Master Thesis

Large scale multiple genome alignment via an efficient kernel method

Ammar Qaseem

Feb. 28, 2014

Albert-Ludwigs-Universität Freiburg im Breisgau
Technische Fakultät
Institut Für Informatik
Lehrstuhl Für Bioinformatik

.

Gutachter

Prof. Dr. Rolf Backofen

Prof. Dr. Wolfgang R. Hess

Betreuer

Dr. Fabrizio Costa

Declaration

Statement of authenticity I hereby declare, that I am the sole author and composer of my thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work. I hereby also declare, that my thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Erklärung Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet habe und alle Stellen, die wörtlich oder sinngemäss aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

Freiburg im Breisgau, February 26, 2014

Unterschrift ____________________

Acknowledgment

Foremost, I would like to thank God for his blessings upon me.

A very special thank to my family in Yemen: my parents and my brothers, for supporting me spiritually throughout my life and study. Your support is the reason of being at this position.

Many thanks to Prof. **R. Backofen** who offered me the opportunity to accomplish my master thesis. I would like to express my sincere gratitude to my advisor Dr. **F. Costa** for the continuous support of my Master thesis study and research, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me during my thesis work. Also I would like to thank all the group of Bioinformatic.

Special thanks to my brother **O. Alkhnbashi**, the person who I always find beside me to help me in my study. Beside that, I thank my friends **H. Almohtaseb** , **M. Amayreh** and **M. Ghadiry** for their help in reviewing my written script.

I would like also to take the chance to thank my brother **K. Alazzani**, the person who supported me in my study and life.

Special thanks for the program co-ordinator **Ms. Epe** to give me the chance to study my master in this university.

Last but not the least, I would like to thank my chief in my HiWi **D. Spinner** who gave me all the time to do my thesis and my colleague **M. Kuhn**. My thanks to everyone who contributed to this work directly or indirectly or in any way.

Zusammenfassung

Multipel, großumfängliche genetische Anordnungen sind eine der wichtigsten Aufgaben in der genetischen Forschung. Aufgrund der ständig wachsenden Zahl von genetischen Sequenzen, ist die Verarbeitung unter Einsatz von IT immer schwerer lösbar. Daher werden effiziente Methoden benötigt, um multiple genetische Anordnungen auflösen zu können. Hier wird eine neue, zweistufige Methode vorgeschlagen. Im ersten Schritt, werden Mutationen aufgespürt. Dies ist extrem wichtig, um genetische Anordnungen im zweiten Schritt vervollständigen zu können. Die vorliegende Thesis stellt eine neue Methode zur effizienten Aufspürung von genetischen Mutationen vor. Folgendes leistet diese neue Methode: (1)Erkennung der Mutationsart sowie der Region, in der die Mutation stattfindet; (2)Beachtung von Sequenz und struktureller Informationen; (3)Ausführung in linearer Komplexität im Bezug auf die Laufzeit und des Speicherbedarfs

Abstract

Multiple large scale genome alignment is one of the most important tasks in genomic research. Since the number of sequences genomes is rapidly increasing, this problem computationally is becoming harder. Therefore, an efficient method to solve large scale multiple genome alignments is needed. We propose a new two stages method. In the first stage, mutation events are detected. This information is very important for completing genomes alignment in the second stage. This thesis introduces a new method to efficiently solve the problem of mutation events detection. The method is able to: (1) identify the regions where the mutation events happen and also the type of mutation events; (2) consider both sequence and structure information; (3) run in linear complexity both regarding runtime and memory.

Contents

CHAPTER 1

Introduction

1.1. Motivation

We are currently witnessing an explosion in the availability of genomic information as the entire genome of several species are sequenced and their information is made available at an increasing rate. In order to make use of this large amount of data however, efficient algorithmic procedures are needed.

One of the most fundamental type of processing for genomic data is that of genome alignment, whereby regions belonging to several related genomes are put in bi-univocal correspondence. As a result of the alignment procedure, information of biological relevance can be derived, such as the evolutionary conservation rate of given regions. The sequences in these regions are believed to be important and to correspond to functional biological entities like proteins and non-coding RNA. Correct alignments allow, in other terms, the (semi)automatic discovery of biological objects (either belonging to known classes, or even to yet unknown classes). However, current genomic alignment techniques are suitable for relatively closely related species, and can process a relatively small number of genomes. In order to allow alignments for thousands of genomes, novel efficient techniques are needed. The choice of computational models suitable for this task has to take into consideration several requirements: (1) efficiency: the complexity cannot exceed O(n) as typical sizes for this data ranges in the order of several tens of Gigabyte; (2) accuracy: small errors in the alignment result in unusable information; (3) flexibility: the proposed method should allow several types of information to be easily incorporated and used. Multiple sources of information can generate different types of constraints that can greatly reduce the space of feasible candidate alignments.

1.2. Objective

The goal of this thesis is finding a method that can identify and predict the position and type of mutation events that can happen in the related genomes in linear complexity, and appropriate for sequential or structural genomics. This identification can be very helpful in aligning multiple genomes.

1.3. Method Overview

The proposed method for identifying the genome mutations can be described as following: First, the genomes are split into chunks or fragments of equal size, then the k nearest neighbor of each fragments are identified, after that the fragments are represented in an 1-D layout, this representation will simplify the manipulation and management of the fragments, then construct a single large graph, after that filtering and smoothing techniques can be applied to filter inconsistent data. Compression of the data can be done by using encoding techinque, where the data can be represented using short meaningful symbols. Finally, a machine learning model is build based on the coding data, this model will be used to identify and predict the mutation events that might happen in the sequences.

1.4. Thesis Structure

The content of this work is divided into six chapters. Chapter 2 introduces a short overview of literatures of related work. Chapter 3 discusses background and most essential concepts such as graph kernel and nearest neighbors. In chapter 4, the proposed approach is provided in details. The experimental results and evaluation of this new approach is discussed in chapter 5 and finally chapter 6 gives the final conclusion and future work.

CHAPTER 2

Related Work

This chapter reviews related work to *multiple Genome alignment*. A general overview of the common techniques is presented in addition to that the chapter shows how these techniques scale with the number of genomes.

2.1. Pairwise and multiple genome alignment

Pairwise alignment methods are used to find the best-matching alignments of two genome sequences, however the multiple genome alignment is a sequence alignment for more than two sequences. The importance of multiple genome alignment is that it allows to infer the phylogenetic relationship, useful for studying organisms evolution and analyzing sequence-structure relationships.

2.2. Multiple Genome Alignment

Here it will focus on the multiple genome alignment, and it will present shortly the most well known techniques in this field of alignment.

2.2.1. Dynamic programming approaches

Dynamic programming algorithms, such as Needleman-Wunsch [4], Smith-Waterman [5], and MSA[6], are theoretically applicable to any number of sequences and can be extended to align optimal of multiple sequences. However, it is computationally expensive in terms of computation time and memory requirement, it needs $O(2^n l^n)$ and $O(l^n)$, for time and memory complexity respectively, where n is the number of sequences of equal length l. These techniques are useful when identifying an accurate alignment for very few sequences are needed, otherwise, heuristic approaches are used.

2.2.2. Heuristic approaches

These techniques are used for speed up the process of exhaustive search, and finding a satisfactory solutions, but can't guaranteed to be optimal. The general idea of this type of approaches as follows: First, identify anchors/seeds(conserved short sequence) across genomes. Then use these anchors to map collinear homologous segments across genomes, finally, align collinear segments efficently. Heuristic approaches, such as progressive and iterative alignments, are widely used.

2.2.3. The progressive alignments

Progressive techniques, do a multiple sequence alignment by first aligning the most similar sequences and then adding successively less related sequences or groups to the alignment until the entire query set has been incorporated into the solution. This type of alignments is based on pairwise comparisons FASTA [6]. All the sequence pairs are aligned, then based on the pairwise alignment scores, the distance matrix is constructed and the guide tree is produced, then the sequences are aligned guided by the similarity relationship indicated by the tree. Many variations of the Clustal [7] progressive are used for multiple sequence alignments, as well as the more accurate, but slower algorithm T-Coffee [8] is used.

2.2.4. Iterative methods

Progressive alignment results are sensitive to inaccuracies in the initial pairwise alignments. To overcome this problem, iterative approaches attempt to improve on the strongly dependence on the accuracy of initial pairwise alignments, acheiving this by making initial alignments for group of sequences and then revise the alignments to compute a more accurate result. Such approaches include many methods suc as MAFFT [9], MUSCLE [10], and PROBCONS[11].

2.2.5. Chaining alignment strategy

This type of techniques aligns multiple sequences without making a guide tree. It scores all pairwise local alignments and then chain them. This happens by looking for the maximal exact matches occurring in a multiple alignments as anchor, then trying to extend that diagonally. Eventually the diagonals with maximum sum of weights are taken as the optimal alignments. Such methods include MGA [12] and DIALIGN[13]. The problem in this technique, that when searching for anchors across multiple genomes, if this anchor occurs many times in the sequences, then it will be difficult to specified which regions should aligned. If the number of those anchors grows, then the possible combination of alignments will grow exponentially.

2.2.6. Mauve : Multiple alignments of conserved Genomic sequences with rearrangements

Mauve is a multiple whole genome alignment. It can align two or more genomes that have undergone rearrangements due to recombination and substantial amounts of segmental gain and loss. Sum-of-pairs breakpoint score is used as an objective alignment score. Mauve avoids the problem that arises due to the repetitiveness of the particular anchors, by using Multiple Maximal Unique Matches (multi-MUMs) of some minimum length k as alignment anchors. Multi-MUMs are exactly matching regions shared by two or more genomes that occur only once in those genomes and that are bounded on either side by mismatched nucleotides. Firstly, the algorithm finds local alignments multi-MUMs, then the phylogenetic guide tree are builded using multi-MUMs, after that selecting subset of multi-MUMs to be used as anchors which are partitioned into collinear groups LCBs. Finally, using the guide tree; it performs a recursive alignment of every LCB.

ProgressiveMauve: Multiple Genome Alignment with Gene Gain, Loss and Rearrangement

ProgressiveMauve is the improvement of the last version of **Mauve**. The version demonstrates some cases that were not handled in the previous version. It can align multiple genomes that have undergone rearrangements due to recombination and substantial amounts of segmental gain and loss. Sum-of-pairs breakpoint score is used as an objective alignment score, as well as probabilistic alignment filtering method to remove erroneous alignments of unrelated sequences[14].

Collinear vs. Rearrangement

Genomic regions derived from a common ancestral genomic region, having the corresponding regions arranged in the same linear order, and the collinear order is the result of common ancestry rather than due to random chance, however, rearrangement is affect the order and orientation of the blocks/regions within a genomes. Rearrangement happened when breaks occur in a sequence, then these breaks can result in a deletion, inversion, duplication or translocation. If the breaks result in a loss of a piece, it is called a deletion. Inversion results when a block of sequence breaks off, and reversed (inverted), and is reinserted into the original sequence but in reverse order. Duplication happens when multiple copies of a small segment is presence in the sequence.

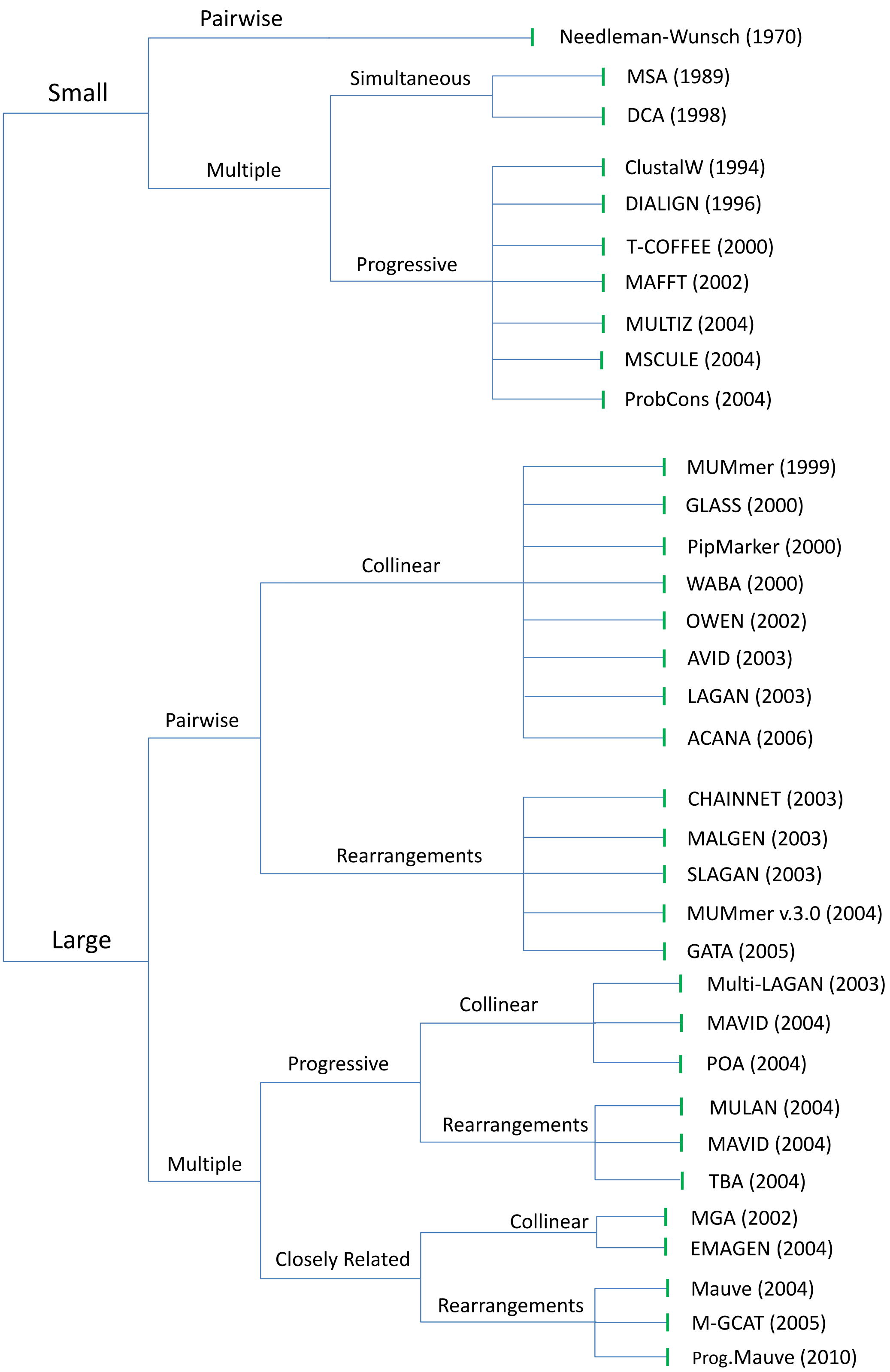


Figure 2.1.: *Tracing the growth in related global genome comparison tools. Based on[15]*

CHAPTER 3

Fundamentals

This chapter introduces the background and the most essential concepts that form the basis of this thesis. It also introduces the tools and algorithms that will be used in the proposed method.

3.1. Graphs

A graph $G(V, E)$ is an abstract structure of a set of objects V that is used to model a relation E over those objects V, such that some pair of objects are connected. Recently, graphs are used in many real-time applications such as mathematics, technology and applied science. A simple introduction about graph notations, types and representation will be introduced.

Graph notation

Let $G = (V, E)$ be an undirected graph with vertex set $V = v_1, \ldots, v_n$, such that each vertex v_i in the graph represents a data point x_i. The edge $(i, j) \in E$ if v_i and v_j

are connected. Undirected implies that there is no orientation in edges, that is (i, j) is identical to (j, i). The graph is called connected is there is a path from any vertex to another vertex in the graph.

Definition: Labeled Graph
A labeled graph is a graph whose vertices and edges are labeled from a set of symbols, this set is composed of integers or real numbers.

3.1.1. Weighted graphs

A weighted graph is a special type of labeled graphs in which the labels are represented by weights (positive real numbers). A weighted undirected graph G is associated with a weight function $w : V \times V \longrightarrow \mathbb{R}$ satisfying

$$w(u, v) = w(v, u) \quad \text{, and} \qquad w(u, v) \geq 0$$

Note that if { u,v } $\notin E(G)$, then $w(u, v) = 0$. Unweighted graphs are just the special case where all the weights are 0 or 1.

One advantage of using weighted graphs are that all definitions and manipulations of graphs can be easily handled and carried out.

Definition (Degree of a vertex)
The degree of a vertex $deg(v)$ is the number incoming edges of a vertex v. In the present context degree of v_i, $deg(v_i)$ is defined as:

$$deg(v_i) = \sum_j A_{ij}$$

which is the sum only over all vertices adjacent and have connection to v_i. The degree matrix D is defined as the diagonal matrix contains the degree of vertices $v_1 \dots v_n$ on the main diagonal. A is an adjacency matrix or (0,1) matrix, such that a_{ij} is 1 if there is connection between v_i and v_j, and 0 otherwise.

3.1.2. Similarity graphs

The aim of similarity measure is to identify the closely related pairs by assigning them a higher weight than the distantly related pairs. There are many possible interpretations for individuals to be closely related, leading to the construction of different matrices. A simple interpretation of relatedness is the average genetic distance.
Given a set of points $\chi = x_1 \dots x_n$ and similarity S between pairs, such that s_{ij} is the similarity between x_i and x_j. If the similarities between data points are the information only in the hand, then the good way of representing data points is the similarity graph $G = (V, E)$. Such that each vertex v_i in the graph represents a data point x_i, and s_{ij} is the label of the connection/link between v_i and v_j, where s_{ij} represents the similarity measure of x_i and x_j. It is assigned to 0 if there is no similarity or if the similarity is less than some threshold.

3.2. Graph Laplacians

In this section different graph Laplacians are defined, and carefully distinguish between the variants of graph Laplacians. Graph Laplacian is a matrix representation of undirected graph. The Laplacian matrix L of a graph $G = (V, E)$, is a symmetric $n \times n$ matrix, and can be defined as:

$$L = D - A$$

Where D is a *degree matrix*, and A is the adjacency matrix of the graph. Here L is unnormalized laplacian.

Definition(Adjacency Matrix) Adjacency Matrix is defined as:

$$A_{ij} = \begin{cases} 1 & \text{if } i \neq j \text{ and } v_i \text{ adjacent to } v_j \\ 0 & \text{otherwise} \end{cases}$$

Remark The weight of edge (i, j) can be used to define A_{ij} if the graph is weighted. That indicates $A_{ij} \in \mathbb{R}^+$.

Unweighted Laplacian
For unweighted graphs, the entries of the Laplacian matrix are defined as follows:

$$l_{ij} = \begin{cases} deg(v_i) & \text{if } i = j \\ -1 & \text{if } i \neq j \text{ and } v_i \text{ adjacent to } v_j \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

The diagonal elements l_{ij} of Laplacian matrix$L$ equal the degree of the vertex v_i.

Weighted Laplacian

For weighted graphs, the laplacian matrix is defined as:

$$l_{i,j} = \begin{cases} deg(v_i) - w_{ii} & \text{if } i = j \\ -w_{ij} & \text{if } v_i \text{ adjacent to } v_j \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

where w_{ij} is the weight or similarity of elements x_i and x_j. $deg(v_i)$ is degree of x_i, such is:

$$deg(v_i) = \sum_{j=1}^{n} w(i, j)$$

which is the sum over all weights of adjacent vertices of v_i.

3.2.1. Symmetric normalized Laplacian

The symmetric normalized Laplacian is defined as

$$L_{norm} = I - D^{-1/2} A D^{-1/2} \tag{3.3}$$

where I is identity matrix, which is n Ã n square matrix with ones on the main diagonal and zeros elsewhere. A is the adjacency matrix and D is the degree matrix. Since the degree matrix D is diagonal, its reciprocal square root $D^{-1/2}$ is simply defined as a diagonal matrix, having diagonal entries which are the reciprocals of the positive square roots of the corresponding positive diagonal entries of D. The symmetric normalized Laplacian is a symmetric matrix.

The symmetric normalized Laplacian of unweighted graph can also be written as

$$L_{norm} = I - D^{-1/2}AD^{-1/2} = D^{-1/2}LD^{-1/2} \tag{3.4}$$

Normalized unweighted Laplacian

Simply normalized Laplician for unweighted graphs is defined using the following formula:

$$l_{ij} = \begin{cases} 1 & \text{if } i = j \text{ and } deg(v_i) \neq 0 \\ -\frac{1}{\sqrt{deg(v_i)deg(v_j)}} & \text{if } i \neq j \text{ and } v_i \text{ adjacent to } v_j \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

Normalized weighted Laplacian
For a weighted graphs, where the weight of edges are used, the Laplacian matrix is defined as[16]:

$$l_{ij} = \begin{cases} 1 - \frac{w_{ii}}{deg(v_i)} & \text{if } i = j \text{ and } deg_i \neq 0 \\ -\frac{w_{ij}}{\sqrt{deg(v_i)deg(v_j)}} & \text{if } i \neq j \text{ and } v_i \text{ adjacent to } v_j \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

3.3. Eigenvalues and Eigenvectors

In many contexts, a vector can be assumed to be a list of real numbers(elements).

Definition(scalar multiples)
Two vectors are called to be *scalar multiples* of each other (also called parallel or collinear) if they have the same number of elements, and if every element of one vector is multiplied by the same scalar to obtain the corresponding element in the other vector.

Definition
Square matrix A of $n \times n$ can by multiplied by a vector v of $n \times 1$, the result will be a vector w of $n \times 1$ as follow :

$$\begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

is mapped to

$$\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} A_{1,1} & A_{1,2} & \dots & A_{1,n} \\ A_{2,1} & A_{2,2} & \dots & A_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n,1} & A_{n,2} & \dots & A_{n,n} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

$$w = Av$$

where

$$w_i = A_{i,1}v_1 + A_{i,2}v_2 + \cdots + A_{i,n}v_n = \textstyle\sum_{j=1}^{n} A_{i,j}v_j$$

the vectors v and $A\ v$ are parallel if there is some real number λ such that $A\ v = \lambda\ v$. v is called an eigenvector of A, and the scale λ is called the corresponding eigenvalue to that eigenvector v. Each eigenvector is paired with a corresponding eigenvalue, any such pair $(\lambda,\ v)$ is called an eigenpair for A. Eigenvectors are a special set of vectors associated with a linear system of equations (i.e. a matrix equation)(Marcus and Minc 1988, p. 144).

Eigenvalues are multipliers, they represent how much stretching has taken place or scaled up. Eigenvalues and eigenvectors have many applications in computer science and applied mathematics. The second smallest eigenvector is used to segment the graph into different groups.

3.3.1. Computing the eigenvectors

An eigenvector of a square matrix A is a non-zero vector v that is when the matrix is multiplied by v, yields a constant multiple λ of v. That is:

$$Av = \lambda v \tag{3.7}$$

The scalar λ is called the eigenvalue of A corresponding to v. Once the exact value of an eigenvalue is known, the corresponding eigenvector can be computed by finding the non-zero solutions of the eigenvalue equation, which yields a system of linear equations with known coefficients.

3.3.2. Arnoldi Algorithm

The standard method that are used to compute all the exact eigenvalues and corresponding eigenvectors is relatively expensive in runtime O(n^3) and memory O(n^2). In addition, the standard algorithms can not solve very large scale problems is very expensive and time consuming. The standard algorithms are not appropriate for many tasks, that are looking for a few(say k) eigen pairs, such as those of largest and smallest magnitude. The other possibility is to use an efficient algorithms that can compute a few approximate eigen pairs, like the largest and smallest values in linear runtime and memory. *Arnoldi algorithm* is one of those algorithms, it is a technique for approximating a few eigenvalues and corresponding eigenvectors of a general $n \times n$ matrix in a linear runtime and memory.

ARPACK

ARPACK is a collection of Fortran77 subroutines designed to solve large scale eigenvalue problems. ARPACK stands for ARnoldi PACKage. This software is based upon an algorithmic variant of the Arnoldi process called the Implicitly Restarted Arnoldi Method (IRAM). The software is designed to compute a few approximate eigen pairs such as those of largest or smallest eigen pairs using $n \cdot \mathcal{O}(k) + \mathcal{O}(k^2)$ storage. It is appropriate for large structured matrices $\mathbf{A}$ where structured means that a matrix-vector product $\mathbf{w} \leftarrow \mathbf{Av}$ requires $\mathcal{O}(n)$ rather than the usual $\mathcal{O}(n^2)$ floating point operations[3]. The stopping condition is determined by the user through the parameter *tol*, which is the precision of approximation. Typically, small of tol will increase the work required to satisfy the stopping criterion. However, setting *tol* too large may cause eigenvalues to be missed. Care should be taken, when setting this parameter, because it possible setting *tol* so small that convergence never occurs.

Remark. For more details about this PACKAGE see[3].

3.4. Spectral graph layout

In graph theory, the spectral approaches have became a standard technique. These techniques utilize the eigenvalues and corresponding eigenvectors of the laplacian matrix of a graph. The spectral method for graph visualization computes the layout of the graph using certain eigenvectors of related matrices. The spectral methods graph layout have two significant attractive advantages. The first is the ability of those methods to compute the optimal layout. Whereas the second advantage is the efficient computation time.

The basic idea behind the spectral graph layout to compute the eigenvectors of the laplacian matrix of a graph is as follow: For a give undirected graph $G = (V, E)$, the laplacian matrix is defined as:

$$L = D - A$$

Where D is a *degree matrix*, and A is the adjacency matrix of the graph. Here L is laplacian matrix. To compute the eigenpairs, the following equation have be solved:

$$Lu = \lambda u \tag{3.8}$$

The scalar λ is called the eigenvalue of L corresponding to the eigenvector u. Once the exact value of an eigenvalue is known, the corresponding eigenvector can be computed by finding the non-zero solutions of the eigenvalue equation, which yields a system of linear equations with known coefficients.

The eigenvalues for solving **Eq. 3.8** are $0 = \lambda_1 <= \lambda_2 \ldots \lambda_n$, and the corresponding eigenvectors are $1 = u_1, u_2 \ldots u_n$.

Since L is a real, symmetric matrix hence all it's eigenvalues are real, and the corresponding eigenvectors are orthognals. The trivial eigenvector of L that associated with the smallest eigenvalue[1] is $1_n = [1, 1, \ldots, 1]^T$.

Based on laplacian matrix, a two dimensional spectral layout of a graph G is defined as $p = (u_2, u_3)$ where u_2 and u_3 are the second and the third smallest eigenvectors corresponding to λ_1 and λ_2.

The method of spectral graph drawing can used to reduce the dimensionality of the objects, such that the large graphs (especially mesh graphs) tends to look pretty simple and easy to understand and manipulate. The intuition behind using such technique that it displays a particular arrangement of the vertices and edges of a given graph in a low dimensionality. The projection of the vertices onto the plane reflects the similarity between those vertices, the closer vertices the more similar to each other.

[1] The smallest eigenvalue of L is 0.

3.5. Graph Kernels

In real world many data are represented as graphs, such data include sequences, phylogentic trees, RNA structures (Durbin et al., 1998). All computations in *kernel methods* are done using kernel functions. A kernel function is an inner product of two vectors in feature space. Recently, using kernels in machine learning became very popular.
Graph similarity is essential for most learning tasks such as clustering and classification on graphs.
Graph kernel is a kernel function that computes an inner product on graphs. It compares substructures of graphs that are countable in polynomial time and measures the similarity of pairs in graphs.

3.5.1. Neighborhood Subgraph Pairwise Distance Kernel (NSPDK)

The NSPDK is the fast graph kernel that is suitable for large datasets of sparse graphs with discrete vertex and edge labels. The NSPDK is a decomposition kernel, it operates over all possible subgraphs of graphs of finite size that are defined by a specific relation function. The relation function decomposes graph G into all possible pairs of neighborhood subgraphs of radius r and exact distance d.

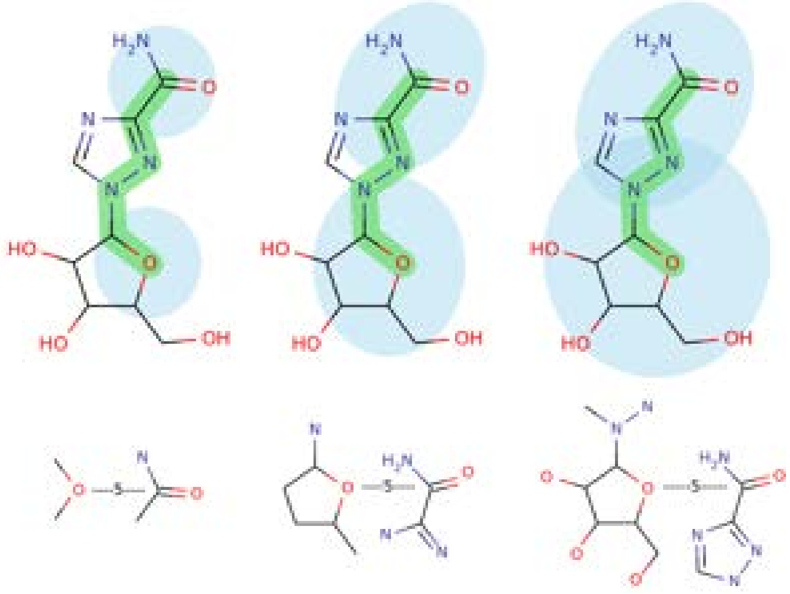


Figure 3.1.: *Neighborhood subgraph pairs for radius r = 1,2,3 and distance d = 5 [1].*

Definition : Neighborhood subgraph

The neighborhood subgraph of radius r of vertex v, is defined as the subgraph of G rooted in v induced by the set of vertices at a distance less than or equal to r and denoted by $N_r^v(G)$, where the distance between two vertices u and v, denoted $D(v, u)$ is the number of edges of the shortest path between u and v.

Definition : Bijection

A bijection or bijective function denoted as $f : X \rightarrow Y$, is a function between elements of two sets. It is one to one function, where every element of X is mapped onto exactly one element of Y, and vice versa.

Definition : Graph isomorphism

An isomorphism of graphs G and G' is a bijection between the vertex sets of G and G'.
$f : V(G) \rightarrow V(G')$.
such that any two vertices u and v are adjacent in graph G if and only if $f(u)$ and $f(v)$ are adjacent in G'.

$$uv \in E(G) \iff f(u)f(v) \in E(G')$$

If this relation holds, then G and G' are called isomorphic and it can be written as $G \simeq G'$.

Definition : Neighborhood pair relation $R_{r,d}$

The neighborhood pair relation $R_{r,d}$ of two neighborhood subgraphs of radius r, is defined to hold when the distance between the roots of these subgraphs are exactly equal to d. The kernel function $\kappa_{r,d}$ over $G \times G$ in NSPDK is defined as the decomposition kernel on the relation function $R_{r,d}$, such that :

$$\kappa_{r,d}(G \times G') = \sum_{\substack{A_v, B_u \in R_{r,d}^{-1}(G) \\ A'_{v'}, B'_{u'} \in R_{r,d}^{-1}(G')}} \delta(A_v, A'_{v'})\delta(B_u, B'_{u'}) \tag{3.9}$$

where $\delta(x, y)$ is an exact matching kernel, that is :

$$\delta(x, y) = \begin{cases} 1 & \text{if } x \simeq y \\ 0 & \text{otherwise} \end{cases} \tag{3.10}$$

R^{-1} is the inverse of the relation $R_{r,d}$, that yield all the possible pairs of neighborhood subgraphs of radius r in the given graph G, where the distance between the roots of the each subgraphs pair is d.
Simply, we can say that $\kappa_{r,d}$ is just a function to count the number of identical pairs of neighboring graphs of radius r at distance d between two graphs.

Finally, the unnormalized NSPDK is defined as the sum of all kernels over all radii and distances. For efficiency reasons, NSPDK limits the sum of the kernels $\kappa_{r,d}$ for all increasing values of r and d from zero up to a specific given value for r^* and d^*.

$$K_{r^*,d^*}(G,G') = \sum_{r=0}^{r^*} \sum_{d=0}^{d^*} \kappa_{r,d}(G,G') \tag{3.11}$$

The normalized version of $\kappa_{r,d}$ is suggested, to ensure an equal weight to relations of all orders, regardless of the size of the induced part sets.

$$\bar{\kappa}_{r,d}(G,G') = \frac{\kappa_{r,d}(G,G')}{\sqrt{\kappa_{r,d}(G,G)\kappa_{r,d}(G',G')}} \tag{3.12}$$

3.6. Sparse Matrix

A sparse matrix is a matrix that allows special techniques to take advantage of the large number of zero elements. It is a special type of matrices, where only non-zero values are provided. It is common, In many applications to deal with very large matrices with a significant number of zero-valued elements, this type of matrices is known as a sparse matrix. The opposite of sparse is dense matrix, such that all the entries of the matrix are stored explicitly, even if some of them are zeros. Standard dense matrix structures and algorithms that are working on them are relatively slow and memory consuming. When storing and manipulating sparse matrices it is necessary to use special data structures and algorithms that take advantage of the sparse-matrix structure. In case of using sparse matrices formats, memory consumption can be reduced. Sparse data are relatively easy to be compressed, this compression almost results in a significantly less memory usage. In addition to the significantly reduced memory, the performance can be increased by using an efficient algorithm that can manipulate and exploit the sparse matrices features.

Sparse Matrix Representation

The basic data structure for a matrix is a two-dimensional array. Each entry in the array represents an element a_{ij} of the matrix and can be accessed by the two indices i and j. Data structures used to maintain sparse matrices must only access the non-zero elements of a matrix in a manner which facilitates efficient implementation. There are many formats to store sparse matrices. One of the simplest formats is *(row,col,value)*, where each non-zero element is represented by two indices i, j (which is the position of this element in the matrix) and the value of this element(**Fig. 3.2**). Substantial memory requirement reductions can be realized by storing only the non-zero entries. Depending on the number and distribution of the non-zero entries, different data structures can be used which saves memory when compared to the basic approach. In general any components of a sparse matrix that are exactly 0 should not be stored, but in some cases it is more computationally efficient to allow a few to be included.

Memory Reduction

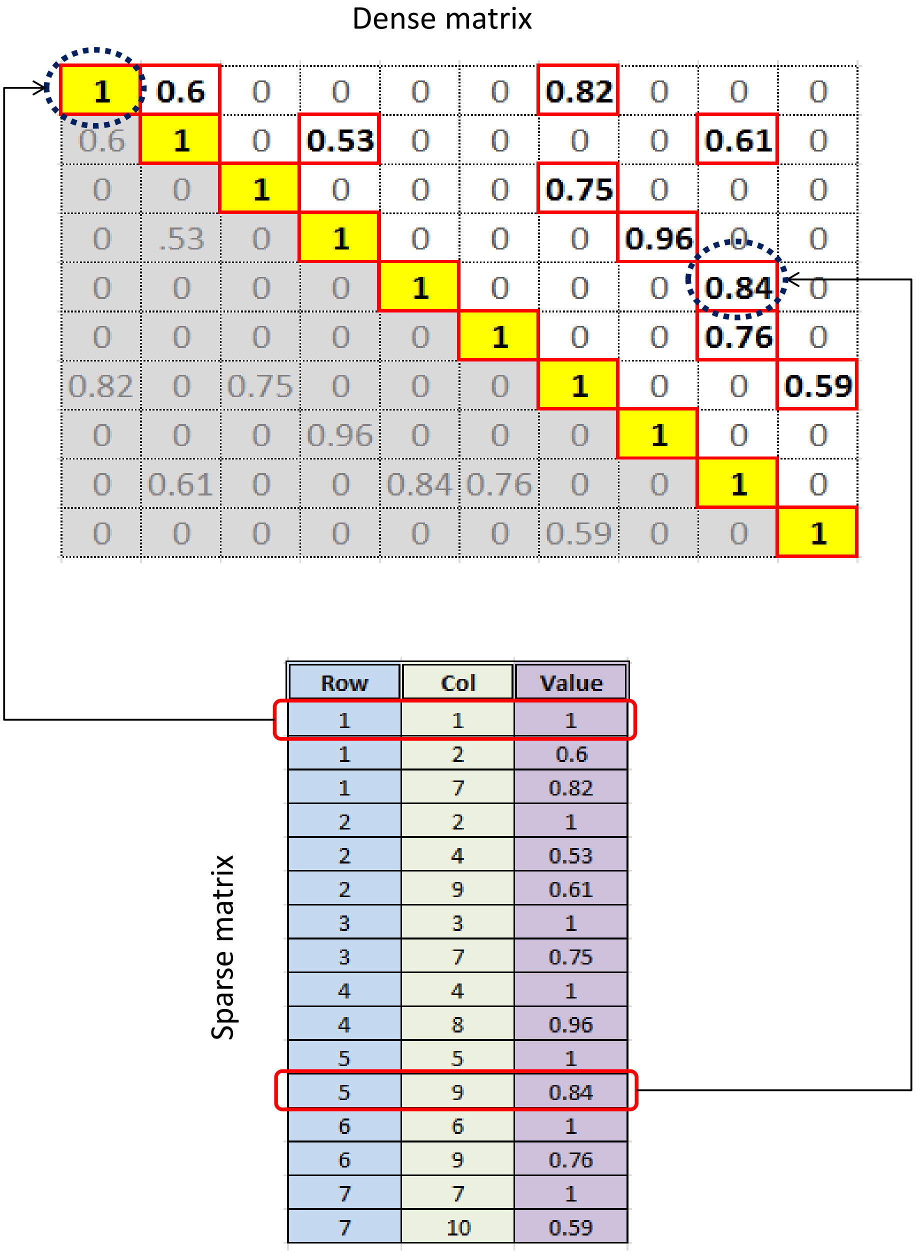
For large symmetric matrices that have a very large number of zero elements, sparse matrix techniques are the best strategy to store such matrices. In this case, it is necessary to store only the upper triangular half of the matrix (upper triangular format) or the lower triangular half of the matrix (lower triangular format). This will result in a significant reduction in memory storage, and saves time of manipulation. The memory usage is reduced from O(n^2) to O(n).

Example:
Given a 100,000 × 100,000 symmetric matrix, and 8$bytes$ to store each element. Suppose that each row has k non-zero elements ($k \ll n$).

Store the matrix based on standard structure
100,000 × 100,000 = 10^{10} elements, 8bytes to store each one then in total we need 8×10^{10} bytes$\sim$ 80GB).

Store the matrix sparsely
Each element is connected to k (say nearest neighbors), assume that $k = 150$. There are 100,000 × 150 non-zero elements. Since the matrix is symmetric, it is necessary

to store only the upper triangular half and the diagonal of the matrix. In total, only $75 \times 10^5 \times 8$bytes $= 60 \times 10^6$ bytes are needed to store. In addition, some memory is needed to store the indices which in total needs $\sim$ (400MB).



Dense matrix

1	0.6	0	0	0	0	0.82	0	0	0
0.6	1	0	0.53	0	0	0	0	0.61	0
0	0	1	0	0	0	0.75	0	0	0
0	.53	0	1	0	0	0	0.96	0	0
0	0	0	0	1	0	0	0	0.84	0
0	0	0	0	0	1	0	0	0.76	0
0.82	0	0.75	0	0	0	1	0	0	0.59
0	0	0	0.96	0	0	0	1	0	0
0	0.61	0	0	0.84	0.76	0	0	1	0
0	0	0	0	0	0	0.59	0	0	1

Sparse matrix

Row	Col	Value
1	1	1
1	2	0.6
1	7	0.82
2	2	1
2	4	0.53
2	9	0.61
3	3	1
3	7	0.75
4	4	1
4	8	0.96
5	5	1
5	9	0.84
6	6	1
6	9	0.76
7	7	1
7	10	0.59

Figure 3.2.: *Sparse matrix contains many zero elenemts. And the (row,col,value) representation of this matrix sparsely.*

3.7. Smoothing and Filtering

In statistics and signal processing, smoothing a data set means creating an approximating function that attempts to capture important patterns in the data, while leaving out noise and inconsistent data[17].
Smoothing can be used in two important ways that can help in data analysis
(1) By being able to extract more information from the data set as long as the assumption of smoothing is reasonable.
(2) By being able to give analysis that are robust and flexible [16]. It is some kind of noise reduction.
In smoothing, the data points of a signal are modified, so the points that are higher than the adjacent points are reduced, and those that are lower are increased to reduce the noise and get smoother signal. Smoothing and filtering inconsistent elements is typically a pre-processing step to improve the quality of the results for later processing. Smoothing methods often associated with a tuning parameter which is used to control the extent of smoothing.

3.7.1. Median Smoothing/Filtering

It is one of the simplest smoothing algorithms. It is a very widely technique used in digital signals because it preserves edges while removing noise. In this algorithm, each point in the signal is replaced with the median of m preceding and following adjacent points, where m is a positive integer called the smooth width or smooth window. Usually m is an odd number.

Median smoothing description
The main idea of the *median smoothing* is to run through the signal entry by entry, replacing each entry with the median of neighboring entries. The pattern of neighbors is called the "window", which slides entry by entry over the entire signal. For 1-D signals, the window is the k preceding and following entries. Usually k is an odd number, and the median is simply the median value after the entries in the window are sorted numerically ascending or descending.
Suppose the data set entries form are $(x_1, x_2, ...x_n)$, these entries can be replaced by the smoothing data set $(y_1, y_2, ...y_n)$, where the smoothed value y_i is computed as:

$$y_i = median[x_{i-k}, x_{i-k+1}, \ldots, x_{i+k-1}, x_{i+k}] \tag{3.13}$$

where k is the number of preceding and following entries, and the window size is $2k+1$. x_i is replcaed by y_i in the set.

Example :
Given a simple data set signal x = [3 20 7 9 10]
Smoothing this signal using the median method with window size 1, will be as follow:

y_1 = Median[3 3 20] $\xrightarrow{sorting}$ Median[3 3 20]= 3
y_2 = Median[3 20 7] $\xrightarrow{sorting}$ Median[3 7 20] = 7
y_3 = Median[20 7 9] $\xrightarrow{sorting}$ Median[7 9 20] = 9
y_4 = Median[7 9 10] $\xrightarrow{sorting}$ Median[7 9 10] = 9
y_5 = Median[9 10 10] $\xrightarrow{sorting}$ Median[9 10 10] = 10

This means that the signal X will be replaced by the smoothed one Y = [3 7 9 9 10].

$$[3\ 20\ 7\ 9\ 10] \Longrightarrow [3\ 7\ 9\ 9\ 10]$$

Remark. In the example above, because there is no entry preceding the first value (following the last one), the first value is repeated (last value is repeated), to obtain enough entries to fill the window.

3.8. Mutation

In biology, mutation is a random change in the base pair sequence of the genetic material (usually DNA or RNA) of an organism. Mutations result from unrepaired damage to DNA or RNA genomes (typically caused by either chemical or physical mutagens), errors in the process of replication, or from the insertion or deletion of segments of DNA by mobile genetic elements[18].

Mutations in genes can either have no effect, alter the product of a gene, or prevent functioning of the gene properly or completely. Changing the structure of a gene, results in a variant form which may be transmitted to subsequent generations. Usually, mutations are not always bad, when variations occur within genes, there is often a consequence. The cell is able to repair most of these changes, and rarely cause death or disease. Mutations can be classified into two main types, the small-scale and large-scale mutations.

3.8.1. Small-scale mutation

A small-scale mutation affects a single or few base pairs of gene, where one base pair is replaced by another.

- *Point mutation.* A point mutation is a type of small-scale mutation, it happens when a single base pair is altered.
- *Insertions.* One or more extra base pair are added into genetic material.
- *Deletion.* Remove one or more base pair from the genetic material.

3.8.2. Large-scale mutation

A large-scale mutation affects a large regions of the sequence.

- *Insertion.* An insertion changes the number of base pairs in a gene by adding a part of genetic material(DNA or RNA). As a result, the protein made by the gene may not function properly.
- *Deletion.* A deletion changes the number of bases by removing a region of genetic material(DNA or RNA). Larger deletions can remove an entire gene or several neighboring genes. The deletion may alter the function of the resulting proteins.
- *Duplication.* A duplication consists of a piece of DNA that is abnormally copied one or more times. This type of mutation may alter the function of the resulting protein.
- *Inversion.* In this type of mutation the orientation of a sequence segment is reversed.

Fig. 3.3 shows the different types of mutations.

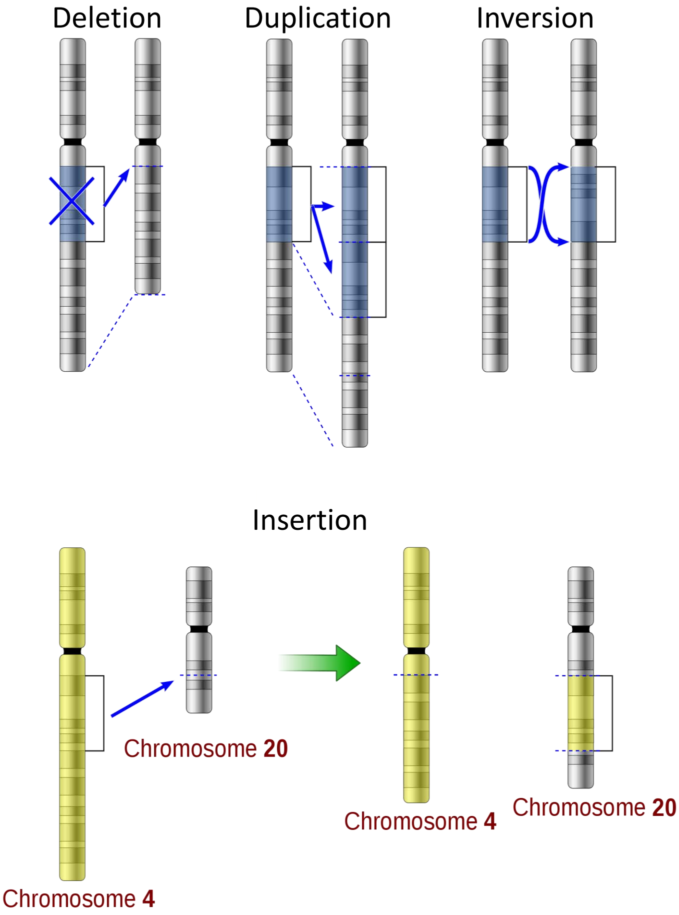


Figure 3.3.: *Types of mutations. Taken from[19].*

CHAPTER 4

Method

The following chapter presents a detailed description of the proposed model for solving large scale multiple genomes alignment. A pipeline model is proposed, that can efficiently identify the regions of genome where the mutation events happened for large, multiple, and closely related genomes using a graph kernel technique.
In this model, six phases could be distinguished (**Fig. 4.1**):

(1) Preprocessing phase, initially, the genome sequences are splitted into chunks/fragments of equal sizes. (2) Then an efficient graph kernel technique (**NSPDK**) is used to find and retreive the k nearest neighbors of each fragment. (3) Using the k nearest neighbors, a single large graph is constructed to show the relation between the fragments. (4) After that, a spectral graph representation is used, to reduce the dimensionality and layout the fragments in 1-D representation. (5) Smoothing and filtering technique is applied to remove noise and inconsistent data. (6) Encoding the 1-D layout into a discrete meaningful sequence/string to use machine learing on these sequences to make a predictive models of genomic mutation events. Finally, create the prediction model that can detect the mutation events.

In the following, each of those phases will be described and analyzed in further details.

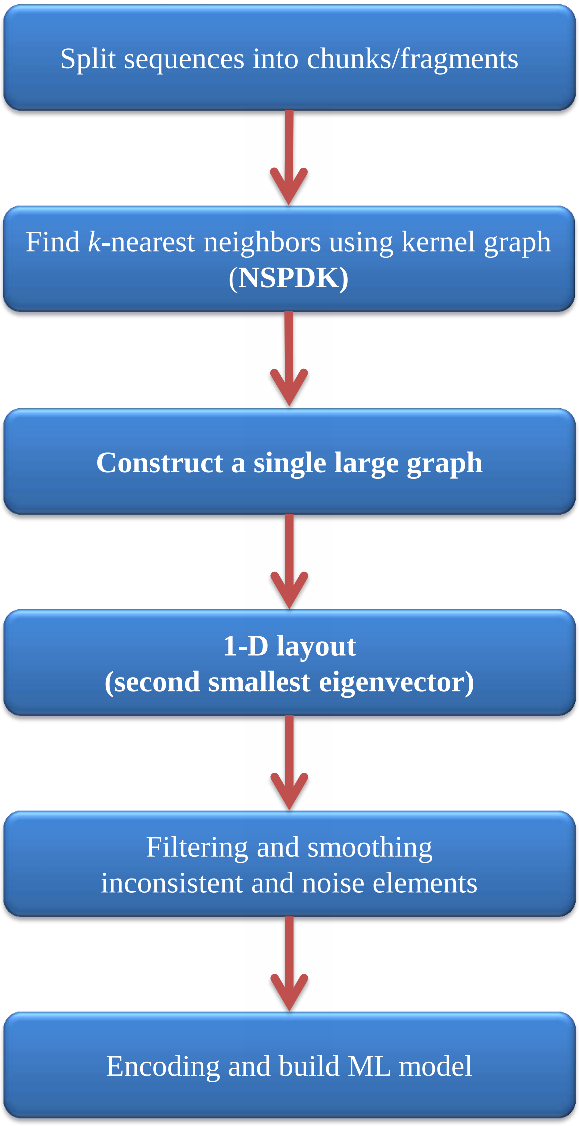

Split sequences into chunks/fragments
Find *k*-nearest neighbors using kernel graph **(NSPDK)**
Construct a single large graph
1-D layout (second smallest eigenvector)
Filtering and smoothing inconsistent and noise elements
Encoding and build ML model

Figure 4.1.: *The pipeline phases of the proposed model.*

4.1. APPROACH :

In the following, the phases of the proposed model will be described and analyzed each of those phases of the model in further details.

4.1.1. Step 1 : Splitting

In this preprocessing step, all input genome sequences are splitted into chunks/fragments of equal length (say 250 nucleotides). This length reflects the expected length of the signals to be found. Slightly larger or smaller window is usually ok, but too small windows can disturb existing signals, and too large window can result in losing the existing mutation events. Reasonable window size is between 75-300. The last fragment is considered a bit special in order to obtain equal length fragments. The shift of the sliding window is managed via *"win_shift"* option. Relative window size in % for window shift or overlap between the adjacent windows during input preprocessing are more recommended, to extract most of the informations of the signals. Small shifts usually increase the quality of the results, but yield much more fragments and more runtime for manipulation. Higher shifts result in less fragments and less runtime, but can losing more information about the signals and less quality of getting the nearest neighbors. Reasonable percentage for window shift is between 25-75% of the window size(**Fig. 4.2**).

Remark. We can also use *RNA Shape* can be used to establish the secondary structure, then feed this into the next step. The proposed method consider both the sequence and structure information.

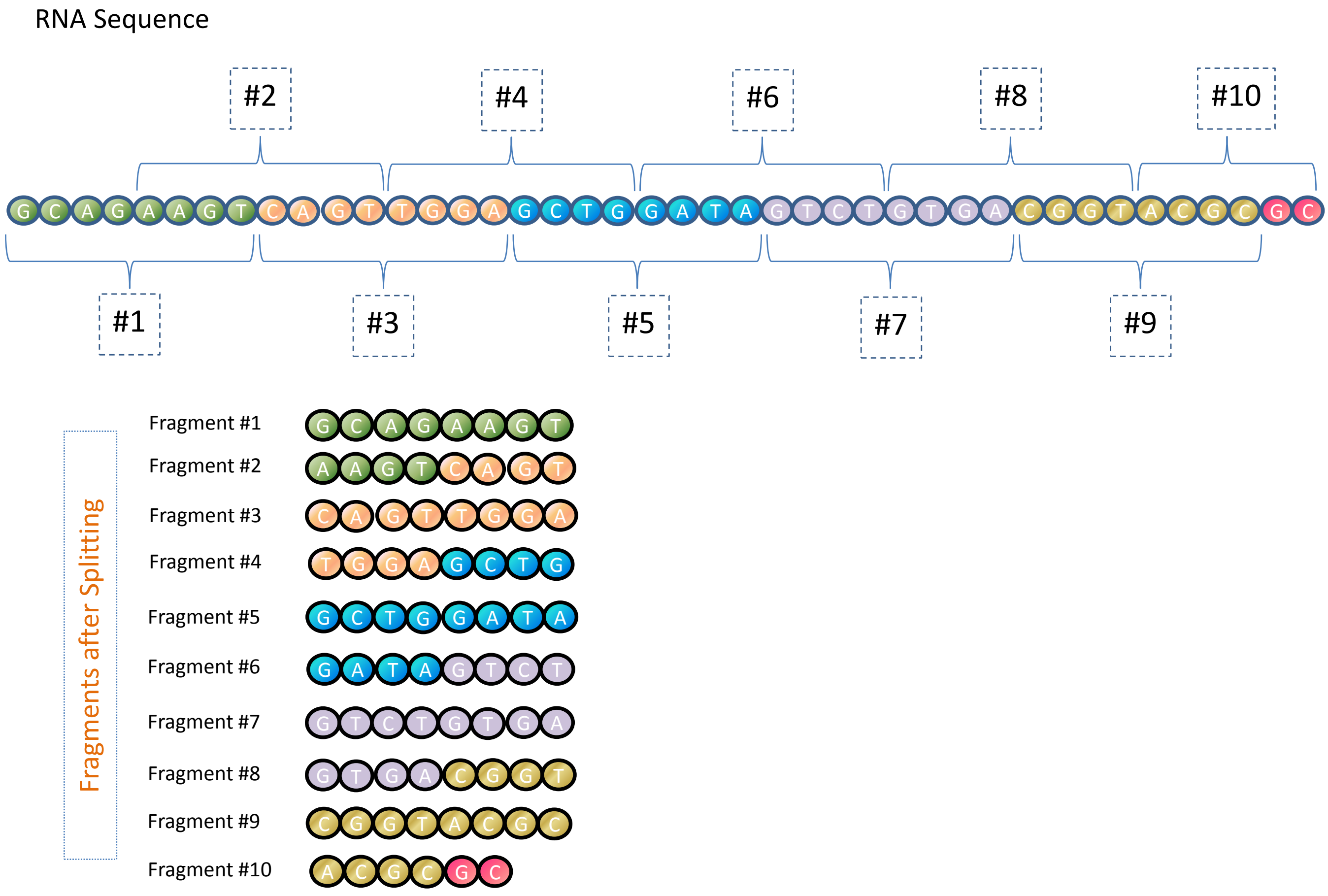


Figure 4.2.: *Illustration of splitting process of a sequence.*

4.1.2. Step 2 : Finding the k-nearest neighbors

In this phase the fragments/chunks are represeted as a graph. By exploting the properties of graphs, an efficient graph kernel technique is used to find and retreive the k closest instances of each fragment that have the highest similarities. Getting the closest instances will form the basis to establish a single large graph that represents the relation across all different fragments of all genomes.

k-nearest neighbors

Using an efficeient graph kernel technique such that Neighborhood Subgraph Pairwise Distance Kernel **NSPDK** (F. Costa 2010), all fragments/chunks are transformed into representitive graphs. Then the similarity between those representitive graphs is computed, and the k closest neighbor instances of each fragment with the highest similarities are retreived. For retrieving the nearest neighbors, a small sufficient number of different structures for each sequence is sampled. Then each structure is encoded as a labled graph preserving all the information about this structure in terms of it's nucleotides and bonds types. The similarity between the representative graphs is computed using an efficient graph kernel (**NSPDK**). To avoid the quadratic number of comparisons, a vector representation for each graph is extracted, then on those vector representations, an inverse index is constructed via a hashing technique. This will allow to retreive the nearest neighbors in a constant time for each entry. Finally, a covariance model is introduced, which is used to scan the dataset to look for the missing entries, try to recognize them in order to enhance the output results of retrieving the nearest neighbors[2].

4.1.3. Step 3 : Construct a single connected graph

Once the k-nearest neighbors of each fragments are retreived, a undirected weighted graph $G = (V, E)$ that represent the relationship between fragments of all genomes is constructed. Each fragmant is represented by a vertex v. A vertex v_i is connected to vertex v_j if v_j is among the k nearest neighbors of v_i, the edge e_{ij} is weighted by the similarity s_{ij} between v_i and v_j. if the similarity $s_{ij} \leq \tau$, this means that the vertices v_i and v_j are not connected. As G is undirected this required $w_{ij} = w_{ji}$.

Removing the weak connections

Usually, retrieving the k-nearest neighbors may include some weak relations. Those relations can disturb the results of the graph layout in the next step. Powering the similarity to some threshold p will filter all the weak similarites, and only the strong similiaries will survive. That is:

$$\bar{S_{ij}} = S_{ij}^p \tag{4.1}$$

Where S_{ij} is a non-negative similarity measure between x_i and x_j, p is a scale parameter that must, somehow, be chosen. Powering similarity S_{ij} to p will drop down the weak similarities into 0. p should be enough to remove and filter all those similarities that are below some threshod(∼50-60%). p must be choose carefully, the higher p may remove some significant values, also lower p can leave some weak similarities.

Here should include a figure shows this process(Removing weak similarities).

Preserving the original sequential structure

Some edges/connections must be added to connect each vertex with the preceding and following vertices to preserve the original sequential structure of each individual genome (**Fig. 4.3**). Each vertex v_i is connected to k preceding and following vertices. Those connections have different weights, depending how far is the vertex from v_i, the nearer ones have the higher weight has and so on. This step can be managed as follow:

For $i - k \leq j \leq i + k$, and $i \neq j$

$$w_{ij} = \frac{1}{(|i-j|)(win\_size)(\alpha)} \tag{4.2}$$

where k is the number of preceding and following vertices. k should be enough to preserve the original sequential structure of genomes(say 3). $|i-j|$ is positive and indicates how far v_j from v_i. win_size is window size . α is a control parameter, it controls the given weight.

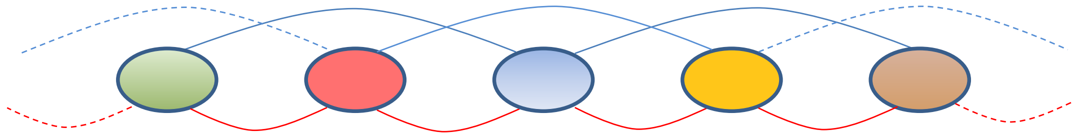


Figure 4.3.: *Illustration of the added connections between the adjacent fragments that preserve the sequential structure of the sequence.*

Storing Graph Sparsely

Finally, the graph G is represented as $n \times n$ matrix. Since this matrix will be very large, symmertic with a large number of zero elements, working with such very large matrix is time and memory consuming then storing the matrix sparsely is the best way to store such matrices. Reductions can be realized by storing only the upper triangular non-zero entries of the matrix. Storing the matrix sparsely will save time of manipulation and memory of storing. Since each element is connected only to k-nearest neighbors, the memory consumption to store the matrix will drop from $\mathrm{O}(n^2)$ to $\mathrm{O}(n)$ (see Ch3).

4.1.4. Step 4 : Dimensionality Reduction and 1-D layout

Once the sparse similarity matrix that represents the connected graph with the whole relations between the fragments has been constructed, the spectral graph technique that makes use of the spectrum (eigenvalues) of the similary matrix can be used to perform dimensionality reduction. The fragments is represented in 1-D layout by getting the second smallest eigenvector. Whenever the intrinsic dimensionality of a data set is smaller than the actual one, dimensionality reduction can bring an improved understanding of the data apart from a computational advantage and memory consumption. This phase can be summarized as following:
(1) Construct the normalized *laplacian matrix*,
(2) Then, compute the second smallest eigenvector.

Next sections will present each step in further details.

Construct Laplacian Matrix

In case of spectral layout for the fragmants by computing the non-zero smallest eigenvector, the *laplician matrix* has to be constructed. *Laplacian matrix* is an $n \times n$ square-matrix representation of a graph. Once the similarity matrix has been constructed, then the *(normalized) laplician matrix* can simply be constructed.

- *Normalized weighted laplacian.* matrix can simply be constructed by using the following formula :

$$l_{ij} = \begin{cases} 1 & \text{if } i = j \text{ and } deg_i \neq 0 \\ -\frac{w_{ij}}{\sqrt{deg(v_i)deg(v_j)}} & \text{if } i \neq j \text{ and } v_i \text{ adjacent to } v_j \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

Where w_{ij} is the weight or similarity between x_i and x_j. $deg(v_i)$ is degree of x_i, and is defined as:

$$deg(v_i) = \sum_{j=1}^{n} w_{ij}$$

which is only the sum over all vertices adjacent and have connection to v_i. The degree matrix D is defined as the diagonal matrix contains the degree of vertices $v_1 \dots v_n$ on the main diagonal.

Compute the Second smallest Eigenvector (1-D)

Once the *laplacian matrix* has been constructed, the dimensionality reduction and 1-D representation of the fragments can be done by applying the eigen algorithm. This representation can simplify the problem as one dimensional problem.
In this step, the eigenvalues of the Laplacian matrix and the corresponding eigenvectors are computed. Retreiving the second smallest eigenvector and give the 1-D layout of the graph. In addition, this representation gives somehow the distance between the fragments, the more closer the more similar.

Computing the second smallest eigenvector

An eigenvector of a square matrix A is a non-zero vector v that when the matrix is multiplied by v, yields a constant multiple of v. The multiplier is commonly denoted by λ.That is:

$$Av = \lambda v \tag{4.4}$$

The number λ is called the eigenvalue of A corresponding to v.

Once the exact value of an eigenvalue is known, the corresponding eigenvector can be computed by finding non-zero solutions of the eigenvalue equation, that becomes a system of linear equations with known coefficients.

The standard algorithms compute all the exact eigenvalues and the corresponding eigenvectors. Unfortunately, those algorithms are computationally expensive and high memory consuming. For these reasons, those methods are not appropraite to problem in this work, because the second smallest eigenpair is only needed. Looking for an effecient algorithm is an essential task. *Arnoldi Algorithm* is one of the effeciaent algorithms that solve large scale eigenvalue problems. *Arnoldi Algorithm* is designed to compute a few approximate eigen pairs such as those of the largest or the smallest eigen pairs using $n \cdot \mathcal{O}(k) + \mathcal{O}(k^2)$ storage.

Remark. In R, *ARPACK* which stands for ARnoldi PACKage, can be used to solve this problem. It is a software based upon an algorithmic variant of the Arnoldi process called the Implicitly Restarted Arnoldi Method (IRAM). The key insight is that *ARPACK* solves the function passed to it. The adjacency matrix gets passed to *ARPACK* via this function, so there is some dependence on scoping (arpack has an option to specify the environment, so encapsulation is possible if desired). For further details about this package see[3].

After getting the second smallest eigenvector, the fragments will be represnted in 1-D layout. This representation will reduce the dimensionality of the fragments, and simplify the problem as one dimensional problem. Typically, solving such problems (1-D problems) can be computationally efficient.

Figure. Here will be a figure shows the 1-D representation of the fragments

4.1.5. Step 5 : Filtering and Smoothing Data

The 1-D representation results from the previous step can contain some noises and inconsistent data. Filtering and smoothing techniques can be used to overcome this problem. Smoothing is a technique to remove inconsistent data and attempts to capture important patterns in the data. There are many techniques in this field such as *exponential*, *average*, and *median* smoothing. *Median smoothing* algorithm is used in the proposed method, because it is robust and more suitable to problem in this work.

Edge preservation properties

Median Smoothing is one kind of the smoothing techniques. All smoothing techniques are effective in removing noise in smooth patches or smooth regions of a signal, but adversely affect edges. Often though, at the same time when reducing the noise in a signal, it is important to preserve the edges. Edges(big jumps in our case) are of critical importance to capture and identify the position where the mutation events happened. **Fig. 4.6** shows smoothing after 5 and 20 iterations can show how those edges (big jumps) are preserved, and the other regions are smoothed.

Using smoothing in the model

In this phase, two stages of smoohing can be distinguished:

1. Smoothing and filtering 1-D representation of the fragments.
2. Smoothing the fragment derivatives.

(1) Smoothing and filtering 1-D representation:
With resonable window size k, the *median filtering* algorithm is applied for the second smallest eigenvector(1-D representation) that results from the previous phase. This procedure will remove the noise that can arise, and enhance the quality of the eigenvector result(**Fig. 4.4**).

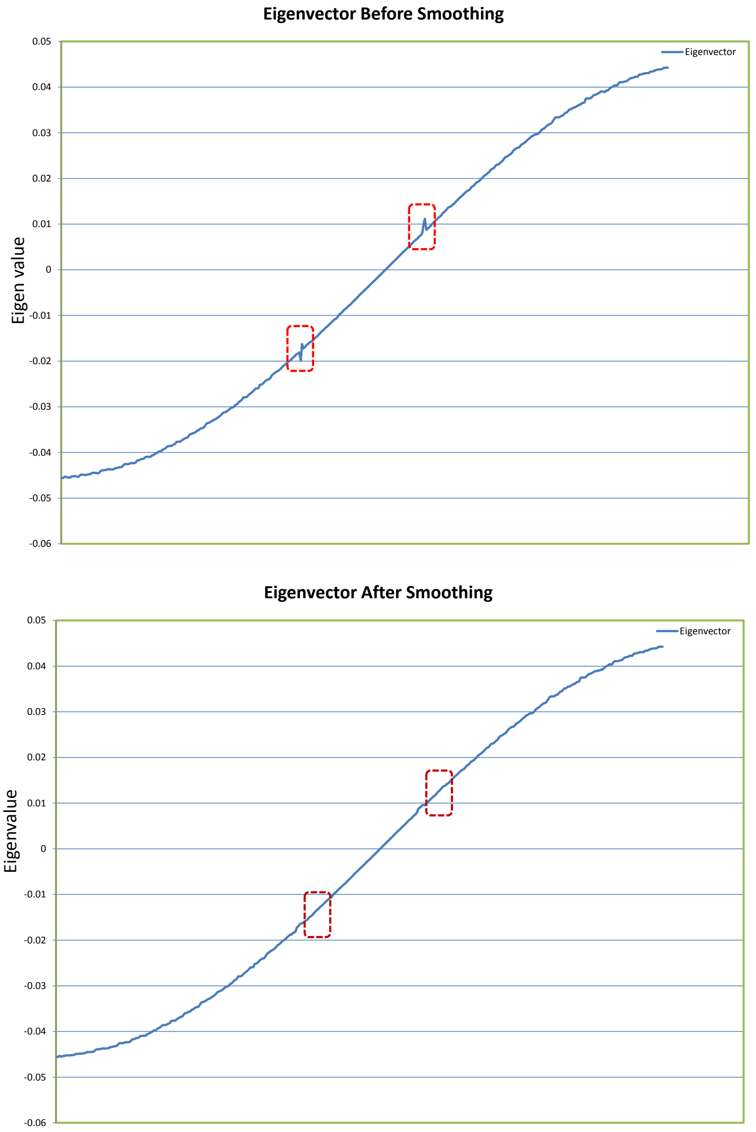


Figure 4.4.: *Filtering the noises that can arise after getting the second smallest eigen-vectors. (A) Data set of eigenvector with some noises (B) Data set of eigenvector after filtering and solving the noises using the median filtering algorithm.*

(2) Smoothing the derivatives:

In this stage, the data set of the eigenvector is transformed into signal that can easily show the significant regions in the data set. Iteratively smooth the signal (derivatives) with excepting the significant regions in the signal, this exception can be done by selecting the top percentage of the modulus of the derivatives(**Fig. 4.6**).

Description of the function :

(a) Compute the derivative of the adjacent entries in the data set(1-D representation of fragments).

Such that, for $i = 1 \ldots n$, where n is the number of fragments, compute:

$$\Delta_i = \begin{cases} e_i & \text{if } i = 0 \text{ or } i = n \\ e_i - e_{i-1} & otherwise \end{cases}$$

(b) Sort the modulus of the derivatives and get the top p percentage which represents the significant regions in the signal.

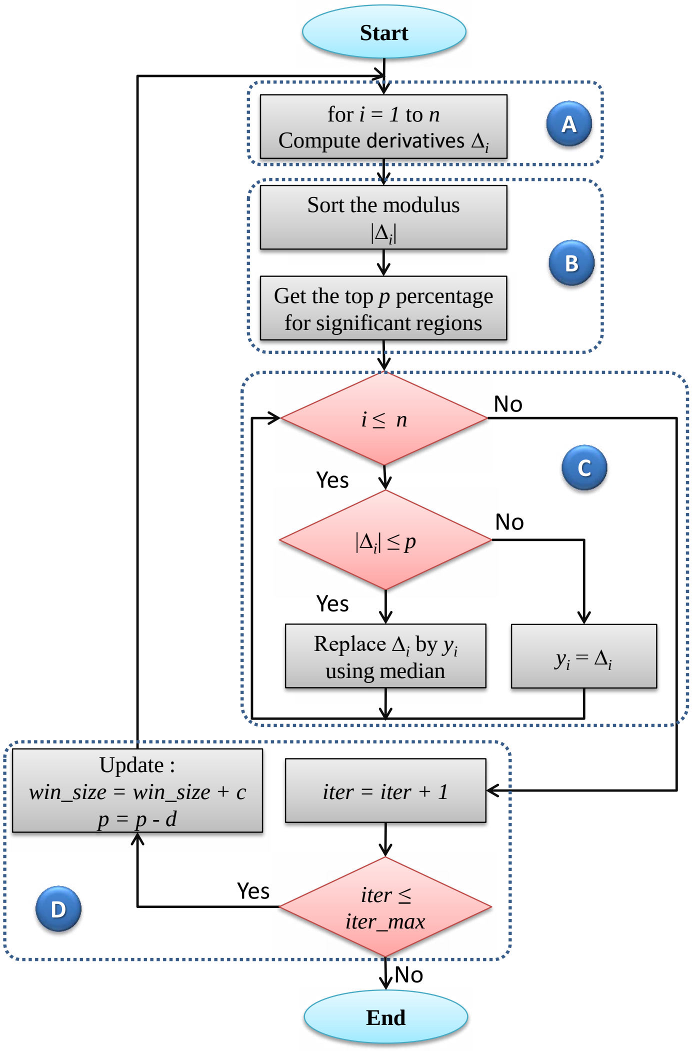


Figure 4.5.: *Flowchart shows the smoothing process. (A) Compute the derivatives. (B) Get the top percentage kept away from smoothing. (C) Apply the median smoothing algorithm. (B) Iteratively repeat smoothing process with upadte parameters.*

(c) Smooth the entries :

For those entries that have modulus of the derivatives less than p do :

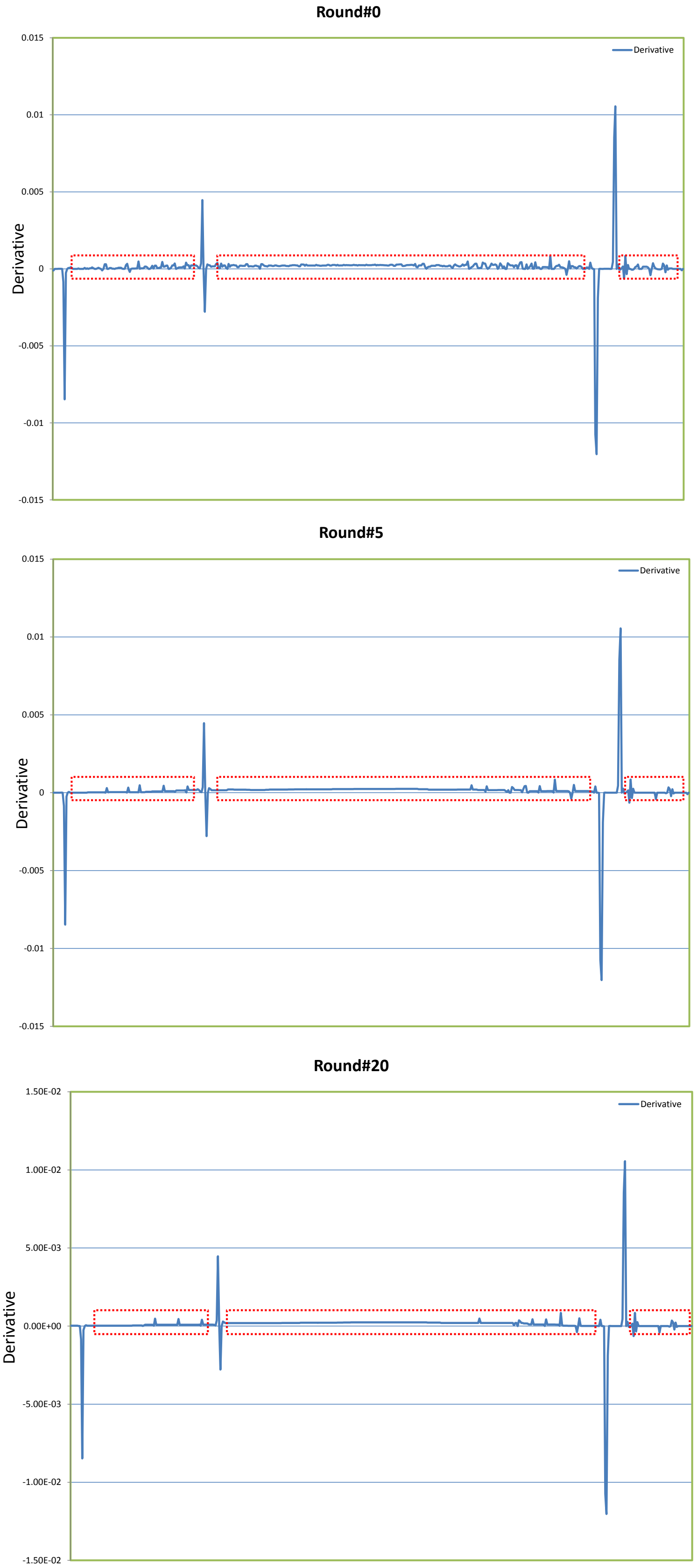
- replace the current entry Δ_i by y_i such that :

$$y_i = median\ [\Delta_{i-k}, \Delta_{i-k+1}, \ldots, \Delta_{i+k-1}, \Delta_{i+k}]$$

The k entries in the terminals(left most and right most) are considered a bit special in order to obtain an equal window size. For those entries, the first(last) entry can be repeated to obtain enough entries to fill the window.

(d) Iteratively smooth the signal, by repeating steps **(a-c)** with increasing the smoothing window size parameter, and decreasing the the percentage p of the significant entries(that must be preserved).

Figure 4.5 shows the flowchart of this stage.


Round#0

Round#5

Round#20

Figure 4.6.: *Iterative smoothing. (A) Before Smoothing. (B) After the 5th iteration. (C) After the 20th iteration. In addition this figure shows the Edge preservation property.*

4.1.6. Step 6 : Encoding and Machine Learning(ML) model

The following step shows how the signal can be encoded and compressed into some meaningful string. This compression can save time of manipulation and storing of memory. Once the codes have been created, the data set can be constructed out of those codes. Finally, a machine-learning (ML) model is built that can make accurate prediction about new, unseen mutation event.

Encoding

Encoding is the process by which a piece of information is converted into symbols (typically individual letters and numbers). In this phase, codes are considered, which encode each piece of information (interval of fragments that have common features) into a code word from some dictionary. Concatenation of such code words give us an encoded string of the whole signal which represents a genome sequence. This phase can be assumed as data compression, where signals can be compressed into a few codes by exploiting the features for the adjacent intervals(fragments). Since the signals that are the 1-D representation of genomes that already got from the previous "filtering and smoothing" phase, the block of fragments that consist of adjacent fragments with the similar features can be encoded into a single code.

Function description

The encoding process can be divided into three stages. In the first stage the sign of the input data (interval) is assigned(0,+ve,-ve), the input data is discretized vertically in the second step. In the last stage, a block of input data that have the same features is assigned into one horizontal segment block. To show this process, we assumed some meaningful symbols, such that each *block* of data can be represented by three parts (**Figure 4.7**), each part has a different meaning.

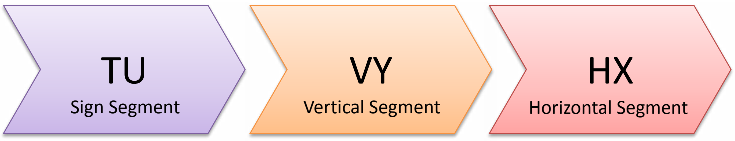


Figure 4.7.: *Illustration of the different parts of encoding string.*

A prefix code of each part (T, V, H) is fixed, which means it always in this format, where "**T**" for Sign, "**V**" indicates the vertical level, and "**H**" indicates the Horizontal level.

The following describes how to encode each part in details.

(1) Part of Sign

The First part "TU" is the sign part of the code. "**U**" indicates the Sign of the data block(i.e. where the block lies in +ve, -ve or zero). "**U**" can be encoded as follow:

$$U = \begin{cases} Z & \textit{if the block lies in range Zero}(R_0) \\ P & \textit{if the block is +ve and doesn't lie in range Zero} \\ N & \textit{if the block is -ve and doesn't lie in range Zero} \end{cases} \tag{4.5}$$

where R_0 can be specified as below (Eq 4.6).

(2) The Vertical Part

The second part "**VY**", "**V**" indicates the vertical part, "**Y**" is the vertical segment block. "**Y**" can be defined as follow:

(a) Define the range "R_i" of each segment block according to this formula:

$$R_i = \begin{cases} \frac{max}{2} & if\; i = N-1 \\ \frac{R_{i+1}}{2} & if\; i = (N-2)\dots 1 \\ max - \sum_{j=1}^{N-1} R_j & if\; i = 0 \end{cases} \tag{4.6}$$

where N is the maximum number of vertical segment blocks, R_i is the range of segment block i , max is the maximum value of inputs. Since the inputs are concentrated when going towoard the zero, for this reason the range of each segment block is half the above segment to make some type of balance and get good discrimination. The other option for R is the equal sizes.

(b) $Y = i$, i is the range where the data block lies.

(3) The Horizontal Part

The third part "**HX**", where "**H**" is the horizontal, "**X**" is the horizontal segment block. X is defined as follow:

$$X = \begin{cases} \frac{f}{m} & \textit{if } f \leq \varphi \\ L & \textit{if } f > \varphi \end{cases} \tag{4.7}$$

where m is the maximum number of fragments per segment block. φ is the maximum number of fragments per single event.
It is obvious that this step can compress large signals significantly into few blocks of codes. This compression technique should represent a signal as accurately as possible using the fewest number of bits.

(Figure. here will be a figure that shows the discretized.)

Data Set

DataSet is a collection of *instances*. Commonly a dataset corresponds to the contents of a single data table, or a single matrix. Every column of the table represents a particular *attribute* or *variable*, and each row corresponds to one *instance*. The collection of instances form the *DataSet*, and the block of code creates an *instance* with specific number of *attributes*.
To create the data set, that consists of a collection of *instances*, the number of attributes that the *instance* can have has to be determined. The example below creates a data set of *instances* each one has 15 attributes. This example can show how to create the data set using the codes that are already created in the previous step "Data Encoding".

How to create the *DataSet* :
To create Data set, two parts are necessary, the instance and the target of this instance which represent the mutation event.

1- Create the *instances*
With window size (say 5), go through each entry/code and joint it with immediately preceding and following entries/codes. Usually the window size is an odd number. Since the mutation events almost consist of at least three regions, statring, ending of the event and the region in between, so the reasonable size between 5-9 .

2- Create the *target*

The *target* of this *instance* that represents also the class or the mutation event is simply the majority of the events of the codes that forms this *instance.*

Example :
Given the following codes and the corresponding targets/events.

TP V1 H0	0
TN V1 H0	0
TZ V0 HL	0
TN V2 H0	Insertion
TZ V0 H4	Insertion
TP V1 H0	Insertion
TZ V0 HL	0
TN V1 H0	0
TZ V0 HL	0

The data set that will form out of this codes when the window size is 5 is:

TP V1 H0 TN V1 H0 TZ V0 HL TN V2 H0 TZ V0 H4	0
TN V1 H0 TZ V0 HL TN V2 H0 TZ V0 H4 TP V1 H0	Insertion
TZ V0 HL TN V2 H0 TZ V0 H4 TP V1 H0 TZ V0 HL	Insertion
TN V2 H0 TZ V0 H4 TP V1 H0 TZ V0 HL TN V1 H0	Insertion
TZ V0 H4 TP V1 H0 TZ V0 HL TN V1 H0 TZ V0 HL	0

Remark. ”0” assumes that no mutation event happened in this area.

Building Machine learning model

In the machine learning (ML), the main objective is to build a model that can gain an experience from learning based on the training data set. This model has ability of a learning machine to make accurate prediction on new, unseen events, i.e. can predict the class of unlabeled instances/examples based on its experience.
The last step in the proposed method is to build a machine learning (ML) model. This ML model can accurately detect the mutation event (that could happen in the genomes) of a new unknown instances after having experience from learning based on the training data set.

CHAPTER 5

Results and Evaluation

The following chapter presents the data set that is used as an input in the proposed method, and how this data set is built. The performance measurements and evaluation of the proposed model will be reported in the next section. Finally, the benchmarking process, the cross check of some parameters and how the model can be influenced by the change of those parameters will be part of the last section of this chapter.

5.1. Data Set

An artificial data set is used as an input to the model. This type of data set is used because there is full control on the mutation events that can happen in the sequences. In addition, it is very easy to manage and measure the performance of the proposed model when using such a data set. The genome sequences are artificially generated using the nucleotide subunits (adenine, guanine, cytosine and thymine, referred to by the letters A, G, C and T) that represent the DNA nucleotides. A single genome sequenced can comprise millions to billions of nucleotides. Whenever the Ancestral genome (the root genome in the generated tree) has been generated, a phylogenetic tree is generated based on the Ancestral genome.

5.1.1. Generating the artificial phylogenetic tree

The artificial phylogenetic tree is generated by defining different specifications:

(1) The parameters that must be specified initially:

- The maximum number of generations.
- The maximum number of genome sequences per generation.
- The maximum number of mutation events that can happen per sequence.
- The maximum and minimum percentage of occurrences of mutation event.

(2) The mutation events that are considered for building the phylogenetic tree are(**Fig. 5.1**):

- Single nucleotide mutation.
- Insertion.
- Deletion.
- Duplication.
- Inversion.

Remark. more about this events can be found in chapter 3.

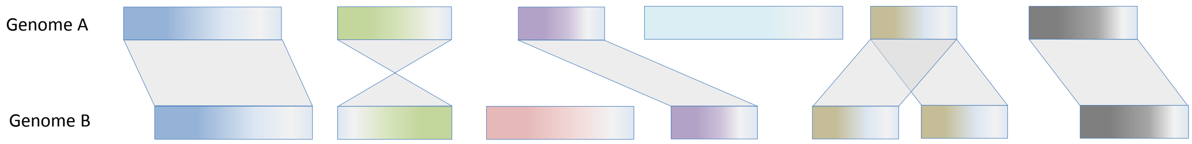


Figure 5.1.: *Illustration of the different mutation events.*

Any mutation event can happen randomly at any position in the sequences of the next generation. Which type of events is it and where it happened are saved in a *history table*. The *history table* contains the lenght of segments in terms of the number of nucleotides of each event, and the position of where this event happened within the sequence. The mutation events are generated in a specific percentage. After generating the whole tree, all the terminal nodes/leafs of this phylogenetic tree are collected(**Fig. 5.2**). This collection forms the related sequences that can be considered as the input data set and can feed into the proposed method to test and evaluate.

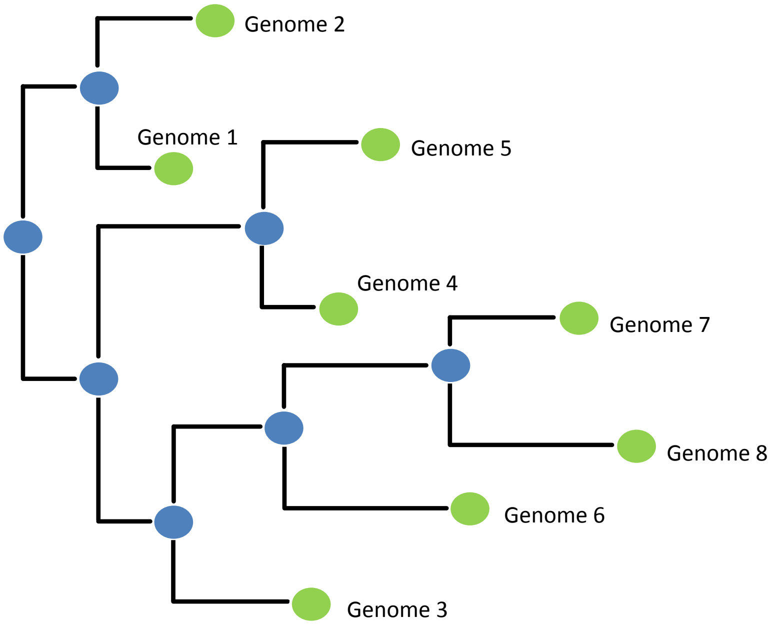


Figure 5.2.: *Illustration of the structure of artificial phylogenetic tree.*

5.2. Results

All the experiments and tests have been done on this *platform* :

OS : Fedora release 18 (Spherical Cow) 64-bits.
CPU : Intel(R) Core(TM) i3-2100 CPU @ 3.10GHz (64-bits).
Memory : 8GB RAM.
Compiler : C++11.
Explicit Decomposition with Neighborhoods(**EDeN**) vesion 1.2.0.
R : Version 2.15.2, with Arnoldi Package (Arpack v.1.0).

5.2.1. Performance metrics

The Common performance metrics for classification problems:

True Positive (TP). A positive instance is predicted as a positive (correctly identified).
True Negative (TN). A negative instance is predicted as a negative (correctly identified).
False Positive (FP). A negative instance is predicted as a positive(incorrectly identified).
False Negative (FN). A positive instance is predicted as a negative(incorrectly identified).

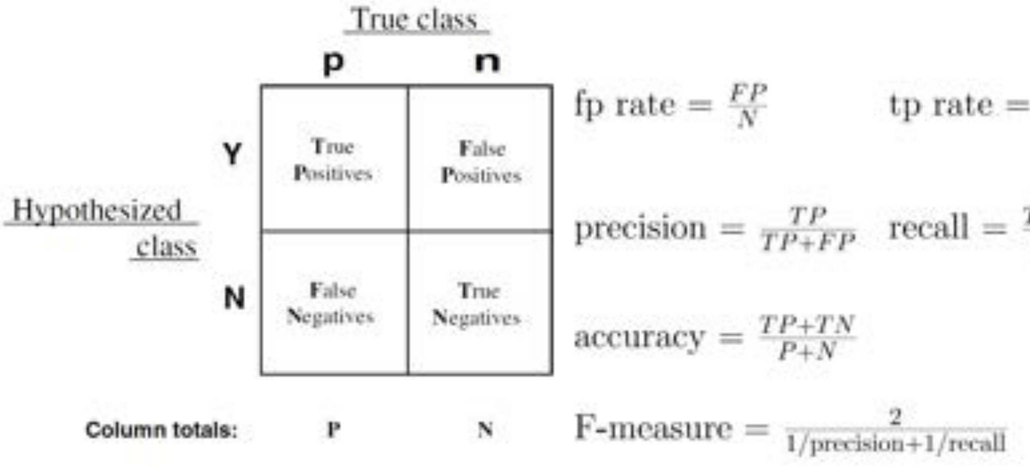


Figure 5.3.: *A confusion matrix and common performance measurements[64].*

Accuracy(ACC). The degree of closeness of measurements or predictions of a quantity to that quantity's actual (true) value.

$$ACC = \frac{TP + TN}{P + N} \tag{5.1}$$

Precision(PRE). The fraction of retrieved instances that are relevant.

$$PRE = \frac{TP}{TP + FP} \tag{5.2}$$

Recall(REC). The fraction of relevant instances that are retrieved.

$$REC = \frac{TP}{P} \tag{5.3}$$

F-measure (PRF). A measure of a test's accuracy. It considers both the precision PRE and the recall REC of the test to compute the score.

$$PRF = \frac{2}{1/PRE + 1/REC} \tag{5.4}$$

Average mean precision (APR). Area under the precision recall curve (APR): Given that the precision is plotted as a function of the recall.

Area under the ROC curve (aka AUC) or ROC. The ROC curve is generated by plotting the fraction of true positives out of the positives against the fraction of false positives out of the negatives at different thresholds.

5.2.2. Performance measurement

This subsection shows the performance of the proposed method. Mainly, in measuring the performance, two phases can be distinguished: (1) Event-occurrence detection. (2) Event type discrimination.

Event-occurrence detection

Event-occurrence means that there is a mutation event happened in a specific region of the sequence. In this phase of measuring, the performance of how much accurate the model can detect an event in a certain region is measured. The proposed model can accurately detect up to 95% of the occurrence mutation events. The undetected events can be due to some reasons:

1. The overlap between more than one event. Those overlapped events can be assumed as a single event.

2. The smoothing process. During the smoothing process some significant data that represents the boundary of an event can be lost. Consequently, that event can't be detected.

The event-occurrence detection is influenced by the number of the vertical segment blocks. Increasing the number of the vertical segment blocks can decrease the influence of the smoothing, and can preserve the significant edges (big jumps). **Fig. 5.4** shows this influence. In addition, the event-occurrence detection can be influenced by the number of sequences and how much distance those sequences are from each other.

Remark. More details about the performance of the model and the measurements of detecting a mutation event for different parameter values can be found in Appendix-A.

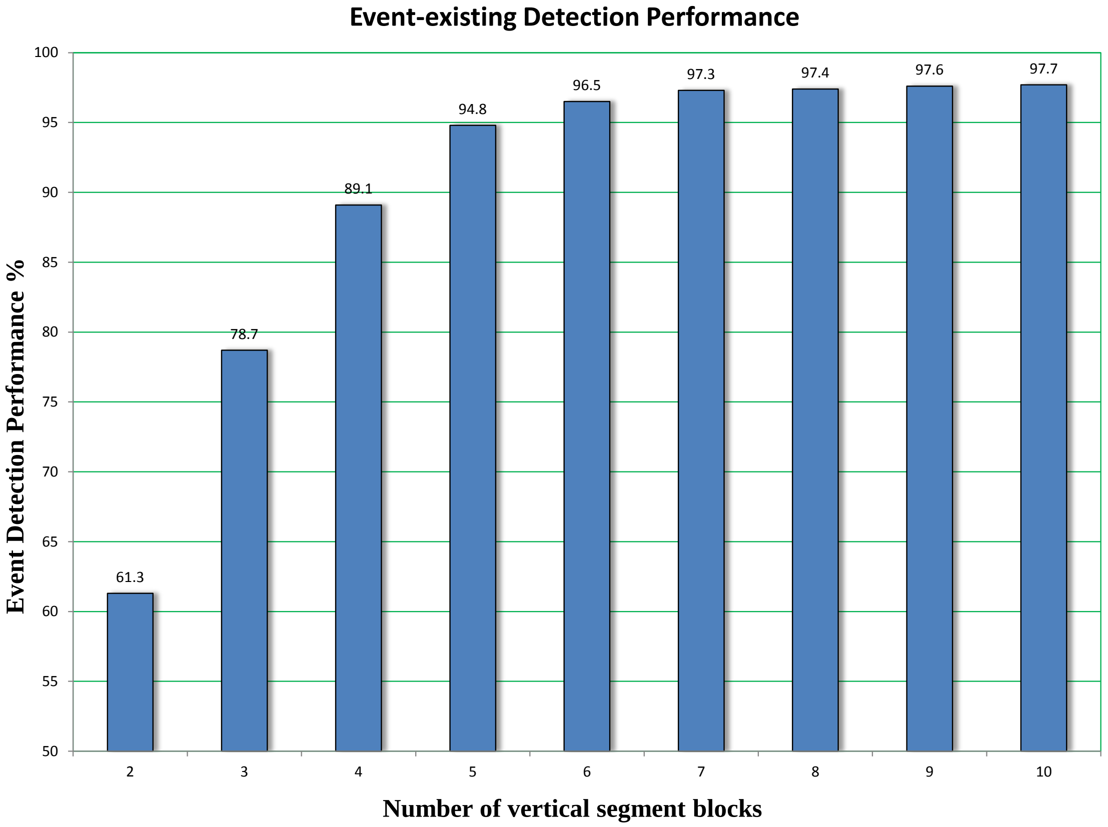


Figure 5.4.: *Event-occurrence detection performance vs. number of the vertical segment blocks.*

Event type discrimination

Event type discrimination measuring means that the model has detected that an event happened in a certain region, but it needs to discriminate which type of mutation event it is. In general, the model can accurately discriminate between the different types of mutation events with one exception.

The following figures show the performance of discriminate types of the mutation events under different performance metrics.

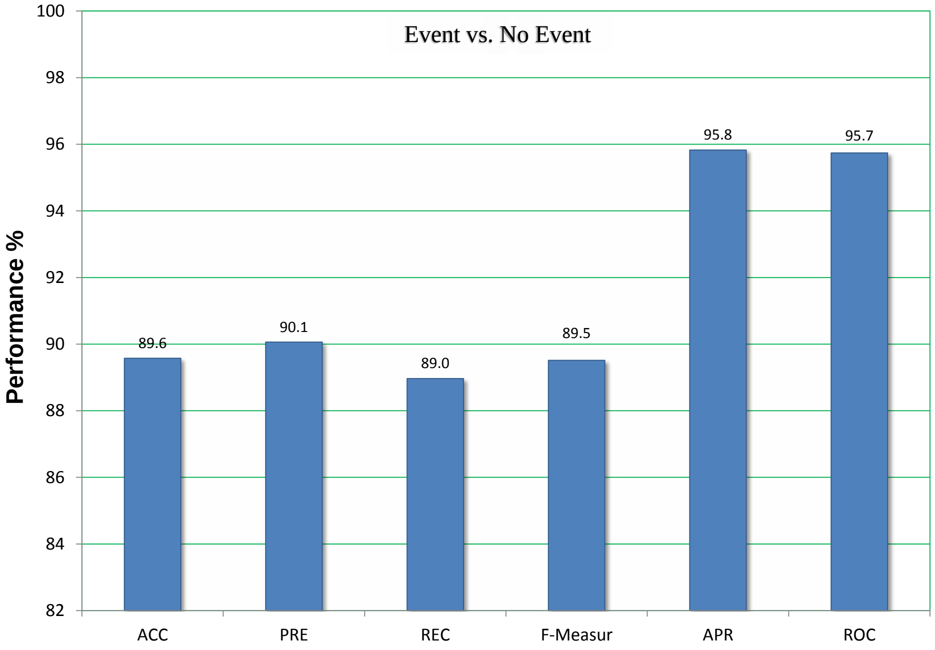


Figure 5.5.: *Output of different performance metrics for measuring the result quality of Event vs. No Event discrimination.*

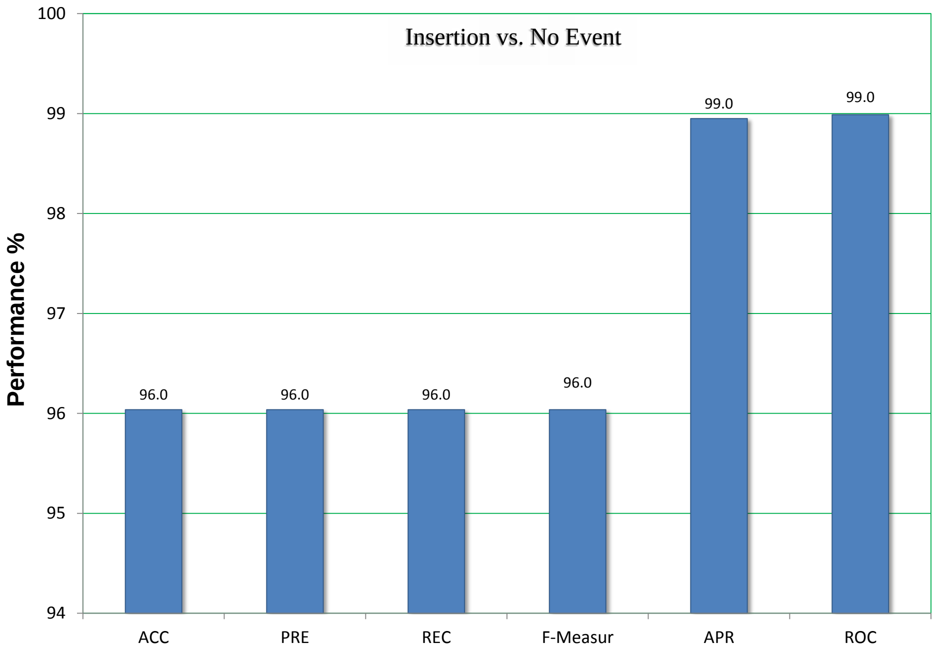


Figure 5.6.: *Output of different performance metrics for measuring the result quality of Insertion vs. No Event discrimination.*

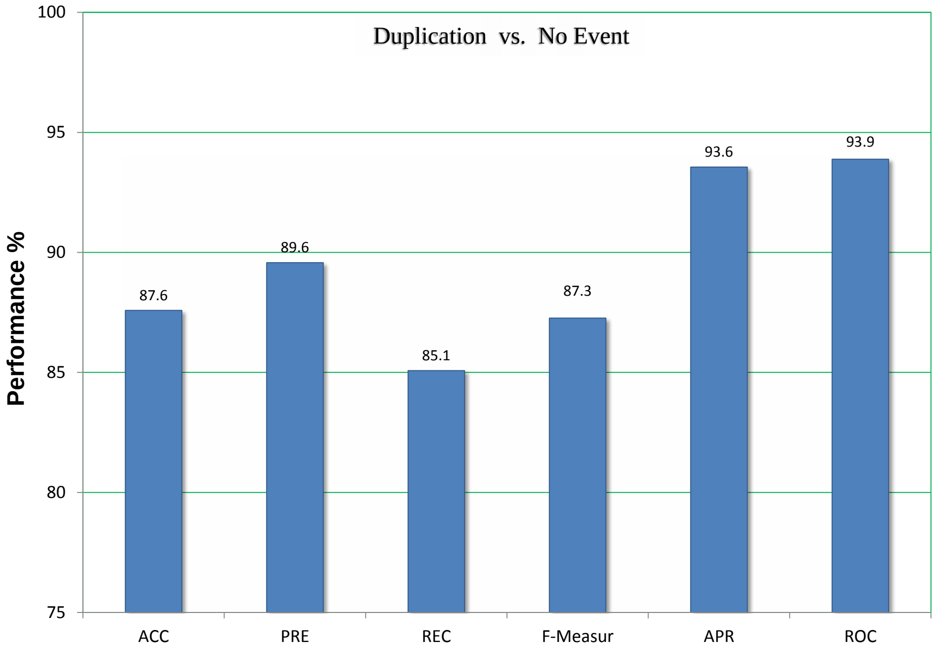


Figure 5.7.: *Output of different performance metrics for measuring the result quality of Duplication vs. No Event discrimination.*

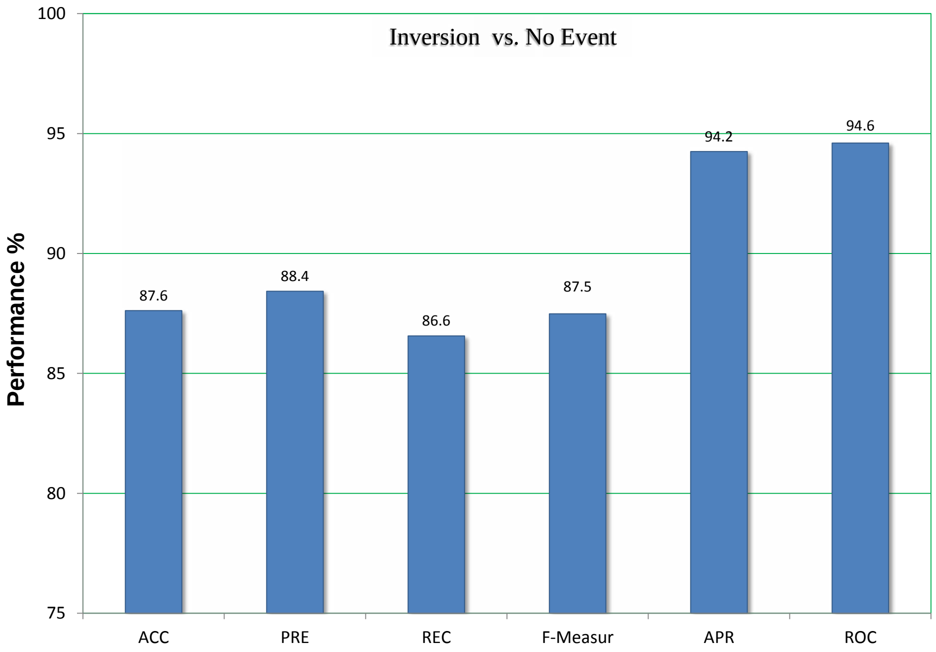


Figure 5.8.: *Output of different performance metrics for measuring the result quality of Inversion vs. No Event discrimination.*

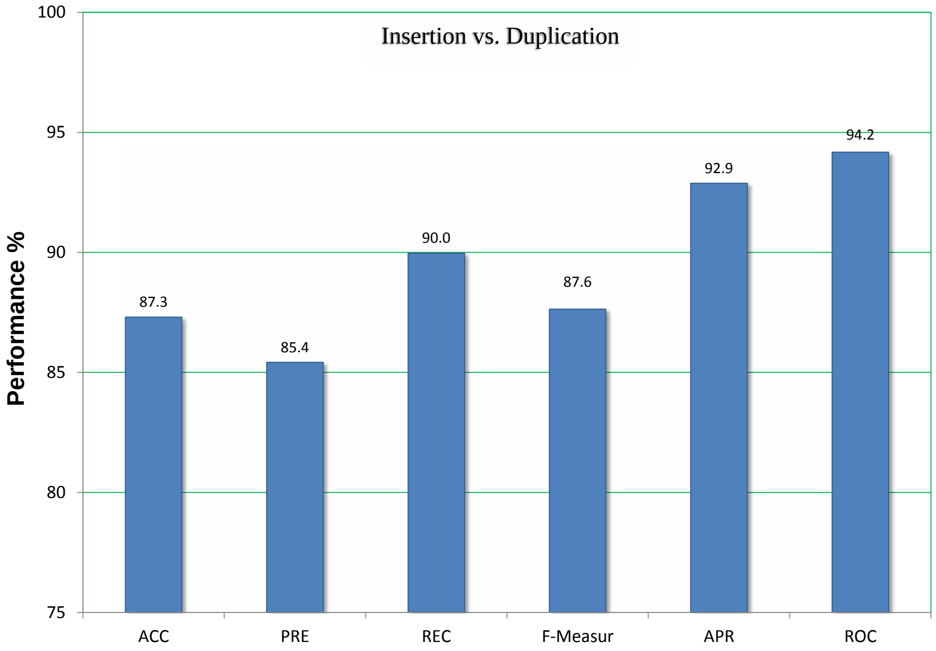


Figure 5.9.: *Output of different performance metrics for measuring the result quality of Insertion vs. Duplication discrimination.*

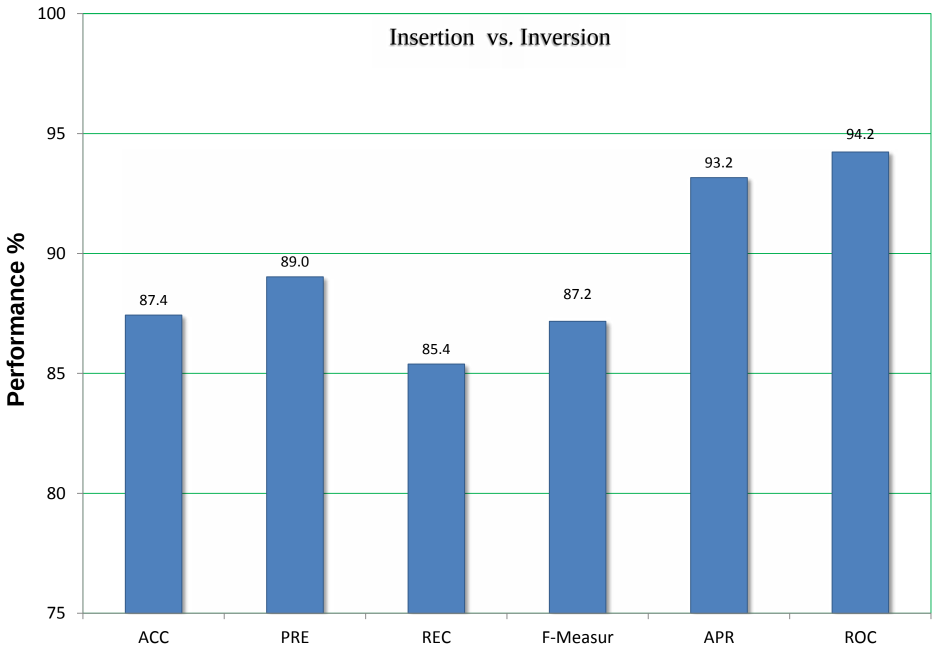


Figure 5.10.: *Output of different performance metrics for measuring the result quality of Insertion vs. Inversion discrimination.*

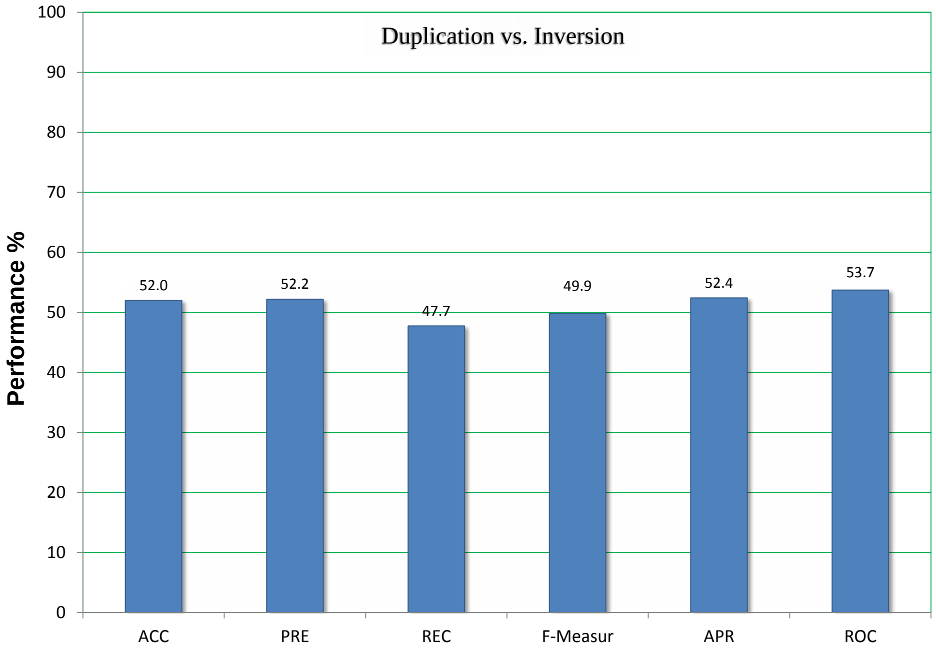


Figure 5.11.: *Output of different performance metrics for measuring the result quality of Duplication vs. Inversion discrimination.*

All of the above performance measurements which represent the best performance are found when value of parameters are as follow:

- The number the vertical segment blocks: the best value of the this parameter is 5.
- The size of the vertical segment blocks: the best performace when the segment blocks are equal.
- The size of the instances of the dataset: the best value of the this parameter is 4.

Some remarks on this phase of performance:

1. The model can discriminate very accuratly the occurrence of an event from no event occurrence, specially for *insertion* (Figures 5.5 - 5.8).
2. Discrimination between insertion mutation event and the other events is also very accurate up to 94% (Figures 5.9 - 5.10)
3. *Deletion* is not included, because it can assumed as *insertion.* Delete a segment from one sequence is somehow like insert a segment in another sequence. For this, the perormance of *deletion* is same as *insertion.*
4. The model can not discriminate between inversion and duplication. Fig. **5.11** shows the performance of this problem clearly. The reason of this problem is that those events (*duplication* and *inversion*) are assumed the same because those events have the same properties, only the difference between them is the direction, one is happened in the opposite direction of the other.

5.2.3. Complexity

When it comes to evaluate the complexity of this pipeline model, the most steps that influence the runtime and the memory usage of the model are: (1) finding the k-nearest neighbors. (2) computing the eigenvectors. The complexity of the model is linear for the runtime and the memory usage with growth of the size and the number of sequences.

Space Complexity

The memory consumed in the pipeline model depends on the memory consumed to find the k-nearest neighbors. The memory consumed for finding the k-nearest neighbors is linear, but it can be clear from **Fig. 5.12** that in some points the memory consumed is doubled. This doubling in the memory usage is related to the implementation of the EDeN (the tool that is used to find the nearest neighbors). The reason behind the doubling of the memory usage is that when certain vector is full, the size of this vector is doubled to be able to accomdate with the increasing of the data entries. Regarding of the memory usage for computing the eigenvector, it needs $n \cdot \mathcal{O}(k) + \mathcal{O}(k^2)$ memory storage[1]. The memory consumed for the other steps of the pipeline model is very small relative to the memory consumed for finding the nearest neighbors. Since the model is pipeline(i.e. work in sequential order) then the overall memory consumed is the same as the memory consumed for the step that has the highest memory consumption among all the steps of the pipeline model.

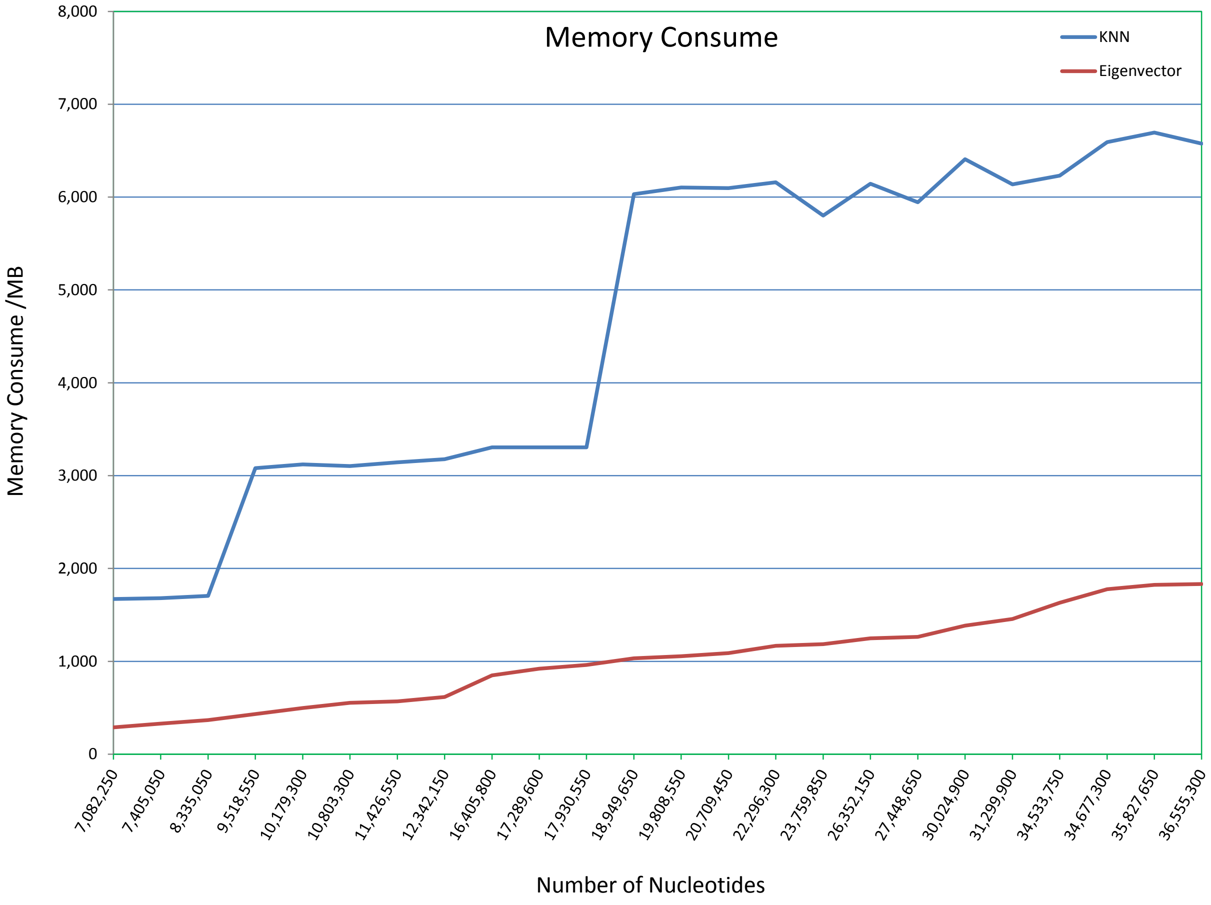


Figure 5.12.: *The memory consumption of (a) Nearest nieghbors (b) Eigenvector.*

[1] where k is the number of eigenvectors which are needed. For computing the second smallest eigenvector $k = 2$. n is the number of fragments.

Runtime

This subsection introduces a runtime of the proposed model that is collected from different experiments. The runtime of the model is depending on the runtime of two significant steps in the pipeline model.

1. The time of finding k-nearest neighbors. This time is linear when $k \ll n$, where n is the number of fragments and k is the number of the nearest neighbors of each entry.
2. The time of computing the second smallest eigenvector. This time is differ, since it depends on when the convergence happens. It can take from few up to tens of hundreds of iterations to reach the convergence. For this reason, the actual runtime can differ, but the complexity remains still linear $\mathcal{O}(n)$(**Fig. 5.13**).

The runtime of the other steps in the pipeline model is very small, it can not exceed of than couple of minutes. The runtime of the overall pipeline model is linear as it is shown in **Fig. 5.13**.

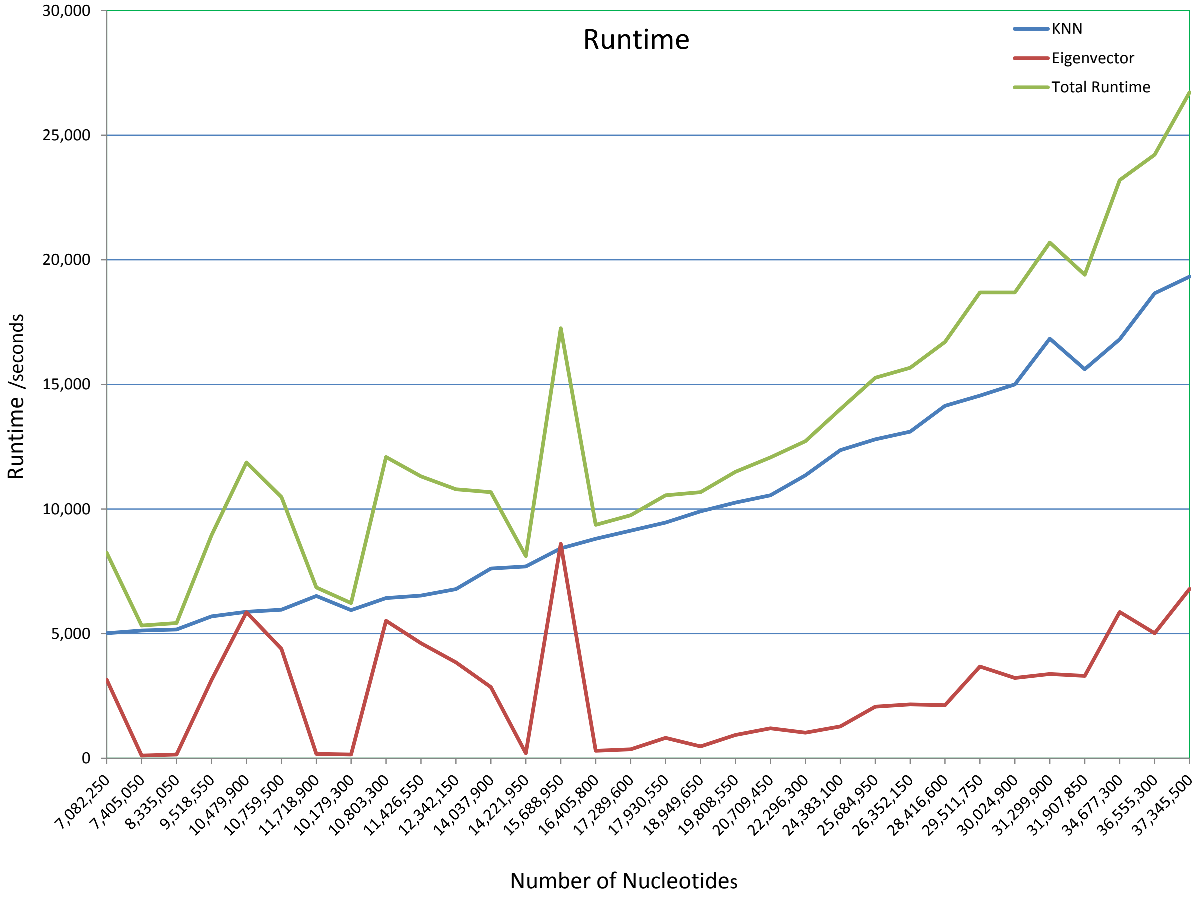


Figure 5.13.: *The runtime of (a) computing the k-nearest nieghbors (b) computing the eigenvector (c) the overall runtime of the model.*

5.3. Benchmarking

In this part of measurements, the performance of the model across some parameters that can make a real influence on the result quality of the model is compared.. The parameters that are used in the benchmark process are:

1. The number of the vertical segment blocks. The tested values for this parameter are between 2 and 10.
2. The size of the vertical segment blocks. The tested values for the parameter are (1) Equal segment blocks size (2) Half sizes.[2]

[2] Each segment block has half the size of the upper segment block.

3. The size of the instance (i.e. how many coding blocks that are in each data set instances) of the dataset that the ML model is used to learn. The tested values for this parameter are between 2 and 5.

The best performance of the model is found when the parameters values are as follow: (1) The number the vertical segment blocks (5-6). (2) The size of the vertical segment blocks, the best performace when the segment blocks are equal. (3) The size of the instances of the dataset(3-4).

Increasing the number of vertical segment blocks increases the performance of the event-occurrence detection, but very high value for this parameter can influence the performance of event-type prediction. So it should be compromised between the performance of those two phases.

Remark. More detailed results about the benchmark process can be found in Appendix-A.

CHAPTER 6

Discussion and Further Work

6.1. Discussion

The goal of this work is to come with a new tool that can solve large scale multiple genome alignment. This goal can be achieved in two stages. The first stage that the work of the thesis focus on- includes giving a pipelining model to detect and find the regions where the mutation events happened at the same time detects which type of mutations they are. This method is scaled for a large multiple genomes alignment, and considers both sequences and structure genomic. This proposed pipeline model solves this task efficiently in linear complexity in terms of runtime and memory usage. The information that this proposed model will offer, will be valuable and very useful for doing the second stage (multiple genome alignment) very easily. All the experiments have been conducted on an artificial data sets. In addition, benchmarking and cross checking process has been conducted for some parameters that have a strong impact on the performace of the model. The model has a high performance for detecting the mutation events with one exception[1]. The proposed model has a linear complexity.

[1] The exception is *duplication* vs. *inversion* discrimination. There is an idea to solve this problem. Unfortunately, due to the time limitation, this idea was not considered.

6.2. Future work

The main problem that is not solved in this model is how to discriminate between *duplication* and *inversion*. The reason for this problem is that both events have the same properties but happen in opposite directions. An idea for solving this problem was found, but unfortunately, due to the time limitation, this idea was not tested..

Great improvments can be achieved if it is possible in the future soon to use a parallel version of EDeN (the tool that is used in this work to find the k-nearest neighbors). Using such version will be great and will make a big jump in the runtime performance of the model. Since the runtime for finding the k-nearest neighbors forms the largest time consumption in this pipeline model.

Another improvement that can be possible is looking for another algorithm that can solve the eigen problem efficiently, or a new version of ARPACK[2]. Since the current version of ARPACK has a problem because it can take thousand of iterations to reach the convergence, fortunately, this problem rarely happens.

Unfortunately, not all the parameters benchmarking and the cross checking have been conducted due to the time limitation of the thesis. One can do a benchmarking and cross checking for all parameters that can make a real impact on the performance of the model.

The main extension of this work is how to use the information out of this method to do an accurate multiple genome alignment. This will not be a big task, since significant information like where the mutations happen and which type of mutation events they are, are in the hand.

6.3. Conclusion

After doing the extension and solving all the open tasks as well as doing all the improvments, a complete tool is realized which will offer th following jobs: (1) large scale multiple genome alignment; (2) consideration of both sequence and structure information; (3) detection of the mutation events in terms of where they happen and what are their types. Efficiently doing all of these jobs in linear complexity (runtime and memory usage).

[2] The tool that is based on Arnoldi algorithm and used to solve the eigen problem in the current work.

APPENDIX A

Appendix A: Detailed results.

Here the detailed results of the benchmaking and the cross checking process for some parameters.

Abbrivations that used in this appendix:
Ins. : *Insertion.*
Dup. : *Duplication.*
Inv. : *Inversion.*
Ev. : Occurance of a mutation event.
NE. : No occurance of a mutation event.

Table A.1.: The performance of the method for the following parameter values: (a) Number of the vertical segment blocks = 3, (b) the size of the instances of the training dataset = 3, (c) the size of the vertical segment blocks are equals.

	Performance metrics.					
Discrimination	**ACC**	**PRE**	**REC**	**PRF**	**APR**	**ROC**
Ins. vs. Inv.	80.7	77.3	86.8	81.7	85.2	88.3
Ins. vs. Dup.	81.1	76.0	91.2	82.8	86.4	88.8
Ins. vs. NE	96.7	96.2	97.2	96.7	98.6	99.0
Dup. vs. Inv.	53.7	53.5	57.1	55.3	55.7	56.4
Inv. vs. NE	90.9	88.5	93.9	91.1	95.1	96.3
Dup. vs. NE	91.2	89.4	93.5	91.4	95.7	96.3
Ev. vs. NE	92.0	95.7	87.8	91.6	97.2	97.0

Table A.2.: The performance of the method for the following parameter values: (a) Number of the vertical segment blocks = 3, (b) the size of the instances of the training dataset = 4, (c) the size of the vertical segment blocks are equals.

	Performance metrics.					
Discrimination	**ACC**	**PRE**	**REC**	**PRF**	**APR**	**ROC**
Ins. vs. Inv.	84.2	83.5	85.2	84.4	89.9	91.7
Ins. vs. Dup.	84.2	80.6	90.2	85.1	90.5	92.2
Ins. vs. NE	97.0	96.2	97.9	97.0	98.7	99.2
Dup. vs. Inv.	54.4	56.2	40.0	46.3	57.7	58.3
Inv. vs. NE	90.6	89.5	92.0	90.7	94.6	95.8
Dup. vs. NE	89.8	88.2	91.8	90.0	94.9	95.7
Ev. vs. NE	91.9	94.0	89.5	91.7	97.4	96.9

Table A.3.: The performance of the method for the following parameter values: (a) Number of the vertical segment blocks = 3, (b) the size of the instances of the training dataset = 5, (c) the size of the vertical segment blocks are equals.

	Performance metrics.					
Discrimination	**ACC**	**PRE**	**REC**	**PRF**	**APR**	**ROC**
Ins. vs. Inv.	89.8	88.6	91.3	89.9	95.2	95.7
Ins. vs. Dup.	85.6	83.9	88.1	86.0	92.3	93.5
Ins. vs. NE	96.5	96.9	96.1	96.5	99.0	99.2
Dup. vs. Inv.	53.0	52.9	55.1	54.0	56.8	54.0
Inv. vs. NE	86.1	91.1	80.0	85.2	94.1	95.1
Dup. vs. NE	87.8	88.1	87.4	87.7	94.1	94.4
Ev. vs. NE	90.6	92.4	88.5	90.4	96.9	96.4

Table A.4.: The performance of the method for the following parameter values: (a) Number of the vertical segment blocks = 4, (b) the size of the instances of the training dataset = 3, (c) the size of the vertical segment blocks are equals.

	Performance metrics.					
Discrimination	**ACC**	**PRE**	**REC**	**PRF**	**APR**	**ROC**
Ins. vs. Inv.	81.7	79.4	85.7	82.4	87.1	89.6
Ins. vs. Dup.	82.6	77.4	92.2	84.1	87.9	90.6
Ins. vs. NE	96.9	96.7	97.0	96.9	99.2	99.4
Dup. vs. Inv.	56.1	56.6	52.3	54.3	56.0	58.3
Inv. vs. NE	88.4	88.6	88.1	88.4	92.5	94.1
Dup. vs. NE	87.8	88.7	86.6	87.6	93.3	94.1
Ev. vs. NE	90.0	90.8	89.1	89.9	96.0	96.0

Table A.5.: The performance of the method for the following parameter values: (a) Number of the vertical segment blocks = 4, (b) the size of the instances of the training dataset = 4, (c) the size of the vertical segment blocks are equals.

	Performance metrics.					
Discrimination	**ACC**	**PRE**	**REC**	**PRF**	**APR**	**ROC**
Ins. vs. Inv.	85.5	82.4	90.4	86.2	89.9	92.1
Ins. vs. Dup.	87.7	84.3	92.5	88.2	91.4	93.4
Ins. vs. NE	97.0	97.2	96.9	97.0	99.0	99.3
Dup. vs. Inv.	56.5	57.8	48.0	52.4	56.7	59.4
Inv. vs. NE	89.6	89.3	89.9	89.6	94.3	95.1
Dup. vs. NE	87.9	87.6	88.4	88.0	94.8	95.2
Ev. vs. NE	90.3	92.2	88.0	90.1	96.4	96.2

Table A.6.: The performance of the method for the following parameter values: (a) Number of the vertical segment blocks = 4, (b) the size of the instances of the training dataset = 5, (c) the size of the vertical segment blocks are equals.

	Performance metrics.					
Discrimination	**ACC**	**PRE**	**REC**	**PRF**	**APR**	**ROC**
Ins. vs. Inv.	86.7	85.7	88.1	86.9	93.4	94.1
Ins. vs. Dup.	89.3	86.4	93.2	89.7	94.3	95.3
Ins. vs. NE	96.1	95.8	96.4	96.1	99.0	99.2
Dup. vs. Inv.	55.3	56.3	47.4	51.5	55.8	57.8
Inv. vs. NE	88.3	87.6	89.3	88.5	94.1	95.1
Dup. vs. NE	87.8	87.3	88.5	87.9	94.0	94.3
Ev. vs. NE	89.9	93.0	86.4	89.5	96.3	95.8

Table A.7.: The performance of the method for the following parameter values: (a) Number of the vertical segment blocks = 5, (b) the size of the instances of the training dataset = 3, (c) the size of the vertical segment blocks are equals.

	Performance metrics.					
Discrimination	**ACC**	**PRE**	**REC**	**PRF**	**APR**	**ROC**
Ins. vs. Inv.	85.5	83.1	89.3	86.0	89.6	91.7
Ins. vs. Dup.	85.1	84.2	86.4	85.3	90.9	92.6
Ins. vs. NE	96.4	96.0	96.9	96.5	98.9	99.0
Dup. vs. Inv.	53.1	53.6	46.4	49.7	54.0	55.5
Inv. vs. NE	86.4	87.7	84.7	86.1	92.7	93.7
Dup. vs. NE	87.0	88.2	85.5	86.8	93.7	94.0
Ev. vs. NE	88.9	89.2	88.6	88.9	95.3	95.4

Table A.8.: The performance of the method for the following parameter values: (a) Number of the vertical segment blocks = 5, (b) the size of the instances of the training dataset = 4, (c) the size of the vertical segment blocks are equals.

	Performance metrics.					
Discrimination	**ACC**	**PRE**	**REC**	**PRF**	**APR**	**ROC**
Ins. vs. Inv.	87.4	89.0	85.3	87.1	93.1	94.2
Ins. vs. Dup.	87.3	85.4	89.9	87.6	92.8	94.1
Ins. vs. NE	96.0	96.0	96.0	96.0	98.9	98.9
Dup. vs. Inv.	52.0	52.2	47.7	49.8	52.4	53.7
Inv. vs. NE	87.6	88.4	86.5	87.4	94.2	94.6
Dup. vs. NE	87.5	89.5	85.0	87.2	93.5	93.8
Ev. vs. NE	89.5	90.0	88.9	89.5	95.8	95.7

Table A.9.: The performance of the method for the following parameter values: (a) Number of the vertical segment blocks = 5, (b) the size of the instances of the training dataset = 5, (c) the size of the vertical segment blocks are equals.

	Performance metrics.					
Discrimination	**ACC**	**PRE**	**REC**	**PRF**	**APR**	**ROC**
Ins. vs. Inv.	86.1	88.5	83.0	85.7	92.8	94.0
Ins. vs. Dup.	89.3	89.6	88.9	89.2	94.5	95.2
Ins. vs. NE	96.5	97.3	95.5	96.4	99.1	99.0
Dup. vs. Inv.	51.7	51.8	50.1	50.9	52.7	53.6
Inv. vs. NE	87.6	88.1	87.0	87.5	94.0	94.7
Dup. vs. NE	86.8	89.1	83.9	86.4	92.9	93.8
Ev. vs. NE	89.4	90.3	88.2	89.2	95.9	95.6

Table A.10.: The performance of the method for the following parameter values: (a) Number of the vertical segment blocks = 6, (b) the size of the instances of the training dataset = 3, (c) the size of the vertical segment blocks are equals.

	Performance metrics.					
Discrimination	**ACC**	**PRE**	**REC**	**PRF**	**APR**	**ROC**
Ins. vs. Inv.	86.0	85.8	86.2	86.0	91.4	92.1
Ins. vs. Dup.	86.1	86.1	86.1	86.1	92.7	92.9
Ins. vs. NE	93.8	94.7	92.9	93.8	98.2	97.9
Dup. vs. Inv.	53.8	54.0	50.7	52.3	53.1	55.8
Inv. vs. NE	84.3	87.1	80.5	83.7	91.3	92.0
Dup. vs. NE	85.7	86.5	84.5	85.5	92.7	92.6
Ev. vs. NE	87.8	88.2	87.4	87.8	94.4	94.5

Table A.11.: The performance of the method for the following parameter values: (a) Number of the vertical segment blocks = 6, (b) the size of the instances of the training dataset = 4, (c) the size of the vertical segment blocks are equals.

	Performance metrics.					
Discrimination	**ACC**	**PRE**	**REC**	**PRF**	**APR**	**ROC**
Ins. vs. Inv.	87.0	86.2	88.1	87.1	92.7	93.0
Ins. vs. Dup.	87.7	86.3	89.6	87.9	94.3	94.4
Ins. vs. NE	94.9	96.6	93.2	94.8	98.7	98.6
Dup. vs. Inv.	53.0	53.5	46.5	49.8	51.8	54.8
Inv. vs. NE	84.4	87.4	80.3	83.7	92.3	92.5
Dup. vs. NE	87.4	89.3	85.0	87.1	93.3	93.8
Ev. vs. NE	88.7	89.3	88.1	88.7	95.1	95.2

Table A.12.: The performance of the method for the following parameter values: (a) Number of the vertical segment blocks = 6, (b) the size of the instances of the training dataset = 5, (c) the size of the vertical segment blocks are equals.

	Performance metrics.					
Discrimination	**ACC**	**PRE**	**REC**	**PRF**	**APR**	**ROC**
Ins. vs. Inv.	88.0	88.5	87.4	87.9	94.6	94.6
Ins. vs. Dup.	87.7	89.9	84.9	87.3	94.5	94.5
Ins. vs. NE	95.6	97.3	93.8	95.5	98.6	98.6
Dup. vs. Inv.	52.6	53.4	41.8	46.9	51.9	54.6
Inv. vs. NE	86.0	89.3	82.0	85.5	93.4	94.1
Dup. vs. NE	86.0	88.2	83.2	85.6	93.3	93.7
Ev. vs. NE	88.9	89.4	88.3	88.9	95.4	95.2

Table A.13.: The performance of the method for the following parameter values: (a) Number of the vertical segment blocks = 7, (b) the size of the instances of the training dataset = 3, (c) the size of the vertical segment blocks are equals.

	Performance metrics.					
Discrimination	**ACC**	**PRE**	**REC**	**PRF**	**APR**	**ROC**
Ins. vs. Inv.	84.4	85.2	83.2	84.2	90.2	91.1
Ins. vs. Dup.	84.8	85.3	84.0	84.7	91.9	91.9
Ins. vs. NE	93.0	94.5	91.4	92.9	97.8	97.7
Dup. vs. Inv.	51.5	51.8	44.3	47.8	50.7	52.9
Inv. vs. NE	83.3	85.9	79.6	82.6	90.2	91.3
Dup. vs. NE	83.5	88.0	77.6	82.4	91.9	91.7
Ev. vs. NE	86.7	86.3	87.3	86.8	93.7	93.8

Table A.14.: The performance of the method for the following parameter values: (a) Number of the vertical segment blocks = 7, (b) the size of the instances of the training dataset = 4, (c) the size of the vertical segment blocks are equals.

	Performance metrics.					
Discrimination	**ACC**	**PRE**	**REC**	**PRF**	**APR**	**ROC**
Ins. vs. Inv.	84.8	86.1	82.9	84.5	92.1	92.4
Ins. vs. Dup.	87.0	87.0	87.0	87.0	93.5	93.8
Ins. vs. NE	93.4	96.5	90.1	93.1	98.2	98.0
Dup. vs. Inv.	53.1	53.4	48.3	50.7	50.9	54.5
Inv. vs. NE	85.7	89.8	80.5	84.9	93.2	93.5
Dup. vs. NE	86.0	89.1	82.0	85.4	92.9	93.3
Ev. vs. NE	87.8	87.7	88.0	87.8	94.4	94.5

Table A.15.: The performance of the method for the following parameter values: (a) Number of the vertical segment blocks = 7, (b) the size of the instances of the training dataset = 5, (c) the size of the vertical segment blocks are equals.

	Performance metrics.					
Discrimination	**ACC**	**PRE**	**REC**	**PRF**	**APR**	**ROC**
Ins. vs. Inv.	84.9	86.4	82.8	84.6	91.8	92.4
Ins. vs. Dup.	87.2	87.6	86.7	87.1	94.5	94.7
Ins. vs. NE	94.1	95.9	92.1	94.0	98.2	98.3
Dup. vs. Inv.	53.0	53.2	48.7	50.9	50.6	54.0
Inv. vs. NE	85.0	87.5	81.7	84.5	92.7	93.0
Dup. vs. NE	84.7	86.0	83.0	84.4	91.9	92.1
Ev. vs. NE	88.3	88.0	88.7	88.4	95.0	95.0

Table A.16.: The performance of the method for the following parameter values: (a) Number of the vertical segment blocks = 3, (b) the size of the instances of the training dataset = 3, (c) the size of the vertical segment blocks are halfs.

	Performance metrics.					
footnotesizeDiscrimination	**ACC**	**PRE**	**REC**	**PRF**	**APR**	**ROC**
Ins. vs. Inv.	75.1	73.1	79.4	76.1	81.2	82.3
Ins. vs. Dup.	73.0	71.1	77.5	74.2	78.2	80.0
Ins. vs. NE	89.8	89.5	90.1	89.8	95.5	95.2
Dup. vs. Inv.	53.5	53.7	51.0	52.3	53.0	54.9
Inv. vs. NE	85.4	83.4	88.2	85.8	92.0	92.7
Dup. vs. NE	84.7	82.8	87.7	85.2	92.1	92.6
Ev. vs. NE	85.4	90.3	79.3	84.4	93.2	93.3

Table A.17.: The performance of the method for the following parameter values: (a) Number of the vertical segment blocks = 3, (b) the size of the instances of the training dataset = 4, (c) the size of the vertical segment blocks are halfs.

	Performance metrics.					
Discrimination	**ACC**	**PRE**	**REC**	**PRF**	**APR**	**ROC**
Ins. vs. Inv.	75.6	73.7	79.7	76.6	82.4	83.3
Ins. vs. Dup.	77.4	77.6	77.1	77.4	83.1	84.9
Ins. vs. NE	91.0	91.5	90.4	91.0	96.4	96.1
Dup. vs. Inv.	53.6	53.6	54.5	54.0	54.0	55.6
Inv. vs. NE	85.3	80.1	93.9	86.5	92.5	93.2
Dup. vs. NE	84.6	83.7	85.8	84.7	92.0	92.2
Ev. vs. NE	87.8	91.2	83.7	87.3	94.5	94.6

Table A.18.: The performance of the method for the following parameter values: (a) Number of the vertical segment blocks = 3, (b) the size of the instances of the training dataset = 5, (c) the size of the vertical segment blocks are halfs.

	Performance metrics.					
Discrimination	**ACC**	**PRE**	**REC**	**PRF**	**APR**	**ROC**
Ins. vs. Inv.	79.8	78.3	82.4	80.3	87.4	88.2
Ins. vs. Dup.	78.5	77.5	80.4	78.9	83.9	85.8
Ins. vs. NE	92.6	93.0	92.2	92.6	97.1	97.0
Dup. vs. Inv.	54.5	55.2	48.0	51.3	54.1	56.9
Inv. vs. NE	88.3	85.9	91.6	88.7	94.1	94.8
Dup. vs. NE	85.7	84.6	87.3	85.9	92.9	93.2
Ev. vs. NE	88.0	92.8	82.4	87.3	95.0	95.1

Table A.19.: The performance of the method for the following parameter values: (a) Number of the vertical segment blocks = 4, (b) the size of the instances of the training dataset = 3, (c) the size of the vertical segment blocks are halfs.

	Performance metrics.					
Discrimination	**ACC**	**PRE**	**REC**	**PRF**	**APR**	**ROC**
Ins. vs. Inv.	75.0	74.3	76.2	75.3	79.8	81.8
Ins. vs. Dup.	75.0	73.0	79.5	76.1	80.3	82.2
Ins. vs. NE	91.0	90.6	91.5	91.0	96.4	96.3
Dup. vs. Inv.	55.4	55.5	54.5	55.0	55.4	57.2
Inv. vs. NE	87.1	84.3	91.1	87.6	93.7	94.4
Dup. vs. NE	83.8	80.7	88.8	84.6	91.9	92.1
Ev. vs. NE	87.4	91.1	83.0	86.9	94.7	94.6

Table A.20.: The performance of the method for the following parameter values: (a) Number of the vertical segment blocks = 4, (b) the size of the instances of the training dataset = 4, (c) the size of the vertical segment blocks are halfs.

	Performance metrics.					
Discrimination	**ACC**	**PRE**	**REC**	**PRF**	**APR**	**ROC**
Ins. vs. Inv.	77.3	75.8	80.3	78.0	84.3	85.6
Ins. vs. Dup.	76.8	74.8	80.7	77.6	82.6	84.8
Ins. vs. NE	92.5	91.7	93.5	92.6	97.3	97.3
Dup. vs. Inv.	56.4	56.4	56.3	56.3	56.3	58.1
Inv. vs. NE	87.4	84.9	90.9	87.8	93.8	94.5
Dup. vs. NE	87.2	86.5	88.3	87.4	93.4	93.8
Ev. vs. NE	88.6	93.1	83.3	88.0	95.6	95.4

Table A.21.: The performance of the method for the following parameter values: (a) Number of the vertical segment blocks = 4, (b) the size of the instances of the training dataset = 5, (c) the size of the vertical segment blocks are halfs.

	Performance metrics.					
Discrimination	**ACC**	**PRE**	**REC**	**PRF**	**APR**	**ROC**
Ins. vs. Inv.	79.2	76.6	84.1	80.2	86.0	87.4
Ins. vs. Dup.	78.2	74.5	86.0	79.8	84.9	86.5
Ins. vs. NE	93.9	93.6	94.4	94.0	97.8	97.7
Dup. vs. Inv.	56.8	57.5	53.2	54.7	58.5	59.3
Inv. vs. NE	88.6	83.9	95.4	89.3	94.8	95.5
Dup. vs. NE	87.1	84.5	91.0	87.6	93.4	94.0
Ev. vs. NE	89.4	93.3	84.9	88.9	95.9	95.6

Table A.22.: The performance of the method for the following parameter values: (a) Number of the vertical segment blocks = 5, (b) the size of the instances of the training dataset = 3, (c) the size of the vertical segment blocks are halfs.

	Performance metrics.					
Discrimination	**ACC**	**PRE**	**REC**	**PRF**	**APR**	**ROC**
Ins. vs. Inv.	74.7	73.1	78.2	75.6	79.0	81.8
Ins. vs. Dup.	75.4	73.1	80.4	76.6	78.8	82.0
Ins. vs. NE	93.0	93.0	93.0	93.0	96.2	96.6
Dup. vs. Inv.	54.0	53.9	55.7	54.8	56.2	56.1
Inv. vs. NE	87.3	84.9	90.7	87.7	92.1	93.7
Dup. vs. NE	86.8	84.6	89.9	87.2	93.6	93.8
Ev. vs. NE	88.0	91.4	83.9	87.5	95.1	94.9

Table A.23.: The performance of the method for the following parameter values: (a) Number of the vertical segment blocks = 5, (b) the size of the instances of the training dataset = 4, (c) the size of the vertical segment blocks are halfs.

	Performance metrics.					
Discrimination	**ACC**	**PRE**	**REC**	**PRF**	**APR**	**ROC**
Ins. vs. Inv.	78.1	75.3	83.7	79.2	84.3	86.1
Ins. vs. Dup.	79.3	75.6	86.5	80.7	84.8	86.9
Ins. vs. NE	93.7	93.3	94.1	93.7	97.6	97.7
Dup. vs. Inv.	55.3	55.7	51.7	53.6	56.3	57.2
Inv. vs. NE	88.8	86.0	92.6	89.2	94.3	95.2
Dup. vs. NE	87.1	85..2	89.8	87.5	93.5	94.1
Ev. vs. NE	89.1	92.6	85.0	88.6	95.7	95.4

Table A.24.: The performance of the method for the following parameter values: (a) Number of the vertical segment blocks = 5, (b) the size of the instances of the training dataset = 5, (c) the size of the vertical segment blocks are halfs.

	Performance metrics.					
Discrimination	**ACC**	**PRE**	**REC**	**PRF**	**APR**	**ROC**
Ins. vs. Inv.	79.8	79.7	80.1	80.0	86.0	87.8
Ins. vs. Dup.	81.3	79.5	84.4	81.9	86.9	88.6
Ins. vs. NE	94.0	93.0	95.0	94.0	98.1	98.1
Dup. vs. Inv.	54.0	54.3	51.4	52.8	54.6	55.7
Inv. vs. NE	88.8	86.0	92.6	89.2	94.3	95.2
Dup. vs. NE	89.5	88.2	91.3	89.7	94.8	95.5
Ev. vs. NE	88.0	86.9	89.5	88.2	94.0	94.5

Table A.25.: The performance of the method for the following parameter values: (a) Number of the vertical segment blocks = 6, (b) the size of the instances of the training dataset = 3, (c) the size of the vertical segment blocks are halfs.

	Performance metrics.					
Discrimination	**ACC**	**PRE**	**REC**	**PRF**	**APR**	**ROC**
Ins. vs. Inv.	74.8	72.4	80.0	76.0	78.5	82.0
Ins. vs. Dup.	77.0	76.0	79.0	77.5	81.4	83.7
Ins. vs. NE	91.8	92.1	91.6	91.8	97.0	96.7
Dup. vs. Inv.	53.3	53.6	49.2	51.3	53.0	55.4
Inv. vs. NE	87.5	84.3	92.2	88.0	93.4	94.6
Dup. vs. NE	86.6	84.5	89.7	87.0	94.3	94.4
Ev. vs. NE	88.1	92.2	83.2	87.5	95.3	95.1

Table A.26.: The performance of the method for the following parameter values: (a) Number of the vertical segment blocks = 6, (b) the size of the instances of the training dataset = 4, (c) the size of the vertical segment blocks are halfs.

	Performance metrics.					
Discrimination	**ACC**	**PRE**	**REC**	**PRF**	**APR**	**ROC**
Ins. vs. Inv.	77.1	77.0	77.4	77.2	81.2	83.9
Ins. vs. Dup.	79.7	76.9	84.9	80.7	85.5	87.1
Ins. vs. NE	93.2	92.7	93.9	93.3	97.9	97.8
Dup. vs. Inv.	52.9	53.0	51.7	52.3	53.7	55.1
Inv. vs. NE	88.7	86.4	91.9	89.1	94.2	94.9
Dup. vs. NE	88.3	87.2	89.9	88.5	94.2	94.9
Ev. vs. NE	89.9	93.7	85.5	89.4	96.1	95.8

Table A.27.: The performance of the method for the following parameter values: (a) Number of the vertical segment blocks = 6, (b) the size of the instances of the training dataset = 5, (c) the size of the vertical segment blocks are halfs.

	Performance metrics.					
Discrimination	**ACC**	**PRE**	**REC**	**PRF**	**APR**	**ROC**
Ins. vs. Inv.	79.9	77.7	83.8	80.7	85.3	87.5
Ins. vs. Dup.	80.3	79.3	81.9	80.6	87.2	88.6
Ins. vs. NE	94.2	93.1	95.5	94.3	98.0	98.1
Dup. vs. Inv.	52.2	52.9	40.9	46.2	53.0	54.7
Inv. vs. NE	90.2	87.8	93.2	90.4	94.6	95.7
Dup. vs. NE	88.8	87.1	91.1	89.1	94.7	95.3
Ev. vs. NE	90.1	94.2	85.6	89.7	96.4	96.1

Table A.28.: The performance of the method for the following parameter values: (a) Number of the vertical segment blocks = 7, (b) the size of the instances of the training dataset = 3, (c) the size of the vertical segment blocks are halfs.

	Performance metrics.					
Discrimination	**ACC**	**PRE**	**REC**	**PRF**	**APR**	**ROC**
Ins. vs. Inv.	75.5	74.3	78.1	76.1	79.9	82.3
Ins. vs. Dup.	77.3	74.4	83.3	78.6	81.9	83.8
Ins. vs. NE	92.2	91.5	93.2	92.3	97.3	97.2
Dup. vs. Inv.	52.6	53.0	47.0	49.8	51.6	53.6
Inv. vs. NE	87.7	84.3	92.6	88.3	93.1	94.3
Dup. vs. NE	87.7	86.0	90.2	88.0	93.7	94.4
Ev. vs. NE	88.0	92.6	82.7	87.3	95.3	95.1

Table A.29.: The performance of the method for the following parameter values: (a) Number of the vertical segment blocks = 7, (b) the size of the instances of the training dataset = 4, (c) the size of the vertical segment blocks are halfs.

	Performance metrics.					
Discrimination	**ACC**	**PRE**	**REC**	**PRF**	**APR**	**ROC**
Ins. vs. Inv.	78.3	77.2	80.4	78.8	85.0	86.8
Ins. vs. Dup.	79.4	77.3	83.2	80.2	84.9	87.2
Ins. vs. NE	93.8	92.7	95.1	93.9	97.7	97.8
Dup. vs. Inv.	53.2	53.1	55.2	54.1	52.3	53.7
Inv. vs. NE	88.5	87.0	90.4	88.7	94.0	94.9
Dup. vs. NE	88.4	85.9	91.9	88.8	94.0	94.7
Ev. vs. NE	89.4	93.5	84.7	88.9	96.0	95.7

Table A.30.: The performance of the method for the following parameter values: (a) Number of the vertical segment blocks = 7, (b) the size of the instances of the training dataset = 5, (c) the size of the vertical segment blocks are halfs.

	Performance metrics.					
Discrimination	**ACC**	**PRE**	**REC**	**PRF**	**APR**	**ROC**
Ins. vs. Inv.	80.7	80.0	81.9	80.9	87.5	88.7
Ins. vs. Dup.	80.0	77.1	85.1	80.9	86.2	87.9
Ins. vs. NE	94.5	94.0	95.0	94.5	98.0	98.0
Dup. vs. Inv.	54.9	55.6	49.3	52.2	55.5	56.4
Inv. vs. NE	89.2	86.1	93.4	89.7	95.0	95.6
Dup. vs. NE	88.5	86.4	91.3	88.8	93.7	94.8
Ev. vs. NE	90.2	93.7	86.2	89.8	96.4	96.0

Bibliography

[1] Fabrizio Costa and Kurt De Grave. Fast neighborhood subgraph pairwise distance kernel, 2010.

[2] Steffen Heyne, Fabrizio Costa, Dominic Rose, and Rolf Backofen. GraphClust: alignment-free structural clustering of local rna secondary structures. *Bioinformatics*, 28(12):i224–i232, 2012. doi: 10.1093/bioinformatics/bts224.

[3] R. B. Lehoucq, D. C. Sorensen, and C. Yang. Arpack users guide: Solution of large scale eigenvalue problems by implicitly restarted arnoldi methods., 1997.

[4] Needleman Saul B. and Wunsch Christian D. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Molecular Biology*, 48(3):443–453, 1970. doi: 10.1016/0022-2836(70)90057-4.

[5] Smith Temple F. and Waterman Michael S. Identification of common molecular subsequences. *Molecular Biology*, 147:195–197, 1981. doi: 10.1016/0022-2836(81) 90087-5.

[6] Lipman DJ and Pearson WR. Rapid and sensitive protein similarity searches. *Science*, 227(4693):1435–1441, 1985. doi: 10.1126/science.2983426.

[7] Higgins D. G. and Sharp P. M. CLUSTAL: a package for performing multiple sequence alignment on a microcomputer. *Gene*, 73(1):237–244, 1988. doi: 10.1016/0378-1119(88)90330-7.

[8] Notredame C., Higgins D., and Heringa J. T-Coffee: A novel method for fast and accurate multiple sequence alignment. *J Mol Biol*, 302(1):205–217, 2000. doi: 10.1006/jmbi.2000.4042.

[9] Kazutaka Katoh, Kazuharu Misawa, Keiichi Kuma, and Takashi Miyata. MAFFT: a novel method for rapid multiple sequence alignment based on fast fourier transform. *Nucleic Acids Res.*, 30(14):3059–3066, 2002. doi: 10.1093/nar/gkf436.

[10] Robert C. Edgar. MUSCLE: a multiple sequence alignment method with reduced time and space complexity. *BMC Bioinformatics*, 5:113, 2004.

[11] Chuong B. Do, Michael Brudno, and Serafim Batzoglou. Probcons: Probabilistic consistency-based multiple alignment of amino acid sequences. *Genome Res.*, 15: 330–340, 2005. doi: 10.1101/gr.2821705.

[12] Hoehl M., Kurtz S., and Ohlebusch E. Efficient multiple genome alignment. *Bioinformatics*, 18(1):S312–S320, 2002. doi: 10.1093/bioinformatics/18.suppl_1.S.

[13] B. Morgenstern, K. Frech, A. Dress, and T. Werner. DIALIGN: finding local similarities by multiple sequence alignment. *Bioinformatics*, 14(3):290–294, 1998. doi: 10.1093/bioinformatics/14.3.290.

[14] Darling A., Mau B., , and Perna N. progressiveMauve: Multiple genome alignment with gene gain, loss and rearrangement. *PLoS ONE*, 5(6), 2010.

[15] Treangen T. and Messeguer X. M-GCAT: Interactively and efficiently constructing large-scale multiple genome comparison frameworks in closely related species. *BMC Bioinformatics*, 7(433), 2006.

[16] Fan R. and K. Chung. *SPECTRAL GRAPH THEORY*. AMS, 1992. ISBN 0-8218-0315-8.

[17] Michal Holcapeka and Tomas Tichy. A smoothing filter based on fuzzy transform. *Fuzzy sets and systems*, 180(1):69–97, 2011. doi: 10.1016/j.fss.2011.05.028.

[18] Bertram J. The molecular biology of cancer. *Aspects Med.*, 21(6):167–223, 2000. doi: 10.1016/S0098-2997(00)00007-8.

[19] National Human GenomeResearch Institute. `http://www.genome.gov/`.

[20] Lipman D. J., Kececioglu J. D., and Altschul S.F. A tool for multiple sequence alignment. *Proc Natl Acad Sci USA.*, 86(12):4412–4415, 1989. doi: 10.1073/pnas.86.12.4412.

[21] Robert C. Edgar. Muscle: multiple sequence alignment with high accuracy and high throughput. *NUCLEIC ACIDS RES*, 32:1792–1797, 2004.

[22] Thompson J. D., Higgins D. G., and Gibson T. J. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res*, 22(22): 4673–4680, 1994. doi: 10.1093/nar/22.22.4673.

[23] Chenna R., Sugawara H., Koike T., Lopez R., Gibson T., and Higgins D. Multiple sequence alignment with the clustal series of programs. *Nucleic Acids Res*, 31(13): 3497–3500, 2003. doi: 10.1093/nar/gkg500.

[24] Hirosawa M., Totoki Y., Hoshida M., and Ishikawa M. Comprehensive study on iterative algorithms of multiple sequence alignment. *Comput Appl Biosci*, 11(1): 13–18, 1995. doi: 10.1093/bioinformatics/11.1.13.

[25] Karplus K., Barret C., and Hughey R. Hidden markov models for detecting remote protein homologies. *Bioinformatics*, 14(10):846–856, 1998. doi: 10.1093/bioinformatics/14.10.846.

[26] Darling A., Mau B., Blattner F., and Perna N. Mauve: Multiple alignment of conserved genomic sequence with rearrangements. *Genome Research*, 14:1394–1403, 2004. doi: 10.1101/gr.2289704.

[27] Blanchette M. and et. al. Aligning multiple genomic sequences with the threaded blockset aligner. *Genome Research*, 14(4):798–715, 2004. doi: 10.1101/gr.1933104.

[28] Jeong-Hyeon Choi, Kwangmin Choi, Hwan-Gue Cho, and Sun Kim. *Multiple Genome Alignment by Clustering Pairwise Matches*. Springer Berlin Heidelberg. Vol. 3388, 2005.

[29] Brudno M. and et. al. LAGAN and Multi-LAGAN: Efficient tools for large-scale multiple alignment of genomic dna. *Genome Res.*, 13(3):721–731, 2003. doi: 10.1101/gr.926603.

[30] M. Blanchette. Aligning multiple genomic sequences with the threaded blockset aligner. *Genome Research*, 14(4):708–715, 2004. doi: 10.1101/gr.

[31] Cooper G., Stone E., Asimenos G., and al E. Distribution and intensity of constraint in mammalian genomic sequence. *Genome Research*, 15:901–913, 2005. doi: 10.1101/gr.3577405.

[32] Colin Noel Dewey. *Whole-genome alignments and polytopes for comparative genomics.* PhD thesis, EECS Department, University of California, Berkeley, Aug 2006.

[33] Robert S. Harris. Improved pairwise alignment of genomic DNA. Technical report, 2007.

[34] Höhl M., Kurtz S., and Ohlebusch E. Efficient multiple genome alignment, 2002.

[35] Kent W. Blatâthe blast-like alignment tool. *Genome Research*, 12(4):656–64, 2002.

[36] Paten B., Herrero J., Beal K., Fitzgerald S., and al E. Enredo and pecan: genome-wide mammalian consistency-based multiple alignment with paralogs. *Genome Research*, 18(11):1814â1828, 2008a.

[37] Paten B., Herrero J., Fitzgerald S., Beal K., and al E. Genome-wide nucleotide-level mammalian ancestor reconstruction. *Genome Research*, 18(11):1829–1843, 2008b. doi: 10.1101/gr.076521.108.

[38] Schwartz S., Kent W., Smit A., Zhang Z., and E. al. Humanâmouse alignments with BLASTZ. *Genome Research*, 13:103–107, 2003. doi: 10.1101/gr.809403.

[39] Hernzandez V., Romzan J., Tomzas A., and V. Vidal. *SLEPc Scalable Library for Eigenvalue Problem Computations.* Springer-Verlag Berlin, Heidelberg, 2003.

[40] Calvetti D., Reichel L., and Sorensen D. C. An implicitly restarted lanczos method for large symmetric eigenvalue problems. *Electronic Transactions on Numerical Analysis*, 2(1):1–21, 1994. doi: 10.1.1.37.8747.

[41] Andrew Y. Ng, Alice X. Zheng, and Michael I. Jordan. Link analysis, eigenvectors and stability. In *IN IJCAI-01*, volume 17, pages 903–910. 10.1.1.20.7296, 2001.

[42] Hisashi K., Tsuda K., and Inokuchi A. Kernels for graph. *The Journal of Machine Learning Research*, 11, 2010.

[43] Blondel V. and Van Dooren P. Similarity matrices for pairs of graphs. *30th International Colloquium*, 2719:739–750, 2003.

[44] Lawson D. and Falush D. Similarity matrices and clustering algorithms for population identi?cation using genetic data. *GENOMICS AND HUMAN GENETICS*, 13, 2012.

[45] Stephen Guattery and Gary L. Miller. On the quality of spectral separators. *SIAM JOURNAL ON MATRIX ANALYSIS AND APPLICATIONS*, 19:701–719, 1998.

[46] Weiss Y. Segmentation using eigenvectors: a unifying view. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 975–982 vol.2, 1999. doi: 10.1109/ICCV.1999.790354.

[47] Ulrike Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4): 395–416, 2007. doi: 10.1007/s11222-007-9033-z.

[48] Yehuda Koren. Drawing graphs by eigenvectors: Theory and practice. *Computers and Mathematics with Applications*, 49:2005, 2005. doi: 10.1016/j.camwa.2004.08.015.

[49] Yehuda Koren. On spectral graph drawing. In *Proc. 9th Inter. Computing and Combinatorics Conference , LNCS 2697*, pages 496–508. Springer-Verlag, 2002.

[50] Ali Civril, Malik Magdon-Ismail, and Eli Bocek-Rivele. Ssde: Fast graph drawing using sampled spectral distance embedding, 2006.

[51] Ulrik Brandes, Daniel Fleischer, and Thomas Puppe. Dynamic spectral layout of small worlds. In *IN PROC. 13TH INT. SYMP. GRAPH DRAWING, GD*, pages 25–36, 2005.

[52] Yetian Chen. Learning classifiers from imbalanced only positive and unlabeled data sets. 2008.

[53] Ganganwar V. An overview of classification algorithms for imbalanced datasets. *International Journal of Emerging Technology and Advanced Engineering*, 2(4), 2012.

[54] Terence A Brown. *Genomics.* Oxford Wiley-Liss. ISMB 2002, 2nd edition, 2002.

[55] Omary Z. and Mtenzi F. Machine learning approach to identifying the dataset threshold for the performance estimators in supervised learning. *IJL*, 3(3), 2010.

[56] Strang G. *Introduction to Linear Algebra.* Wellesley-Cambridge Press and SIAM, 4th edition, 2009.

[57] Aminetzach YT, Macpherson JM, and Petrov DA. Pesticide resistance via transposition-mediated adaptive gene truncation in drosophila. *Science*, 309(5735): 764–7, 2005. doi: 10.1126/science.1112699.

[58] Burrus V. and Waldor M. Shaping bacterial genomes with integrative and conjugative elements. *Res. Microbiol.*, 155(5):376–86, 2004. doi: 10.1016/j.resmic.2004.01.012.

[59] Tom Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters - Special issue*, 27(8):861–874, 2006. doi: 10.1016/j.patrec.2005.10.010.

[60] N. L. C. Talbot and G. C. Cawley. A quadratic index assignment algorithm for vector quantisation over noisy transmission channels. In *In Proceedings of the Institute of Acoustics Autumn Conference on Speech and Hearing*, pages 195–199, 1996.

[61] Kent W., Baertsch R., Hinrichs A., and al E. Evolution's cauldron: duplication, deletion, and rearrangement in the mouse and human genomes. *PNAS*, 100(20): 11484â–11489, 2003. doi: 10.1073/pnas.1932072100.