connecting each k-mer using overlap criteria. Thereare many ways to align the reads.

SEQUENCE ALIGNMENT

Sequence alignment is a common and often repeated task in molecular biology. Typical alignment operations consist of finding similarities between a pair of sequences (pairwise sequence alignment) or a family of sequences (multiple sequence alignment). The need for speeding up this treatment comes from therapid growth rate of biological sequence databases: every year their size increases by a factor of 1.5 to 2.

Sequence comparison is a basic operation in DNA sequencing projects, and most of sequence comparison methods used are based on heuristics, which are faster but there are no guarantees that the best alignments are produced. The sequences for the same gene in a group of species will be more different the more distant phylogenetically they are. Sequences will get mutations over time, so the more time has passed since two species split, the more mutations will have their sequences and the more different their sequences will be.

Mutations can be residue (nucleotide or amino-acid) substitutions, insertions or deletions. Aligning similar sequences by any algorithm usually creates alignments that are usually correct, but when sequences are very different aligning them could be a challenge. Once a long time has passed since the split of the species the sequences can be so changed by the mutations that any meaningful similarities could have been lost an creating a meaningful alignment could be very difficult.



Short sequence aligners

- Bowtie
- BWA
- GSNAP
- SOAP 10

Long sequence aligners

- BLAT
- BWA-SW
- Exonerate
- GMAP
- MUMmer 11

USES

Alignments could be used to:

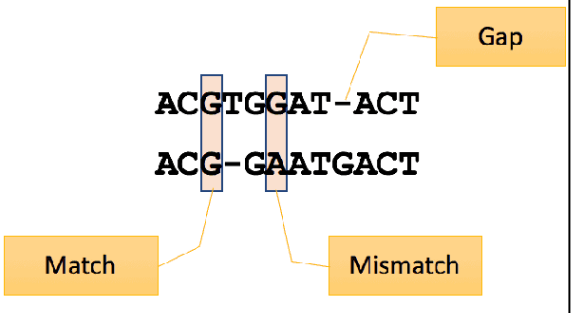
- Quantify the phylogenetic distance between two sequences
- Look for functional domains
- Compare a mRNA with its genomic region
- Identify polymorphisms and mutations between sequences

STEPS

- The first step to compare two sequences is, usually, to align them.
- There could be substitutions, changes of one residue withanother, or gaps.
- Gaps are missing residues and could be due to a deletion in onesequence or an insertion in the other sequence.
- Gaps complicate the alignments. Algorithms should take intoaccount the possibility of introducing gaps and once we allowthem to create gaps several alignments can be constructed between two sequences.

We can devise different scoring schemes with those measures. For instances:

- scoring schema 1: match +1, mismatch: 0, gap creation: -1 gap extension: -1
- scoring schema 1: match +1, mismatch: -1, gap creation: -1 gap extension: 0



$$Similarity(\%) = \frac{Match}{Match + Mismatch} \times 100$$

One alignment will have a different score under different scoring schemes. Speaking of the score of an alignment it is meaningless if we do not take into account the scoring schema used. It also has no sense tocompare the scores of different alignments done under different schemes.

Once we have decided which scoring schema to use, the alignment algorithm should try to create thealignment that obtains the maximum score under that particular scoring schema.

Fitness Score:

A Fitness Score is given to each individual which shows the ability of an individual to “compete”. The individual having optimal fitness score or near optimal are sought. The GAs maintains the population of n individuals along with their fitness scores.

The individuals having better fitness scores are given more chanceto reproduce than others. The individuals with better fitness scores are selected who mate and produce betteroffspring by combining chromosomes of parents.

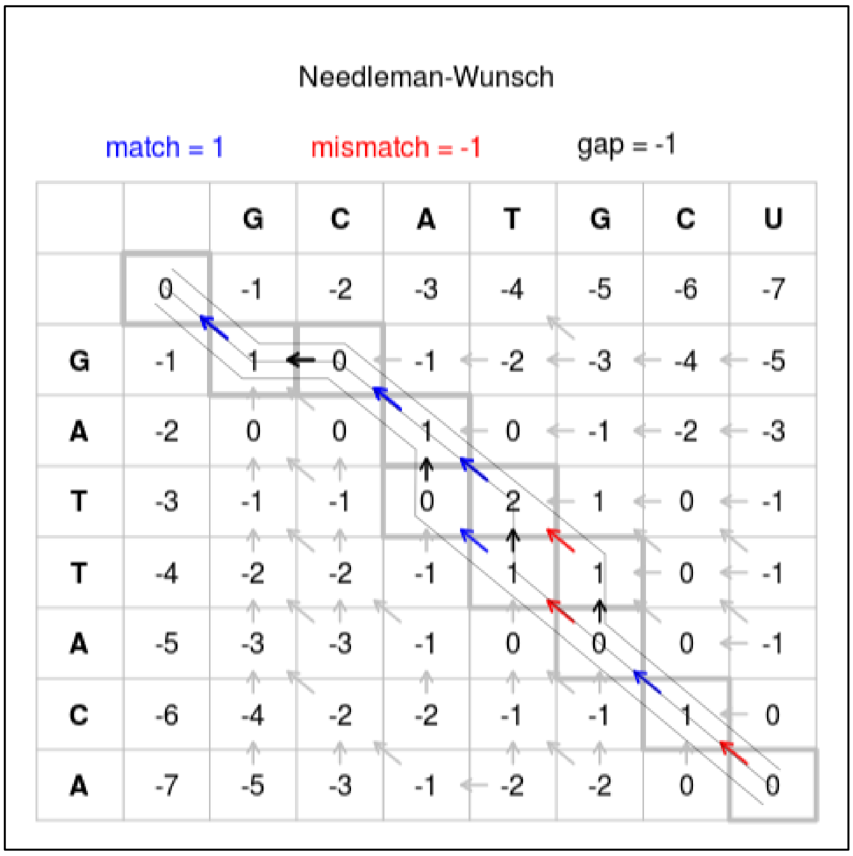
The population size is static so the room has to be created for new arrivals. So, some individuals die and get replaced by new arrivals eventually creating new generation when all the mating opportunity of the old population is exhausted. Each new generation has on average more “better genes” than the individual of previous generations.

Thus, each new generation has better “partial solutions” than previous generations. Once the off-springs produced having no significant difference than offspring produced by previous populations, the population is converged. The algorithm is said to be converted to a set of solutions for the problem.

METHODS

We tested two cutting-edge global sequence alignment methods, Needleman-Wunsch algorithm (NWA),together with their local modifications, Smith-Waterman algorithm (SWA), for aligning records.

Needleman-Wunsch Algorithm


Needleman-Wunsch

match = 1 mismatch = -1 gap = -1

		G	C	A	T	G	C	U
	0	-1	-2	-3	-4	-5	-6	-7
G	-1	1	0	-1	-2	-3	-4	-5
A	-2	0	0	1	0	-1	-2	-3
T	-3	-1	-1	0	2	1	0	-1
T	-4	-2	-2	-1	1	1	0	-1
A	-5	-3	-3	-1	0	0	0	-1
C	-6	-4	-2	-2	-1	-1	1	0
A	-7	-5	-3	-1	-2	-2	0	0

It was firstly developed by Saul B. Needleman and Christian D. Wunsch in 1970 . It was one of the first application of dynamic programming to align and compare protein and nucleotide sequences. As a globalalignment method, NWA introduces a gap rather than warping and filling in an adjacent element when aligning sequences. Therefore, every index in one sequence matches another index or a gap in the other sequence, and the monotonic increase of the mapping indices is maintained.

Mathematically, given two temporal sequences of events

X ([X1, X2, …, Xi, …, Xn]) and ***Y*** ([Y1, Y2, …, Yj, …, Ym]), NWA calculates an accumulated scorematrix ***A****(n + 1) x (m + 1)* by updating the matrix element Ai, j according to the following equation,

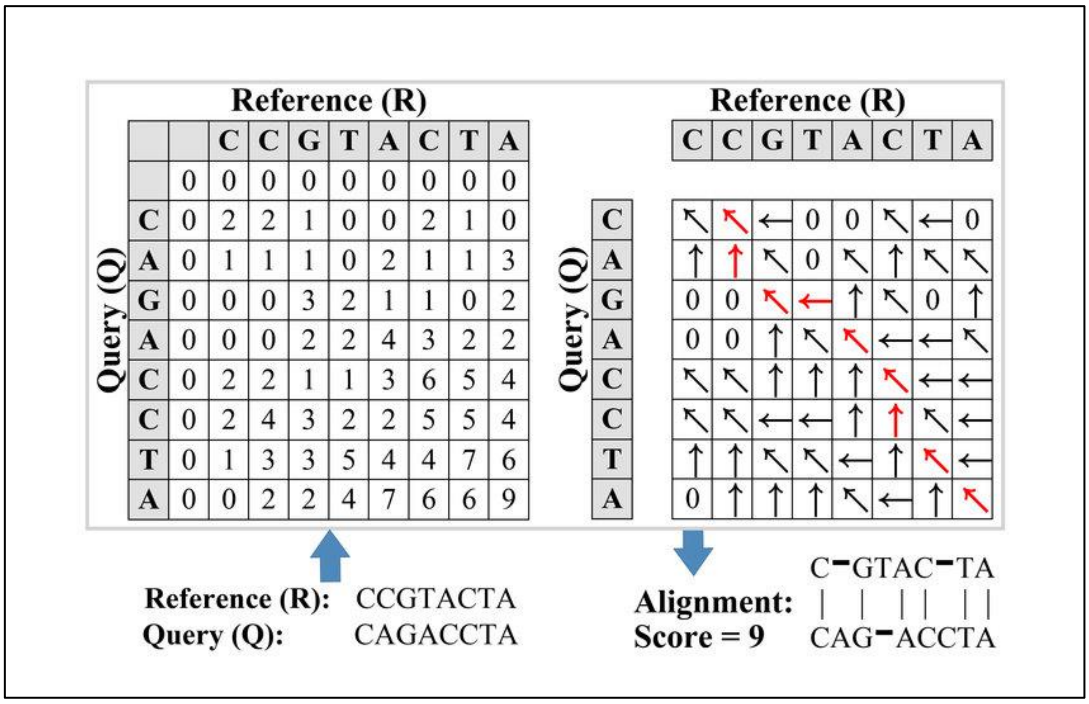
$$A_{i,j} = \begin{cases} 0 & i = 0, j = 0 \\ j * gp & i = 0, j > 0 \\ i * gp & i > 0, j = 0 \\ \max\left(A_{i-1,j-1} + s\left(X_i, Y_j\right), A_{i-1,j} + \mathrm{gp}, A_{i,j-1} + \mathrm{gp}\right) & i > 0, j > 0 \end{cases}$$

Where gp stands for a gap penalty; s (Xi, Yj) denotes the similarity between two elements Xi and Yj in the

sequence of ***X*** and ***Y***, and is calculated using a scoring system shown in Fig. 1(B).

NWA also identifies an optimal alignment path relative to a given scoring system including gap penalty bytracking back from the matrix element A(n + 1), (m + 1) and maximizing the accumulated scores along the path.

Smith-Waterman Algorithm



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

	C	C	G	T	A	C	T	A
C	↖	↖	←	0	0	↖	←	0
A	↑	↑	↖	0	↖	↑	↖	↖
G	0	0	↖	←	↑	↖	0	↑
A	0	0	↑	↖	↖	←	←	↖
C	↖	↖	↑	↑	↑	↖	←	←
C	↖	↖	←	←	↑	↑	↖	←
T	↑	↑	↖	↖	←	↑	↖	←
A	0	↑	↑	↑	↖	←	↑	↖

It is a local sequence alignment algorithm developed by Temple F. Smith and Michael S. Waterman in 1981,which is a variation of NWA for local sequence alignment. SWA has been commonly used for aligning biological sequence, such as DNA, RNA or protein sequences. obtains the best local alignments at the expense of very high computing power and huge memory requirements. In this article, we present and evaluate our experiments with three strategies to run the Smith-Waterman algorithm in a cluster of workstations using a distributed shared memory system. Our results on an eight-machine cluster presented very good speedups and indicate that impressive improvements can be achieved, depending on the strategy used. Also, we present some theoretical remarks on how to reduce the amount of memory used.

Given two temporal sequences of medical events ***X*** ([X_1, X_2, …, X_i, …, X_n]) and ***Y*** ([Y_1, Y_2, …, Y_j, …,Y_m]), SWA calculates an accumulated score matrix ***A****(n + 1) x (m + 1)* by updating the matrix element $A_{i,j}$ according to the following equation,

$$A_{i,j} = \begin{cases} 0 & i = 0 \text{or} j = 0 \\ \max\left(A_{i-1,j-1} + s\left(X_i, Y_j\right), A_{i-1,j} + \text{gp}, A_{i,j-1} + \text{gp}, 0\right) & i > 0, j > 0 \end{cases}$$

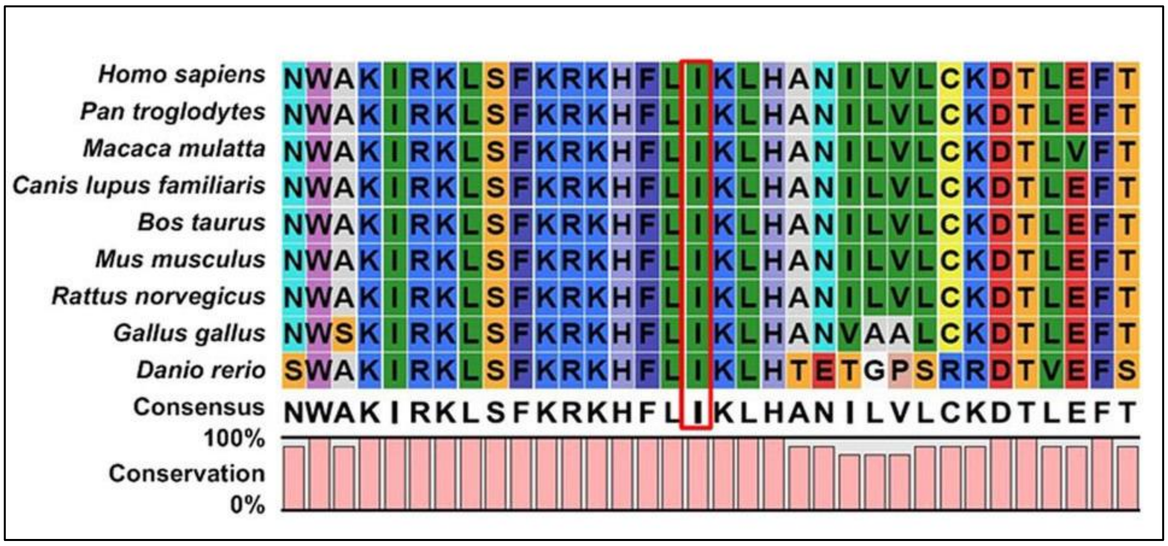
Where gp stands for a gap penalty; s (X_i, Y_j) denotes the similarity between two elements X_i and Y_j in thesequence of ***X*** and ***Y***, and is calculated using a scoring system

More than one alignment

In the dot plot graphical results, we saw that some time there could be no just one but several valid alignments. Usually, the alignment software that implements the Smith-Waterman algorithm will only printjust one alignment by default, the higher scoring one.

Multiple Sequence Alignment

The simultaneous alignment of many nucleic acid or amino acid sequences is one of the most commonly used techniques in sequence analysis. Multiple alignments are used to help predict the secondary or tertiary structure of new sequences; to help demonstrate homology between new sequences and existing families; to help find diagnostic patterns for families; to suggest primers for PCR and as an essential prelude to phylogenetic reconstruction. Multiple DNA sequence alignmentis one of the important research topics of bioinformatics. Because of the huge length of DNA sequences of advanced organisms, some researchers used divide-and-conquer techniques to cut the sequences for decreasing the space complexity for sequence alignment. Because the cutting points of sequences of the existing methods are fixed at the middle or the near-middle points, the performance of sequence alignment of the existing methods is not good enough.


Homo sapiens NWAKIRKLSFKRKHFLIKLHANILVLCKDTLEFT
Pan troglodytes NWAKIRKLSFKRKHFLIKLHANILVLCKDTLEFT
Macaca mulatta NWAKIRKLSFKRKHFLIKLHANILVLCKDTLVFT
Canis lupus familiaris NWAKIRKLSFKRKHFLIKLHANILVLCKDTLEFT
Bos taurus NWAKIRKLSFKRKHFLIKLHANILVLCKDTLEFT
Mus musculus NWAKIRKLSFKRKHFLIKLHANILVLCKDTLEFT
Rattus norvegicus NWAKIRKLSFKRKHFLIKLHANILVLCKDTLEFT
Gallus gallus NWSKIRKLSFKRKHFLIKLHANVAALCKDTLEFT
Danio rerio SWAKIRKLSFKRKHFLIKLHTETGPSRRDTVEFS
Consensus NWAKIRKLSFKRKHFLIKLHANILVLCKDTLEFT
100%
Conservation
0%

Multiple Sequence Alignment using Genetic Algorithm

Genetic algorithms (GAs) are also widely used to build MSA. GAs is helpful in MSA because they can be implemented independently of the objective function. Thus, GA algorithm can define multipleevaluations regardless of any modification in the optimization procedure. GAs can also be easily parallelized to significantly reduce the computational time.

Methodology for Multiple Sequence Alignment

A multiple sequence alignment A is defined as a two-dimensional character array over the alphabet Σ’ , where Σ ‘ = Σ υ {−}. Given an input set of strings S1, Sk, the alignment array A has K rowsand each row Ai is the alignment for string Si. Each MSA induces a pairwise alignment Ai,j on the pair of sequences Si , Sj . Then the MSA is represented as the alignment array.

Differences

The main difference from NWA is that the matrix element with negative accumulated score is set to zero, which is used to mask certain mismatched alignments and render locally matched alignments visible. Sequentially, by starting at the element with the highest accumulated score, the algorithm identifiesthe local alignment path with the highest similarity by tracking back and choosingthe path affiliated with maximal accumulated score until the matrix element with an accumulated score of zero is encountered. The algorithm is also guaranteed to find the optimal local alignment with respect to the predefined scoring system.

Traceback

A simpler score matrix table that contains only the pivot of each cell box calculation is constructed from the original position-dependent arrays of the score matrix table. Arrow pointers are used to direct a path from the highest score or an optimal value in the matrix (which actually occurs at the lower right end corner of the matrix) and traced back to the next biggest value of the predecessors until we reach the intersection of the first row and the column with the initial gap.

Alignment Generation from a Traceback

To write the sequence characters that appear from the optimal alignment path of the traceback stage, the following steps are followed:

- When the arrow is diagonal, write both characters in the alignment
- When the arrow is vertical, write the corresponding horizontal character,and in place of the vertical character, leave a gap
- When the arrow is horizontal, write the corresponding vertical character,and in place of the horizontal character, leave a gap

Thus, for both the vertical and horizontal arrow positions, one character and a gap are written for the alignment, where the gap explicitly replaces a character position in the alignment. An alignment can only be inferred as the best if the optimal value from the score matrix table corresponds to the alignment score calculation.

SCORE MATRIX

The score matrix is a tabular box constructed to keep count of score results. The score matrix for the Needleman–Wunsch algorithm and Smith-Waterman Algorithm begins with an initialization process and ends with the calculation of cell boxes.

Calculation of Cell Boxes in the Score Matrix

- Fill the initialized gap values first on both the horizontal axis and the vertical axis with a defined constant gap value score.

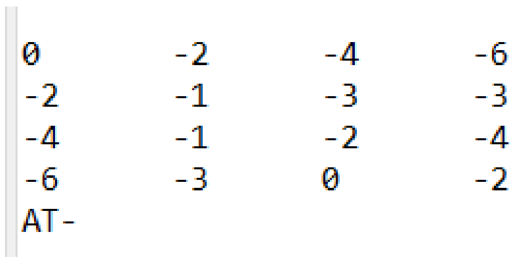
- Follow with the calculation of each cell box having the three position-dependent arrays (left/beside, right/bottom, or diagonal)
- Allow only a match/mismatch value for a “diagonal” position, and allowthe “bottom” or “beside” positions to take linear gap values only.
- For each computed cell box values, find the maximum score and let thatbe the pivot.
- The pivot of a computed cell box directly affects the next cell boxes in therow or the column.

NEEDLEMAN WUNSCH ALGORITHM JAVA

```
package secondSemester;

import java.util.Arrays;

public class NW {

    //This is the filling function for NW. Returns the max score at the
given location
    public int score(int[][] m, String s1, String s2, int i, int j) {
        int u_score = m[i - 1][j] - 2;
        int s_score = m[i][j - 1] - 2;

        int d_score = 0;

        if (s1.charAt(j - 1) == s2.charAt(i - 1))
            d_score += m[i - 1][j - 1] + 1;
        else d_score += m[i - 1][j - 1] - 1;
        int[] l = {u_score, s_score, d_score};
        return Arrays.stream(l).max().getAsInt();
    }

    //main function
    public int[][] nw(String s1, String s2) {
        int m = s2.length() + 1;
        int n = s1.length() + 1;

        int mat[][] = new int[m][n];
        for (int i = 0; i < m; i++)
            for (int j = 0; j < n; j++)
                mat[i][j] = 0;


        //to fill the 1st row and column with multiples of -2
        for (int i = 1; i < m; i++)
            mat[0][i] += mat[0][i - 1] - 2;

        for (int i = 1; i < n; i++)
            mat[i][0] += mat[i - 1][0] - 2;


        //filling loop
        for (int i = 1; i < n; i++)
            for (int j = 1; j < n; j++)
                mat[i][j] = score(mat, s1, s2, i, j);
        return mat;
    }

    public void print_mat(int[][] m) {
        for (int i = 0; i < m.length; i++) {
            System.out.println();
            for (int j = 0; j < m[0].length; j++)
                System.out.print(m[i][j] + "\t");
        }
        System.out.println();
    }

    //returns the character and the next location to trace at
    public String[] traceback_score(int[][] mat, String s1, String s2,
int i, int j) {
        int a = 0, b = 0;
        String sequence="";
        int u_score = mat[i - 1][j];
        int s_score = mat[i][j - 1];

        int max_score = Math.max(u_score, s_score);
```

```java
        if (s1.charAt(j - 1) == s2.charAt(i - 1)) {
            a = i - 1;
            b = j - 1;
            sequence += s1.charAt(j - 1);
        } else if (max_score == u_score) {
            a = i - 1;
            b = j;
            sequence = "-";
        } else if (max_score == s_score) {
            a = i;
            b = j - 1;
            sequence = "-";
        }

        String[] res={Integer.toString(a),Integer.toString(b),sequence};
        return res;
    }

    public String traceback(int[][] mat,String s1, String s2){
        int m=s2.length();
        int n= s1.length();
        String s="";

        while (m>0 && n>0){
            String[] info=traceback_score(mat,s1,s2,m,n);
            m=Integer.parseInt(info[0]);
            n=Integer.parseInt(info[1]);
            s=s+info[2];
        }
        s=reverse(s);
        System.out.println(s);
        return s;
    }

    private static String reverse(String str){
        char ch[]=str.toCharArray();
        String rev="";
        for(int i=ch.length-1;i>=0;i--){
            rev+=ch[i];
        }
        return rev;
    }
    public static void main(String[] args) {
        NW test = new NW();
        int[][] m = test.nw("ATC", "CAT");
        test.print_mat(m);
        test.traceback(m,"ATC", "CAT");
    }

}
```

OUTPUT: -



```
0       -2      -4      -6
-2      -1      -3      -3
-4      -1      -2      -4
-6      -3      0       -2
AT-
```

SMITH WATERMAN ALGORITHM JAVA

```java
package secondSemester;

import java.util.Arrays;

public class SW {
    public int score(int[][] m, String s1, String s2, int i, int j) {
        int u_score = m[i - 1][j] - 2;
        int s_score = m[i][j - 1] - 2;

        int d_score = 0;

        if (s1.charAt(j - 1) == s2.charAt(i - 1))
            d_score += m[i - 1][j - 1] + 1;
        else
            d_score += m[i - 1][j - 1] - 1;

        int[] l = {u_score, s_score, d_score};
        int m_score =Arrays.stream(l).max().getAsInt();
        if (m_score < 0)
        return 0;
    else
         return m_score;
    }

    public int[][] sw(String s1, String s2) {
        int m = s2.length() + 1;
        int n = s1.length() + 1;

        int mat[][] = new int[m][n];//zeros
        for (int i = 0; i < m; i++)
            for (int j = 0; j < n; j++)
                mat[i][j] = 0;


        for (int i = 1; i < m; i++)
            mat[0][i] = 0;

        for (int i = 1; i < n; i++)
            for (int j = 1; j < n; j++)
                mat[i][j] = score(mat, s1, s2, i, j);
        return mat;
    }

    public void print_mat(int[][] m) {
        for (int i = 0; i < m.length; i++) {
            System.out.println();
            for (int j = 0; j < m[0].length; j++)
                System.out.print(m[i][j] + "\t");
        }
        System.out.println();
    }

    public String[] traceback_score(int[][] mat, String s1, String s2,
int i, int j) {
        int a = 0, b = 0, s=0;
        String sequence="";

        int u_score = mat[i - 1][j];
        int s_score = mat[i][j - 1];

        int max_score = Math.max(u_score, s_score);
        if (s1.charAt(j - 1) == s2.charAt(i - 1)) {
            a = i - 1;
            b = j - 1;
            sequence += s1.charAt(j - 1);
            s = mat[i-1][j-1];
        } else if (max_score == u_score) {
            a = i - 1;
            b = j;
            sequence += '-';
            s = mat[i-1][j];
        } else if (max_score == s_score) {
            a = i;
            b = j - 1;
            sequence += '-';
            s = mat[i][j-1];
        }

        String[]
res={Integer.toString(a),Integer.toString(b),sequence,Integer.toString(s)
};
        return res;
    }

    public String traceback(int[][] mat,String s1, String s2){
        int m=s2.length();
        int n= s1.length();
        String s="";

        int a=0,b=0;


        for (int i = 0; i < m+1; i++) {
            for (int j = 0; j < n+1; j++)
                if(mat[i][j]== getMaxValue(mat)){
                    a=i;b=j;
                }
        }

        int breaker=10;

        while (m>0 && n>0 && breaker!=0){
            String[] info=traceback_score(mat,s1,s2,m,n);
```

```java
            m=Integer.parseInt(info[0]);
            n=Integer.parseInt(info[1]);
            s=s+info[2];
            breaker=Integer.parseInt(info[3]);
        }
        s=reverse(s);
        System.out.println(s);
        return s;
    }

    public static int getMaxValue(int[][] numbers) {
        int maxValue = numbers[0][0];
        for (int j = 0; j < numbers.length; j++) {
            for (int i = 0; i < numbers[j].length; i++) {
                if (numbers[j][i] > maxValue) {
                    maxValue = numbers[j][i];
                }
            }
        }
        return maxValue;
    }

    public static String reverse(String str){
        char ch[]=str.toCharArray();
        String rev="";
        for(int i=ch.length-1;i>=0;i--){
            rev+=ch[i];
        }
        return rev;
    }

    public static void main(String[] args) {
        SW test = new SW();
        int[][] m = test.sw("ATC", "CAT");
        test.print_mat(m);
        test.traceback(m,"ATC", "CAT");
    }
}
```

OUTPUT: -

```
0       0       0       0
0       0       0       1
0       1       0       0
0       0       2       0
AT-
```

MULTIPLE SEQUENCE ALIGNMENT

ArrayIndexComparator.java

```java
import java.util.Comparator;

public class ArrayIndexComparator implements Comparator<Integer>
{
    private final Double[] array;

    public ArrayIndexComparator(Double[] array)
    {
        this.array = array;
    }

    public Integer[] createIndexArray()
    {
        Integer[] indexes = new Integer[array.length];
        for (int i = 0; i < array.length; i++)
        {
            indexes[i] = i; // Autoboxing
        }
        return indexes;
    }

    @Override
    public int compare(Integer index1, Integer index2)
    {
         // Autounbox from Integer to int to use as array indexes
        return array[index1].compareTo(array[index2]);
    }
}
```

ArrayIndexComparator1.java

```java
import java.util.Comparator;

public class ArrayIndexComparator1 implements Comparator<Integer>
{
    private final Integer[] array;

    public ArrayIndexComparator1(Integer[] array)
    {
        this.array = array;
    }

    public Integer[] createIndexArray()
    {
        Integer[] indexes = new Integer[array.length];
        for (int i = 0; i < array.length; i++)
        {
            indexes[i] = i; // Autoboxing
        }
        return indexes;
    }

    @Override
    public int compare(Integer index1, Integer index2)
    {
         // Autounbox from Integer to int to use as array indexes
        return array[index1].compareTo(array[index2]);
    }
}
```

ArrayIndexComparator2.java

```java
import java.util.Comparator;
```

```java
public class ArrayIndexComparator2 implements Comparator<Integer>
{
    private final Integer[] array;

    public ArrayIndexComparator2(Integer[] array)
    {
        this.array = array;
    }

    public Integer[] createIndexArray()
    {
        Integer[] indexes = new Integer[array.length];
        for (int i = 0; i < array.length; i++)
        {
            indexes[i] = i; // Autoboxing
        }
        return indexes;
    }

    @Override
    public int compare(Integer index1, Integer index2)
    {
         // Autounbox from Integer to int to use as array indexes
        return array[index2].compareTo(array[index1]);
    }
}
```

GeneticAlgorithm.java

```java
package secondSemester;

import java.util.Arrays;

public class GeneticAlgorithm {
      String[][] seq = new String[125][3];
      String[] initialSeq = {"QWGDNGFFKJPKXTTTTTTTTTT  ",
"AAQWGDNGFFKJPKXYYYYYYYYY", "BBBQWGDNGFFKJPKXZZZZZZZ"};
      String[][] child = new String[125][3];
      String[][] child1 = new String[125][3];
      double random;
      int randomVal;
      String newSeq = null;
```

```java
	int extraSpace = 5;
	int totalLength = initialSeq[0].length()+extraSpace;
	String[] abc = {"QWGDNGFFKJPKXTTTTTTTTTT", 
"AAQWGDNGFFKJPKXYYYYYYYY", "BBBQWGDNGFFKJPKXZZZZZZZ"};
	int[] scoreMatrix = new int[125];
	int[] childScoreMatrix = new int[125];
	int score = 0;
	int totalScore = 0;
	int max_score = 0;
	int max_index;
	int[] modifiedScoreMatrix = new int[125];
	double[] rouletteRange = new double[125];
	double[] randomCrossOver = new double[250];
	Double[] sortedRandomCrossOver = new Double[250];
	int[] parentMappingCrossOver = new int[250];
	Integer[] indexes;
	Integer[] indexesSelection;
	String[] sortedSelectionfinal = new String[250];
	Integer[] dummyindexes;
	int[] probIndex = new int[50];
	Integer[] sortedscores = new Integer[250];
	String[] sortedAlignement = new String[250];
	String[][] tempSeq = new String[125][3];


	public void initialSequenceWithGaps(){
		for(int i=0; i<3; i++){
			for(int j=0; j<extraSpace; j++){

				initialSeq[i] = initialSeq[i].concat("_");
			}
		}
	}

	public void initialSequenceDuplicateWithGaps(){
		for(int i=0; i<3; i++){
			for(int j=0; j<extraSpace; j++){

				abc[i] = abc[i].concat("_");
			}
		}
	}
```

```java
      public int randomValue(int length){
              random = Math.random()*length;
              randomVal = (int) Math.floor(random);
          return randomVal;
      }

      public int randomValueForCrossOver(){
          random = Math.random()*124;
          randomVal = (int) Math.floor(random);
      return randomVal;
      }

     public void population(){
      for(int i=0; i<125;i++){
          for(int p=0; p<3; p++){
              initialSeq[p] = abc[p];
          }
          for(int j=0; j<3; j++){
              for(int k=0; k<5; k++){
                  int value = randomValue(initialSeq[j].length());

                  newSeq = initialSeq[j].substring(0,value);
                  newSeq = newSeq.concat("_");
                  newSeq =
newSeq.concat(initialSeq[j].substring(value,totalLength-1));

                  initialSeq[j]=newSeq;
              }
              //System.out.println(initialSeq[j]);
              seq[i][j]=newSeq;
          }
      }
      System.out.println("initial1 "+seq[0][0]);
      System.out.println("initial2 "+seq[0][1]);
      System.out.println("initial3 "+seq[0][2]);
     }

     public void crossOver(){
      for(int i=0; i<125; i++){
          for(int j=0; j<3; j++){

              child[i][j] =
seq[parentMappingCrossOver[dummyindexes[i*2]]][j];
```

```
                String children = child[i][j].substring(0, 
child[i][j].length()/2);
                String childrenNextHalf = 
child[i][j].substring(child[i][j].length()/2, child[i][j].length());
                child1[i][j] = 
seq[parentMappingCrossOver[dummyindexes[i*2+1]]][j];
                /*child1[i][j] = seq[2][j];
                System.out.println(child1[i][j]);*/
                String children1 = "";
                String childrenNextHalf1 = "";
                int subStringLen = children.length();
                int count =0;
                for(int m=0; m<subStringLen; m++){
                    if(children.charAt(m) != '_')
                        count++;
                }
                for(int p=0; p<child1[i][j].length() && count>0; p++){
                    if(child1[i][j].charAt(p) != '_'){
                        count--;
                    }
                    children1 = children1 + child1[i][j].charAt(p);
                }
                childrenNextHalf1 = 
child1[i][j].substring(children1.length());
                child[i][j] = children.concat(childrenNextHalf1);
                child1[i][j] = children1.concat(childrenNextHalf);
            }
            int maxSeqLength = 0;
            int maxSeqLength1 = 0;
            for(int j=0; j<3; j++){
                if(maxSeqLength < child[i][j].length()){
                    maxSeqLength = child[i][j].length();
                }
                if(maxSeqLength1 < child1[i][j].length()){
                    maxSeqLength1 = child1[i][j].length();
                }
            }
            for(int j=0; j<3; j++){
                while(maxSeqLength > child[i][j].length()){
                    child[i][j] = child[i][j].concat("_");
                }
                while(maxSeqLength1 > child1[i][j].length()){
                    child1[i][j] = child1[i][j].concat("_");
```

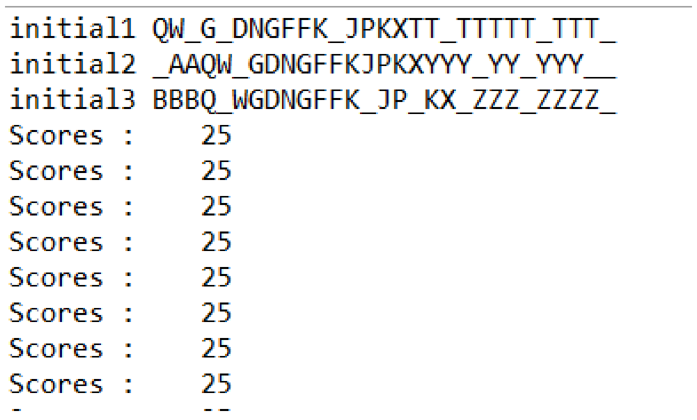
```
                        }
                }
        }
        //System.out.println(child1[0][0]);
        //System.out.println(child1[0][1]);
        //System.out.println(child1[0][2]);
    }

    public void scoringFunctionForChild() {
        for(int i=0; i<125; i++){
                int alignmentLength = child[i][0].length();
                int alignmentLength1 = child1[i][0].length();
                int childScore = 0;
                int childScore1 = 0;
                for(int j=0; j<alignmentLength; j++){
                        /*if(child[i][0].charAt(j) == child[i][1].charAt(j) &&
child[i][1].charAt(j) == child[i][2].charAt(j) && child[i][0].charAt(j)
!= '_'){
                                childScore++;
                        }*/
                        if((child[i][0].charAt(j) == child[i][1].charAt(j)) &&
child[i][0].charAt(j) != '_'){
                                childScore++;
                        }
                        if((child[i][1].charAt(j) == child[i][2].charAt(j)) &&
child[i][1].charAt(j) != '_'){
                                childScore++;
                        }
                        if((child[i][0].charAt(j) == child[i][2].charAt(j)) &&
child[i][0].charAt(j) != '_'){
                                childScore++;
                        }
                }
                for(int j=0; j<alignmentLength1; j++){
                        if(child1[i][0].charAt(j) == child1[i][1].charAt(j) &&
child1[i][1].charAt(j) == child1[i][2].charAt(j) &&
child1[i][0].charAt(j) != '_'){
                                childScore1++;
                        }
                }
                childScoreMatrix[i] = childScore;
                if(childScore<childScore1)
                {
```

```
                child[i][0] = child1[i][0];
                child[i][1] = child1[i][1];
                child[i][2] = child1[i][2];
                childScoreMatrix[i] = childScore1;
            }

            /*if(childScoreMatrix[i]>max_score)
            {
                max_score = childScoreMatrix[i];
                max_index = i;
            }*/
        }
        //System.out.println("child1 "+child[0][0]);
            //System.out.println("child2 "+child[0][1]);
            //System.out.println("child3 "+child[0][2]);
    }

    public void scoringFunction(){
        for(int i=0; i<125; i++){
            int alignmentLength = seq[i][0].length();
            score = 0;
            for(int j=0; j<alignmentLength; j++){
                /*if(seq[i][0].charAt(j) == seq[i][1].charAt(j) &&
seq[i][1].charAt(j) == seq[i][2].charAt(j) && seq[i][0].charAt(j)!= '_'){
                    score++;
                    totalScore++;
                }*/
                if((seq[i][0].charAt(j) == seq[i][1].charAt(j)) &&
(seq[i][0].charAt(j)!= '_')){
                    score++;
                    totalScore++;
                }
                if((seq[i][1].charAt(j) == seq[i][2].charAt(j)) &&
(seq[i][1].charAt(j)!= '_')){
                    score++;
                    totalScore++;
                }
                if((seq[i][0].charAt(j) == seq[i][2].charAt(j)) &&
(seq[i][0].charAt(j)!= '_')){
                    score++;
                    totalScore++;
                }
            }
            modifiedScoreMatrix[i] = totalScore;
            scoreMatrix[i] = score;
        //    System.out.print(modifiedScoreMatrix[i] +" ");
        }
        //System.out.println();
        probabilityFunction();
      }

      public void combinedPopulation() {


        for(int i=0; i<125; i++){
            sortedscores[i] = scoreMatrix[i];
            sortedscores[i+125] = childScoreMatrix[i];
        }
        for(int i=0; i<125; i++){
            sortedAlignement[i] = seq[i][0] + " " +seq[i][1] + " "
+seq[i][2];
            sortedAlignement[i+125] = child[i][0] + " " +child[i][1] + " "
+child[i][2];
        }

        ArrayIndexComparator2 comparator = new
ArrayIndexComparator2(sortedscores);
        indexesSelection = comparator.createIndexArray();
        Arrays.sort(indexesSelection, comparator);
        Arrays.sort(sortedscores);

        //for(int i=0; i<250; i++)
        System.out.println("Scores : " +"   "+sortedscores[249]);

        for(int i=0; i<250; i++){
            sortedSelectionfinal[i] =
sortedAlignement[indexesSelection[i]];
        }


        /*for(int i=0; i<125; i++){
        //    tm.put(scoreMatrix[i], seq[i][0] + " " +seq[i][1] + " "
+seq[i][2]);
            tm.put(childScoreMatrix[i], child[i][0] + " " +child[i][1] + "
" +child[i][2]);
        //    tm.put(i,"abc");
        }
        System.out.println(tm.size());
        // Get a set of the entries
          Set set = tm.entrySet();

          // Get an iterator
          Iterator iterator = set.iterator();

          while(iterator.hasNext()) {
            Map.Entry me = (Map.Entry)iterator.next();
            System.out.print(me.getKey() + ": ");
            System.out.println(me.getValue());
            int j=0;
            sortedscores[j] = Integer.valueOf((Integer) me.getKey());
            sortedAlignement[j] = String.valueOf((String) me.getValue());
            j++;
          }*/


          double top = 125*2*0.2;
        int topPopulation = (int)top;

        for(int i=0; i<topPopulation; i++){
            // Storing in a temp array for time being...need to be moved
to seq[i][j] ==> tempSeq
            String[] temparr = sortedSelectionfinal[i].split("\\s+");
            for(int j=0; j<3; j++){
                 tempSeq[i][j] = temparr[j];
                 //System.out.println(tempSeq[i][j]);
            }
        }
        probabilityFunctionForSelection(topPopulation);
        //System.out.println("final1 "+tempSeq[0][0]);
        //System.out.println("final2 "+tempSeq[0][1]);
        //System.out.println("final3 "+tempSeq[0][2]);

      }

      public void probabilityFunctionForSelection(int top) {
        int totalScoreSelection = 0;
        for(int i=top; i<125*2; i++){
            totalScoreSelection = totalScoreSelection + sortedscores[i];
        }
```

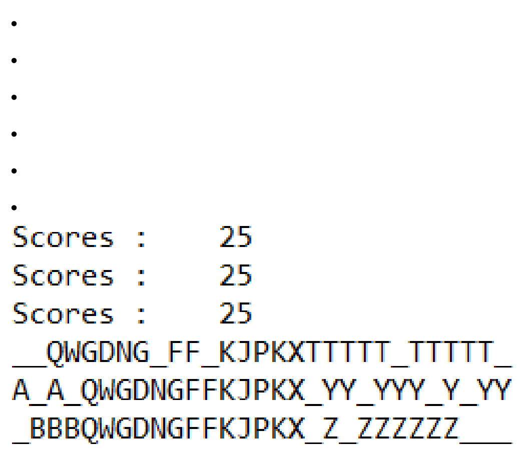
```
		double[] rouletteSelection = new double[125*2-top];

		for(int i=0; i<125*2-top; i++){
			rouletteSelection[i] = 
	sortedscores[i+top]/totalScoreSelection;
		}

		for(int i=0; i<125*2-top; i++){
			double temp = 0;
			temp = rouletteSelection[i] + temp;
			rouletteSelection[i] = temp;
		}

		double[] randomValSelection = new double[125*2-top];
		for(int i=0; i<125*2-top; i++){
			randomValSelection[i] = Math.random();
		}
		Arrays.sort(randomValSelection);

		int j=0;
		int i=top;
		while(i<125*2 && j<125-top){
			if(randomValSelection[j] > sortedscores[i]){
				i++;
			}else{
				String[] temparr1 = 
	sortedSelectionfinal[i].split("\\s+");
				for(int p=0; p<3; p++){
				tempSeq[top+j][p] = temparr1[p];
				}
				j++;
			}
		}
		for(int m=0; m<125; m++){
			for(int k=0; k<3; k++){
				seq[m][k] = tempSeq[m][k];
				//System.out.println("Vivek" +m +seq[m][k]);
			}
		}
	}

	public void probabilityFunction(){
        for(int i=0; i<125; i++){
            rouletteRange[i] = (double)modifiedScoreMatrix[i]/totalScore;
            randomCrossOver[i] = Math.random();
            randomCrossOver[i+125] = Math.random();
            sortedRandomCrossOver[i] = randomCrossOver[i];
            sortedRandomCrossOver[i+125] = randomCrossOver[i+125];
        //  System.out.print(rouletteRange[i] +" ");
        //  System.out.print(randomCrossOver[i] + " ");
        }

        //System.out.println();
        ArrayIndexComparator comparator = new
    ArrayIndexComparator(sortedRandomCrossOver);
        indexes = comparator.createIndexArray();
        Arrays.sort(indexes, comparator);


        /*for(int i=0; i<250; i++){
            System.out.print(sortedRandomCrossOver[i] +" ");
        }*/
        /*System.out.println();
        for(int i=0; i<250; i++){
            System.out.print(indexes[i] +" ");
        }*/
        //System.out.println();
        Arrays.sort(sortedRandomCrossOver);
        for(int i=0; i<250; i++){
        //  System.out.print(sortedRandomCrossOver[i] +" ");
        }

        ArrayIndexComparator1 comparator1 = new
    ArrayIndexComparator1(indexes);
        dummyindexes = comparator1.createIndexArray();
        Arrays.sort(dummyindexes, comparator1);

        //System.out.println();
        for(int i=0; i<250; i++){
        //  System.out.print(dummyindexes[i] +" ");
        }

        int j=0;
        for(int i=0; i<2*125; i++){
            while(sortedRandomCrossOver[i] >= rouletteRange[j]){
                j++;
            }
            parentMappingCrossOver[i] = j;
        }
    }

    public void mutation() {

        int count = 0;
        for(int i=0; i<125*3; i++){
            if(Math.random()<0.05) {
                probIndex[count] = i;
                count++;
            }
        }
        for(int p=0; p<count; p++){
            int i = probIndex[p]/3;
            int j = probIndex[p]%3;
            int mutationIndex = randomValue(seq[i][j].length());

            if(seq[i][j].charAt(mutationIndex)!='_'){
                        seq[i][j] = seq[i][j].substring(0, mutationIndex+1)
  + "_" + seq[i][j].substring(mutationIndex+1);
                        for(int m=0; m<j; m++) {
                            seq[i][m] = seq[i][m] + "_";
                        }
                        for(int m=j+1; m<3; m++) {
                            seq[i][m] = seq[i][m] + "_";
                        }
                }else{
                        seq[i][j] = seq[i][j].substring(0, mutationIndex) +
  seq[i][j].substring(mutationIndex+1);
                        seq[i][j] = seq[i][j] + "_";
                }
        }
        //System.out.println("mutation1 "+seq[0][0]);
        //System.out.println("mutation2 "+seq[0][1]);
        //System.out.println("mutation3 "+seq[0][2]);
    }

    public void selection() {
        combinedPopulation();
    }
```

```java
    public static void main(String[] args) {
     GeneticAlgorithm ga = new GeneticAlgorithm();
     ga.initialSequenceDuplicateWithGaps();
     ga.initialSequenceWithGaps();
     ga.population();
     for(int i=0; i<100; i++){
          ga.scoringFunction();
          ga.crossOver();
          ga.scoringFunctionForChild();
          ga.mutation();
          ga.selection();
     }
     int finalmaximum = 0;
     int finalIndex = 0;

     for(int i=0; i<125; i++){
          int alignmentLength = ga.seq[i][0].length();
          int score = 0;
          for(int j=0; j<alignmentLength; j++){
               /*if(seq[i][0].charAt(j) == seq[i][1].charAt(j) &&
seq[i][1].charAt(j) == seq[i][2].charAt(j) && seq[i][0].charAt(j)!= '_'){
                    score++;
                    totalScore++;
               }*/
               if((ga.seq[i][0].charAt(j) == ga.seq[i][1].charAt(j)) &&
(ga.seq[i][0].charAt(j)!= '_')){
                    score++;
               }
               if((ga.seq[i][1].charAt(j) == ga.seq[i][2].charAt(j)) &&
(ga.seq[i][1].charAt(j)!= '_')){
                    score++;
               }
               if((ga.seq[i][0].charAt(j) == ga.seq[i][2].charAt(j)) &&
(ga.seq[i][0].charAt(j)!= '_')){
                    score++;
               }
          }
          if(score>finalmaximum){
               finalmaximum=score;
               finalIndex = i;
          }
     }
    int tempLen = ga.seq[finalIndex][0].length();

    for(int k=0; k<tempLen; k++){
        if(ga.seq[finalIndex][0].charAt(k) ==
ga.seq[finalIndex][1].charAt(k) &&
                ga.seq[finalIndex][0].charAt(k) ==
ga.seq[finalIndex][2].charAt(k) &&
                ga.seq[finalIndex][0].charAt(k) == '_'){
            ga.seq[finalIndex][0] =
ga.seq[finalIndex][0].substring(0,k) +
ga.seq[finalIndex][0].substring(k+1);
            ga.seq[finalIndex][1] =
ga.seq[finalIndex][1].substring(0,k) +
ga.seq[finalIndex][1].substring(k+1);
            ga.seq[finalIndex][2] =
ga.seq[finalIndex][2].substring(0,k) +
ga.seq[finalIndex][2].substring(k+1);
            tempLen--;
            k--;
        }
    }

    System.out.println(ga.seq[finalIndex][0]);
    System.out.println(ga.seq[finalIndex][1]);
    System.out.println(ga.seq[finalIndex][2]);

    }
}
```

OUTPUT:



```
initial1 QW_G_DNGFFK_JPKXTT_TTTTT_TTT_
initial2 _AAQW_GDNGFFKJPKXYYY_YY_YYY__
initial3 BBBQ_WGDNGFFK_JP_KX_ZZZ_ZZZZ_
Scores :    25
Scores :    25
Scores :    25
Scores :    25
Scores :    25
Scores :    25
Scores :    25
Scores :    25
```



```
.
.
.
.
.
.
Scores :    25
Scores :    25
Scores :    25
__QWGDNG_FF_KJPKXTTTTT_TTTTT_
A_A_QWGDNGFFKJPKX_YY_YYY_Y_YY
_BBBQWGDNGFFKJPKX_Z_ZZZZZZ___
```

IMPLEMENTATION

- Dataset

  The dataset consists of five similar sequences with some variations. The model can alsobe successfully performed on the default standard datasets.

- Overview

  The populations are repeated for all generations until a good solution is obtained. Forthis, first, all the sequences are evaluated and the process is repeated for all chromosomes. This involves selecting two parents and obtaining a crossover probability. The crossover operation is applied for p1 and p2 and then a mutation is applied on the child. The child is then finally added to the new population. The obtained best value of the chromosome is added to the list and in case there is no relevant changes, the execution breaks. The best value of the chromosome is added tothe population and meanwhile the same number of worst chromosomes are removed from the list. For the implementation, the scenario of no relevant changes after 100 generations is considered.

- Algorithm Approach

  We assume that the hyphen character “ - ” is not part of the alphabet so that we can represent alignments based on insertions/deletions in the original string. The computation of an exact MSA is NP-Complete and therefore the exact method is notapplied in MSA for all but unrealistically small datasets.

- Data Pre-processing:

  1. The implementation begins with reading the input file data set. The extra spaces are removed by considering each line as an input to the list of strings. If the length of thechromosome is lesser than the maximum length, it is filled by adding gaps. Next, thechromosome sequences for each individual in a population are printed wrt the original chromosome sequence that was fed into the model.

  2. Population Initialization: In this step, the initial population is created. The python package nwalign is used to compute the pairwise alignments. This will create many different alignments of the sequences. We randomly choose any one alignment of each sequence in the list and store it. The newly created list now contains the newly computed alignment. The necessary gaps are added and the thus the initial populationis created.

  3. Genetic Algorithm Operators: Next, we iterate through each generation of the population until there is no change in the alignment detected. In each generation of population, starting from initial population: Computing Sum of Pairwise-Alignment (SOP): Each sequence list in the population is evaluated using the score with the substitution matrix, BLOSUM62 values. The pairwise scores [SOP] is computed foreach sequence list. Selection: For selecting two parents in a crossover operation, first we normalize the

fitness. Then the mating pool is built and randomly two parents p1 and p2 are selected. Cross-Over: The next step is to apply a crossover operation to the two parents p1 and p2. The horizontal crossover applied to the parents returns thechild. Mutation: In the next step, a mutation is applied to the child. For this first a mutation method is chosen and applied, followed by gap removal. Once gap removalis completed, a k-addition process is carried out to finally obtain the child.

4. New Population Creation: The next step is to complete the gaps and finally remove the useless gaps from the alignments. The python nwalign package is again called to compute the score of the pairwise alignments using a BLOSUM62 scoring matrix forthe newly computed population. Finally, the population of the new generated chromosomes are updated and similar chromosomes are removed. The new population with the better score chromosomes is hence created. The next generation of the population is then iterated for the process.
5. Termination Criteria: In a traversal of minimum 100 generations, if there are no more relevant changes in the sequence alignment of the chromosome, then the code returnsthe best chromosome score and sequence.

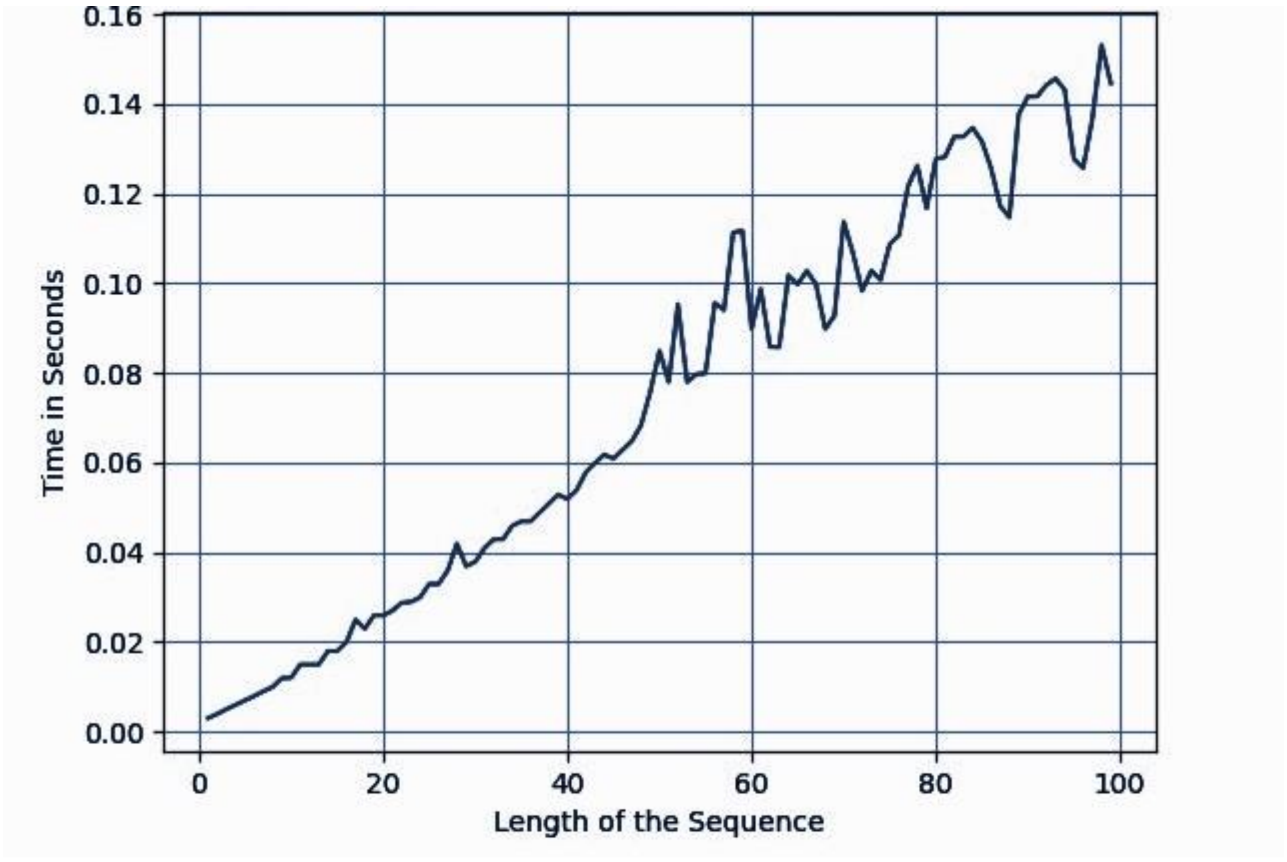
SUBSTITUTION MATRICES

A substitution matrix is a collection of scores for aligning nucleotides or amino acids with one another. These scores generally represent the relative ease with which one nucleotide or amino acid may mutate into or substitute for another, and they are used to measure similarity in sequence alignments. Substitution matrices are used in conjunction with gap costs to construct local or global alignments.

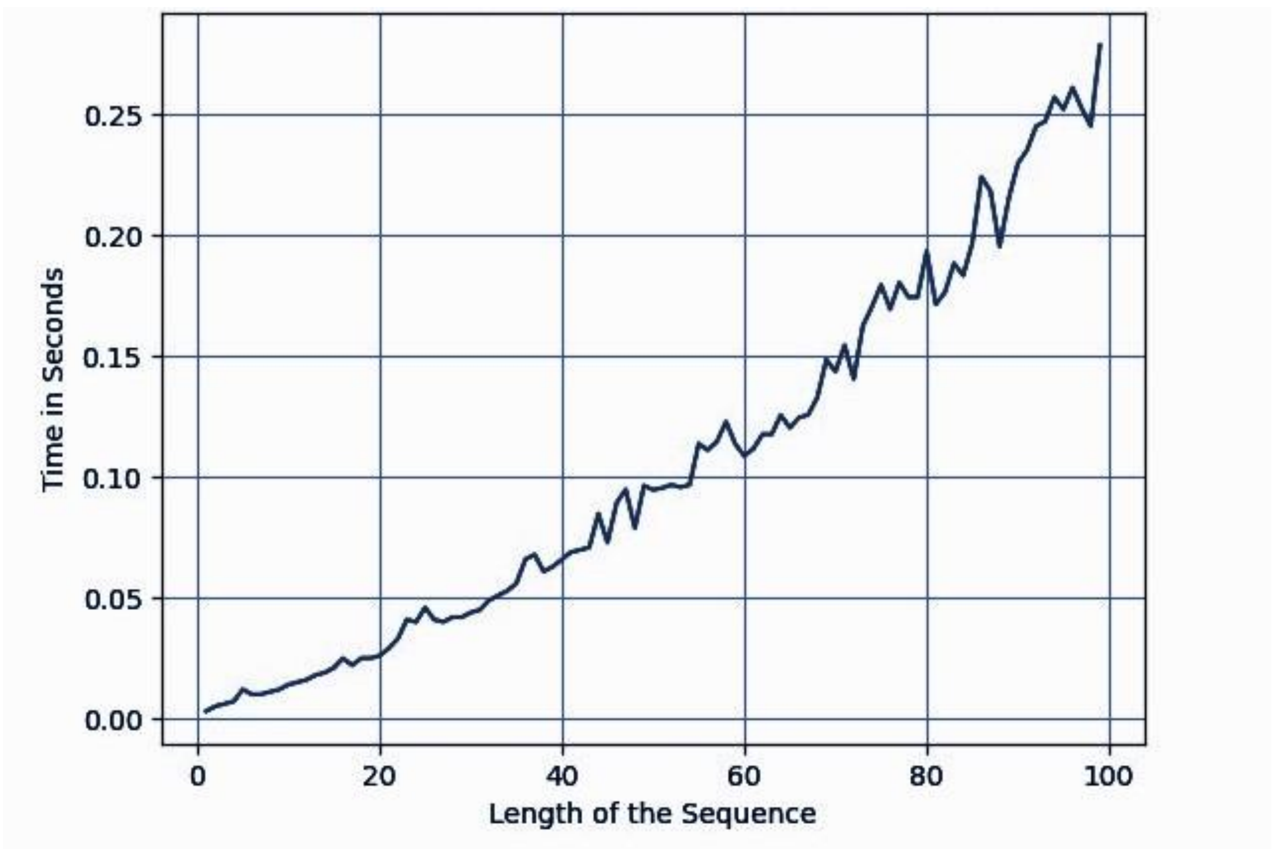
A local alignment substitution matrix implicitly corresponds to a set of target frequencies for aligned letters, which characterize the alignments the matrix may most easily distinguish from chance. Thus, different substitution matrices are optimal for comparing related sequences that have diverged to differing degrees. The point accepted mutation and blocks substitution matrix (BLOSUM) series of matrices for protein sequence comparison have been derived from collections of related sequences. Special-purposematrices may be appropriate for comparing sequences with biased nucleotide or amino acid composition.

OBSERVATION FROM CODE:

Execution time of Needleman-Wunch Algorithm



Execution time of Smith Waterman Algorithm



- When the difference in lengths of the 2 sequence is more than our score function takes more time to compute.
- when we add more gaps, which results in a greater number of combinations.
- When the Length of sequence is more then we will get memory error and it takes more time to execute.

Conclusion:

Alignment of multiple protein sequences taken are successfully implemented using genetic algorithm. The results show more than 85% accuracy for all the sequences tested.

The algorithm implemented leaves scope for improvisation in future to make the algorithm more dynamic. The number of sequences can be taken as a variable and not restricted to a fixed size as with the algorithm above.