An Algorithm to Build a Multi-Genome Reference and its Application

Dissertation
der Mathematisch-Naturwissenschaftlichen Fakultät
der Eberhard Karls Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von
Seyedeh Leily Rabbani
aus Mashhad, Iran

Tübingen
2018

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualikation:	04.10.2018
Dekan:	Prof. Dr. Wolfgang Rosenstiel
1. Berichterstatter:	Prof. Dr. Detlef Weigel
2. Berichterstatter:	Prof. Dr. Michael Kaufmann

Acknowledgement

Firstly, I would like to thank my supervisor Prof. Dr. Detlef Weigel, Director at the Max Planck Institute for Developmental Biology. Without his steadfast support and insight this project would not have come to such a fruitful conclusion.

I would also like to acknowledge my external supervisor Prof. Dr. Michael Kaufmann of the Department of Computer Science at the University of Tübingen for his support during the years of my PhD. He was always on hand to offer invaluable advice and guidance. Thanks are also due to Prof. Dr. Kay Nieselt and Prof. Dr. Stephan Ossowski for their gracious acceptance of their role as thesis defence committee members.

I would also like to give special thanks to Jonas Müller. Without his help it would have been extremely difficult to overcome those unforeseen challenges which arose along the way. Special thanks are also due to Dr. Jörg Hagmann who was not only a valued colleague but also a great friend.

Moreover, I would like to thank Dr. Rebecca Schwab for being a constant light that guided me through the course of my PhD.

Special mention should also be given to Hülya Wicher for her skill navigating the labyrinthine bureaucracy and speeding me on my way.

I would also like to express my appreciation to Dr. Ilja Bezrukov for helping me in hard times, and all the other members of Weigel world for making my life full of joy and fun during my time in Tübingen.

Finally, I would like to acknowledge Dr. Barbara Hummel for her precious help in translating the abstract into German, my parents, my sister and my brother who have provided me with moral support throughout the difficult times, Simon Rapple for never leaving me alone in the most arduous of times and my friends Maryam, Kasra, Sofia, Maria, Sandra, Nicolas, Jorge, Stefan, Farshad, Shervin, Reza, Vahid, Pari and Moji who proved there is no border for friendship and were always there for me without question or hesitation.

Zusammenfassung

Das Erbgut eines Organismus ist in seiner DNA-Sequenz gespeichert. DNA-Sequenzierung versucht, Stücke dieser Information zu lesen, die anschließend zusammengesetzt werden, um vollständige DNA-Komplemente zu generieren. In den letzten zwanzig Jahren wurden in diesen Technologien erstaunliche Fortschritte erzielt. Diese fortschrittlichen Methoden haben zu einer enormen Mengen an Sequenzdaten mit extremer Diversität geführt und haben die Resequenzierung eines ganzen Genoms ermöglicht. Daher zielen viele groß angelegte Genomprojekte darauf ab, diese Daten zu untersuchen und Tausende, wenn nicht noch mehr, ähnlicher Genome gleichzeitig zu analysieren. Herkömmlicherweise wurde dieser Vergleich durch das Vergleichen einiger weniger ähnlicher Genome durchgeführt, wobei eines als Referenz für alle diente und dem direkten Vergleich mit anderen Genomen vorausging. Der Vergleich der großen Menge an erzeugten Daten mit nur einem einzigen Referenzgenom ignoriert jedoch einen relevanten Teil der verfügbaren Diversität.
Um die Einschränkungen zu überwinden, die sich aus der Verwendung eines einzelnen Referenzgenoms ergeben, wird in dieser Arbeit eine Methode vorgeschlagen, die einen Vergleich mit mehreren qualitativ hochwertigen Referenzgenomen gleichzeitig ermöglicht. Zu diesem Zweck wurde ein Algorithmus entwickelt, der einen Graph als Multi-Genom-Referenz erstellt, um den Referenzbias zu entfernen und die Downstream-Analyse zu vereinfachen.

Einzelne spezifische Markov-Kettenmodelle wurden auf allen untersuchten Genomsequenzen sowie ihrer berechneten lokalen paarweisen Alignments trainiert. Dies ermöglicht dem Algorithmus, die Struktur der Daten zu erfassen und Genome, die eine Reihe von Unterschieden aufweisen können, zu vergleichen. Eines der Ziele des Algorithmus ist es, ähnliche Regionen innerhalb sowie zwischen DNA-Sequenzen zu clustern und für jedes Cluster einen Repräsentanten zurückzugeben. Diese Repräsentanten werden später als Knoten im Graphen verwendet. Die Verwendung des Repräsentanten eines jeden Clusters anstelle aller Mitglieder eines Clusters entfernt unwesentliche Variationen durch Zusammenfassung der orthologen[1] und paralogen[2] Regionen, wodurch die Darstellung mehrerer Genome vereinfacht und eine merkliche Menge an Speicherplatz gespart wird Der erstellte Graph stellt alle Daten dar, indem die Shannon-Informationen minimiert werden. Daher muss kein enziger Parameter angepasst werden. Um die Leistung der trainierten Markov-Kettenmodelle zu evaluieren, wurde außerdem ein Werkzeug entwickelt um DNA-Sequenzen zu komprimieren. Es schätzt die Menge an geteilten Informationen zwischen Genomen und ermöglicht einen globalen Genomvergleich.

DNA-Sequenzierungstechnologien erzeugen fragmentierte Sequenzen, sogenannte Sequenzierreads, mit unterschiedlichen Längen. Diese Sequenzierreads werden dann zusammengefügt, um ein Genom zu generieren. Während es mit ausreichenden Reads möglich ist, ganze Genome de novo zu assemblieren, ermöglicht die Verfügbarkeit von genügend ähnlichen Genomen die Genomzusammensetzung durch Mappen gegen eine Referenz. Moderne Technologien sind nicht in der Lage, relativ lange Fragmente, die lange Sequenzreads genannt werden, mit geringen Kosten herzustellen. Ich gehe davon aus, dass in Zukunft, selbst mit sehr

[1]Regionen mit einer gemeinsamen Abstammung als Ergebnis eines Speziationsereignisses.

[2]Regionen mit einer gemeinsamen Abstammung als Ergebnis eines Duplizierungsereignisses.

billigen langen Reads, nicht jedes Genom de novo zusammengesetzt wird, sobald genügend gut zusammengesetzte Genome zur Verfügung stehen. Um Unterschiede zu verfügbaren Genomen zu identifizieren, sollten lange Reads mit einem Genomgraphen als Referenz verglichen werden. Obwohl sich mehrere Studien auf das Erstellen solcher Graphen konzentriert haben, ist das Mapping gegen einen Graphen und das Finden einer optimalen Übereinstimmung zwischen einem Pfad in einem Graphen und einem Sequenzierread immer noch eine Herausforderung. Daher habe ich auch einen Mapping-Algorithmus entwickelt, um den Pfad in einem Graphen zu suchen, der am besten zu einer Sequenz passt. Die vorgeschlagene Methode ist unabhängig von der Länge eines Sequenzierreads. Es berücksichtigt sehr ähnliche Regionen zwischen einem Sequenzierread und einem Genomgraphen und verwendet sie als Seeds. Indem die Pfade zwischen den Seeds durchlaufen werden, wird ein Alignment-Graph erzeugt und der Pfad in diesem Graphen mit der größten Übereinstimmung mit dem Read wird als Mapping-Ergebnis gewählt. Um diesen Weg zu finden, werden Markov-Kettenmodelle auf den Sequenzinhalten des Genomgraphen, der Reads sowie der Seeds trainiert. Die Modellparameter werden dann angewendet, um die Pfade in dem Alignmentgraphen zu gewichten. Die vorgeschlagenen Algorithmen liefern bei relativ kleinen Datensätzen effektive Ergebnisse und halten die Option für zukünftige Verbesserungen offen. In den nächsten Kapiteln wird das Ergebnis des Algorithmus angwandt auf mehrere *E.coli*- und Hefe-Genome sowie auf ein Chromosom des *A.-thaliana*-Genoms vorgestellt.
Ich glaube, dass die vorgestellten Algorithmen einen Fortschritt in der Analyse der Sequenzdaten darstellen und jenen Wissenschaftlern einen Weg aufzeigen werden, die daran interessiert sind, eine Datenstruktur aufzubauen, die mehrere Genome zusammen darstellt.

Abstract

The genetic material of an organism is stored in its DNA sequence. DNA sequencing technologies attempt to read pieces of this information that are subsequently put together to regenerate complete DNA complements. Astonishing advances have been made in these technologies over the last twenty years. These advanced methods have resulted in vast quantities of sequence data with extreme diversity and have made resequencing of an entire genome possible. Therefore, many large-scale genome projects now aim to mine this data and analyse thousands, if not more, similar genomes together. Conventionally, this comparison has been done by comparing a few similar geno-mes whereby one served as a reference for all, preceding direct comparisons of other genomes. However, comparing a large number of generated data against only a single reference genome misses a relevant portion of available diversity.

To overcome the limits imposed by using a single reference genome, in this dissertation a method is proposed that allows a comparison against several high-quality reference genomes simultaneously. To this end, an algorithm has been developed that creates a graph as a *multi-genome* reference to remove the reference bias and to simplify downstream analysis.

Individual specific Markov chain models are used to train over the entire set of genome sequences studied as well as its obtained local pairwise alignments. It enables the algorithm to capture the structure of data and to compare genomes with a range of

differences together. One of the goals of the algorithm is clustering similar regions within and between DNA sequences and returning a representative for each cluster. These representatives are used later as vertices on the graph. Using the representative of each cluster rather than all its members results in removing unremarkable variations by collapsing the orthologous[3] and paralogous[4] regions, thus simplifying the representation of several genomes and saving a noticeable amount of memory storage. The built graph represents all the data by minimizing the Shannon information. Therefore, there is no need to adjust any arbitrary parameter. Furthermore, to evaluate the performance of the trained Markov chain models, a DNA sequence compression tool has been developed. It estimates the amount of shared information between genomes and allows for global genome comparison. DNA sequencing technologies generate fragmented sequences, called sequencing reads, with different lengths. The sequencing reads are then assembled together to regenerate a genome. While with sufficient reads it is possible to *de novo* assemble entire genomes, the availability of sufficient similar genomes enables genome assembly by mapping against a reference. Modern technologies are incapable of producing relatively long fragments, called long sequencing reads, with a low cost. I anticipate that in the future, even with very cheap long reads, not every genome will be assembled *de novo* once enough numbers of well assembled genomes are available. To identify differences from available assemblies long reads should be mapped against a genome graph as a reference. Although several studies have focused on creating such graphs, mapping against a graph and finding an optimal match between a path on a graph and a sequencing read is still a challenge. Therefore, I have also designed a mapping algorithm to search for the path on the graph that fits a sequencing read the best. The proposed method is independent of the length

[3]Regions with shared ancestry as a result of a speciation event.

[4]Regions with shared ancestry as a result of a duplication event.

of sequencing reads. It takes highly similar regions between a read and a genome graph into account and uses them as seeds. By traversing the paths between seeds, an alignment graph is generated and a path with the greatest fit to the read on this graph is chosen as the mapping result. To find this path, Markov chain models are trained on the sequence content of the genome graph, reads and seeds. The model parameters are then applied to weigh the paths on the alignment graph.
The proposed algorithms return effective results on relatively small data sets and keep the option for future improvement open. During the next chapters the result obtained from running the algorithm over several *E.coli* genomes, Yeast genomes and chromosome one of *A.thaliana* genomes will be presented. I believe that the introduced algorithms will offer a step forward in analysing the sequence data and will illuminate a path for those scientists with an interest in building a data structure to represent multiple genomes together.

Contents

Chapter 1

Introduction

1.1 Genomes

Genomes are the full set of genetic information of an individual that has been inherited from both parents. They contain both coding regions (genes) and non-coding regions. Each organism may have several chromosomes that carry its genome. Chromosomes consist of a long Deoxyribonucleic Acid (DNA) string. Each cell has a full set of the genetic information that is mostly saved in the cell nucleus (or nucleoid for Prokaryotes). A small part of it is saved in mitochondria (and chloroplasts in plants). Genes mostly encode proteins that determine an organism's functions. Thus, the difference between genomes (different genotypes) can change the level of gene expression as well as protein structures, and thereby affect their functions as well as the organism's phenotype.
Genetic variation within species makes each individual different from the others. Different phenotypes and genotypes can be explained by studying their genetic variations. Variation can occur in the form of a single different nucleotide (SNP) or several ones, deletions of one or several nucleotides, insertion of nucleotides or even rearrangements.

1.2 Genome Sequencing

During the last two decades several DNA sequencing technologies have been introduced that facilitated the study of genome variations. Variations can be detected either by comparing several full length assembled genomes against each other, or by aligning sequence fragments (sequencing reads) against one or several reference genomes.

1.2.1 Technologies and Challenges

A pioneering technology that helped scientists in this matter was Sanger sequencing [55]. The main shortcoming of the Sanger technology was its sequencing price. It was relatively costly, for example the first *A.thaliana* was sequenced with this technology for about 70 million U.S. dollars. Moreover, the length of the read fragments sequenced with this technology are relatively short and even these days the length of the sequencing reads which can be investigated by this technology are not more than 1kb. On top of that, only a few sequencing reads can be obtained at a time using the Sanger technology.
The more recent sequencing technologies were introduced over a decade ago. They have reduced the time and the cost of sequencing DNA sequences drastically. As opposed to the Sanger sequencing technique, these technologies are capable of processing a large number of DNA molecules in parallel and they became known as Next Generation Sequencing (NGS) technologies [19]. One commonly available approach in the market was introduced by Illumina. All the Illumina platforms produce a vast amount of sequenced data, however the read length is shorter than sequences which are produced by the Sanger technique. The short length of the sequencing reads cause difficulties in assembling genomes especially when the sequence has a high number of rearrangement segments. To overcome the issue of short reads, some technologies such as SMRT sequencing and MinION have

been recently introduced to produce long sequencing reads. Long reads allow for the resolution of large structural features [19] and aid in finding structural variations and long repetitive regions. Although, these long-read technologies provide a large amount of sequenced data, their output has a high error rate. On one hand the high error rate of their outputs reveals the importance of applying a reference in assembling new genomes and on the other hand, having an extensive amount of data with a high level of diversity suggests that the reference needs to contain more information than only a single genome. In fact, the issue to address would be identifying variants based on comparing sequencing reads against several reference genomes.

1.3 Reference Genome

Reference genomes are ideally a single well assembled genome that is used as a coordinate system in studying the genomes of a species. They can be applied as a guide in assembling closely related genomes or detecting the variation between genomes of individuals.

1.3.1 History

Sequencing a genome from its sequencing reads can be done *de novo* where overlapping reads build the largest possible fragments of a genome known as a contig. This method is mainly applied when a reference genome is not provided. Another method used in sequencing a genome is mapping reads against the provided reference genome(s). A genome can then be resequenced after the mapping process. The first human reference genome was built with the Sanger sequencing technique in 2001 [26]. It was the largest genome to be extensively sequenced up to that point, twenty five times as large as any previously sequenced genome and eight times as large as the sum of all such genomes [26]. Afterwards, the attempt to make the full length genomes of other organisms with similar genome size began. In spite of the achievement, sequencing remained very time consuming and very costly. In a decade the number of sequenced genomes have increased exponentially and modern technologies have reduced the time and the cost of sequencing drastically. For example, between 2001 and 2017, both the time and cost of sequencing a human genome have fallen in tandem with each other with the former decreasing from 14 days to about 26 hours and the latter decreasing from 100 million U.S. dollars to 1000 U.S. dollars [65]. The decrease in cost and the time of sequencing resulted in obtaining many genomes of a multitude of species. The large number of generated genome sequences imposed a need for more advanced tools to mine and extract the information.

Using a single genome as the only reference imposes a bias in studying the variation between multiple genomes since each base of them is compared with only a single nucleotide on the reference. Since it is easier to map reference-like sequences than variant sequences, the variants can either be mapped on the wrong position or stay unmapped. This will then lead to a false variant call. Having a large number of genomes already sequenced suggests an alternative for a single reference genome. A replacement for it would use several reference genomes at the same time. It will aid in removing the bias against a single genome and increase the ability of detecting variations. Several studies (e.g. [23], [56], [31], [40] and [41]) have been done to address this problem. One way to deal with this issue is by using a single reference, containing chromosome length sequences and contigs, as the core of a new reference and attach the variations to this collection of sequences. As an example, in [23] a reference multi-genome is proposed which has been made by appending the variations from a set of genomes at the end of a single reference genome. Although the output reference multi-genome remains linear, it contains the information about the variations too and tackles the genomic variation observed at every position. Figure 1.1 taken from [23] shows a multi-genome reference and how it has been built.

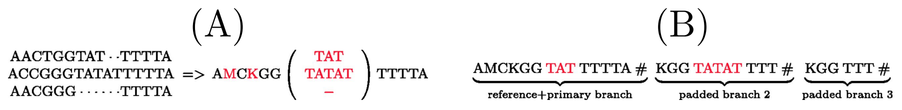


Figure 1.1: (A) Three genomes were superimposed, SNPs have been encoded with IUPAC characters (M, K) and finally indels have formed a bubble with 3 branches. (B) multi-genome reference have been built over the mentioned 3 genomes by considering one of the branches as the primary branch and appending the 2 others at the end of the reference. Each appended branch is padded at both ends with the bases surrounding the bubble. The length of the padding is a parameter that depends on the expected read length, which in this example is set to 4. [23]

An alternative approach would be creating a graph structure to present the variations in the format of different branches. One of the pioneering studies has been done by [56] where they have introduced the GenomeMapper, a tool to map sequencing reads against several genomes simultaneously. Their algorithm indexes the conserved and diverged regions. Identical regions are shown once while the diverged regions are shown as branches. This method provides us with more information than only a single reference, but at the end only a single genome is chosen as the best match for a sequence read. Thus, it is not able to detect sequences which are closer to the combination of several references. Figure 1.2 taken from [56] presents the graph generated by this method. Comparing this figure with the figure 1.1 reveals the differences between linear and non-linear references. The latter study is one of many genome studies where a graph is used to facilitate the analysis. Several of them will be described in the next section.

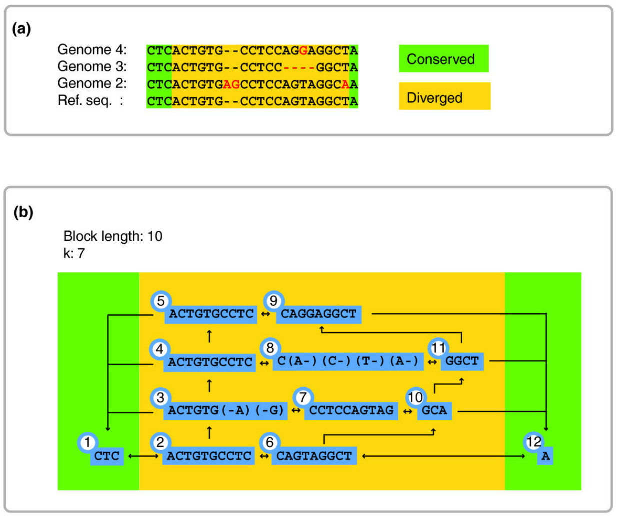


Figure 1.2: GenomeMapper's graph structure, (a) Orthologous sequences in four divergent genomes with shared fragment at their beginning and their end. (b) Graph structure created by the sequences in (a), with k-mer length 7, and maximal block length of 10 [56]

1.3.2 Graphs in Genome Analysis

Definition

In mathematical terminology, a graph is defined as an ordered pair $G(V, E)$ where V is a set of vertices (or nodes) and E is a set of edges. For each $v_1, v_2 \in V$, v_1 is connected to v_2 by an edge $e \in E$ if and only if there is a relation between them. A graph may carry some distinctions in addition to the above definition if necessary. For instance, it can be uni- or bi-directed when the direction of the relation matters. It might have cycles or being acyclic, have weighted edges or weighted vertices to present costs, lengths, sizes *etc.* Handling all these possibilities made graphs a useful tool in modelling data and representing the relations in a system. As a consequence, it has been widely used in different fields of study such as transportation, computer science, studying molecules, sociology, linguistics, biology, studying

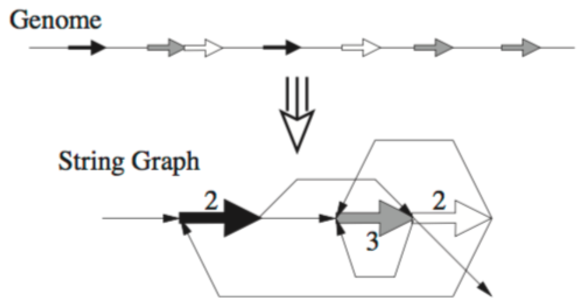


Figure 1.3: String graph. Coloured tick arrows represent repetitive regions on the genome and the numbers on the graph give the number of copies for each repeat.

protein networks and analysing genomes.

Variations

The concept of the string graph has been presented in [38]. It attempts to portray all that can be deduced with regards to a DNA sequence from a collection of shotgun sequencing reads acquired from it. The algorithm is a direct follow up of the *unitig* concept of the Celera assembler which was introduced by the same author in 1995 [37] and revolutionised genome assembly. The vertices on the string graph are fragments of sequences. The vertices are connected by edges so as to present their order on the genome. Figure 1.3 utilised from [38] shows the structure of such a graph.

Another type of graph which has been frequently used in studying genomes is the *de Bruijn* graph [9]. In a *de Bruijn* graph, vertices are sequences of symbols of length k and two vertices are connected by an edge if they have an overlap of length $k-1$. A generalization of the *de Bruijn* graph was later introduced in [50] to handle imperfect reads. This generalized graph is called *A-Bruijn* graph which is defined on an arbitrary set of alignments (A). Figure 1.4, taken from [50], shows its construction steps.

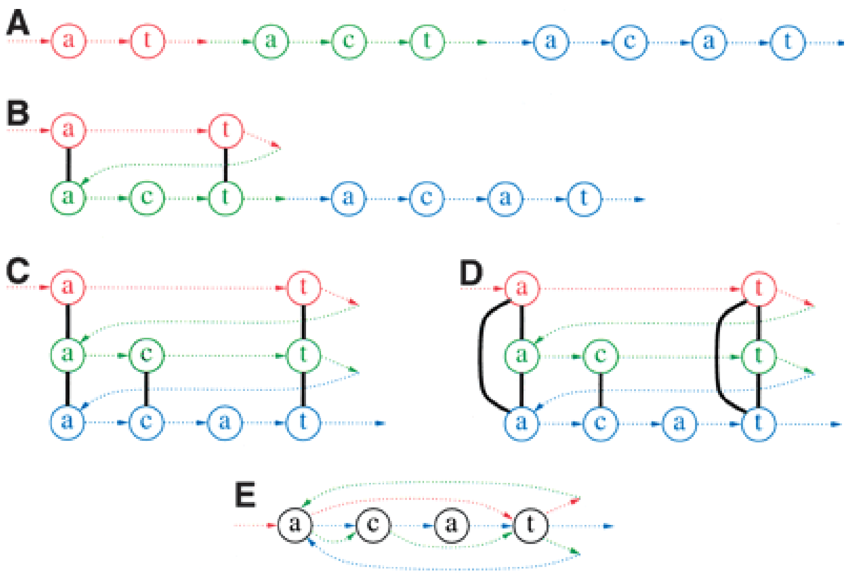


Figure 1.4: Construction of a A-Bruijn graph
(A) Construction of the A-graph from the sequence...*at*...*act*...*acat* by applying three pairwise alignments (B) $a - t$ versus *act*, (C) *act* versus *acat*, and (D) $a - t$ versus *acat*. (D) The A-graph consists of the eight nodes plus the seven thick, black edges created from the alignments; the coloured edges are shown to indicate the relation of the nodes to the sequence, but they are not part of the A-graph. (E) Each of these alignments serves as "gluing instructions" that transform the sequence into the A-Bruijn graph on four vertices; the coloured edges are in the A-Bruijn graph, although the colouring itself is not. [50]

Recently, the concept of the cactus graph which was first defined by [20] has been adapted by [46] for genome comparison. Like *A-Bruijn* graphs, the introduced Cactus graph by [46] is able to detect repeats, rearrangements and duplications, however on top of that, it can visualise the common substructures between several genomes as two-dimensional alignments and nets (see [46]). Nets will then be used as vertices on a Cactus graph. Figure 1.5, taken from [46], represents its general schema.

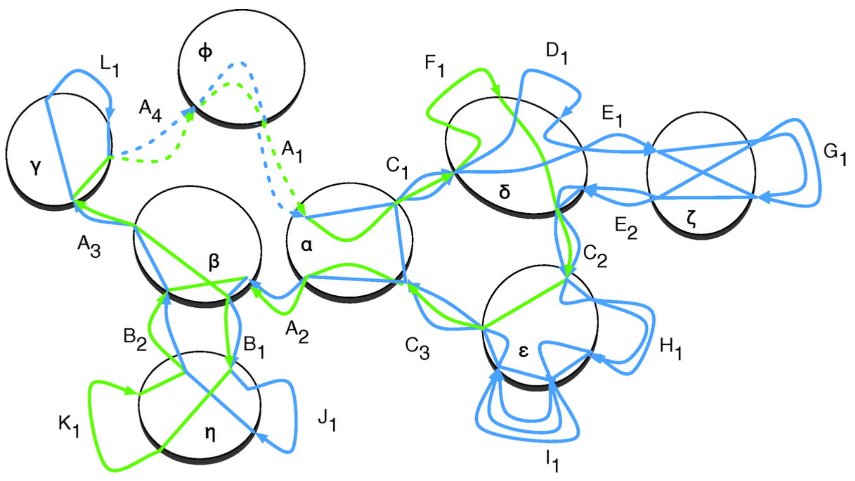


Figure 1.5: Cactus graph

Application

Graphs have been extensively applied in studying genomes. A very common application is in assembling genomes. In several studies, such as [51], [68], [60] and [4], a graph has been introduced to solve the assembling issue. In [51], using *de Bruijn* graphs, an algorithm for assembling DNA fragments was proposed (EULER). The approach was based on finding an Eulerian path[1] on a *de Bruijn* graph to avoid the drawback of traditional overlap based methods such as [37], [6] and [24]. For the first time, the algorithm was not looking for the overlapped pieces of reads to assemble them, hence it could unmask the repeated regions and use them to improve the quality of assembling. Later on, Velvet [68] was proposed as a series of algorithms to assemble genomes by taking advantge of *de Bruijn* graphs. As opposed to the EULER assembler, in Velvet vertices of the graph are not created by reads but they are only fragments of length k and reads are mapped on them through the existing paths. The choice of k

[1]Eulerian path is a path on the graph which visits each and every edge of a graph only once.

has a large affect on the assembly graph. Small k-mers result in shorter contigs with lots of connections, while large k-mers can result in longer contigs with fewer connections. [66] Thus, the choice of k is ultimately dependent on the genome in use. In a more recent study by [4], using a generalized *de Bruijn* graph (*A-Bruijn* graph) which has been introduced in [50], an assembler (SPAdes) has been introduced to generate assemblies of bacterial genomes from single-cell sequencing as well as conventional data sets. Original SPAdes came with the *proviso* that it might not work for larger genomes.
However, in [60] the idea of applying the FM index [2] in assembling genomes has been introduced. As a result of the algorithm, an assembly string graph is produced relatively faster than with the previous methods.
Graphs have been applied multiple times in aligning several sequences and creating multiple sequence alignments such as in [54], [69] and [47]. The concept of *A-Bruijn* graph which has been introduced in [50], is applied in [54] and alignments were represented as weighted directed graphs. As opposed to the linear aligner, the approach presented in [54] gives the user the opportunity of detecting duplications and recombinations. In [69], however, an Eulerian path approach has been presented to create local multiple sequence alignments. The proposed program enables the handling of a large number of sequences due to its low complexity and it is powerful in detecting conserved regions, thus returning accurate alignments. In [47] however, the cactucs graph as it is introduced in [46] has been used in aligning multiple genome sequences. Using Cactus graphs allows for the creation of a hierarchical tree structure. A multiple sequence alignment embedded within a Cactus graph can therefore be hierarchically subdivided into a tree of related but independent subproblems [47]. This method allows for efficient storage and

[2] A sub-string index system based on the Burrows-Wheeler transform [7] which has been created by [15].

random access, decomposition of a multiple sequence alignment into independent subproblems and revealing the general substructure of an alignment.
On top of the above cases, as it has been mentioned earlier in this section, graphs have also been used as a substitute for linear references in helping with detecting variations which is the main interest of this dissertation.

1.3.3 Graph-based References

One of the pioneering studies is the Genome Mapper proposed in [56], where a graph has been introduced as a reference to find the best match of seeds from a novel genome. In this study, an algorithm has been proposed to map short reads simultaneously against several reference genomes rather than only a single reference. The method is based on indexing the conserved and diverged regions of different references. An index graph is then created where the vertices represent the indexed regions and edges represent the occurrence of two regions, one after the other. After mapping short reads over the graph one of the references is chosen as the best match for the mapped reads. Although this method provides us with more information than mapping against a single reference, it is not able to detect the genome which could be closer to a collage of several references. Later on in [31] an algorithm has been introduced to solve the same issue as in [56]. By taking advantage of the *de Bruijn* graphs, they have proposed the de Bruijn Graph-based Aligner (deBGA) to solve the problem of aligning sequence reads to multiple reference genomes using the seed and extension algorithm. deBGA is relatively fast and it is sensitive in detecting the repeats with a high accuracy.
Genome graphs were not only used for mapping reads, but in many cases they have been used in representing the variation between several references. In [34] a compressed *de Bruijn* graph has been constructed from multiple references directly, without

constructing its corresponding uncompressed *de Bruijn* graph. This graph has been used as a pan-genome[3] to present the variation among population and aid in analysing genomes of multiple individuals together. In several studies such as [40] and [41], bidirected graphs were used to cope with the structure of double stranded DNA. Using a bidirected graph makes the presentation of both forward and reverse strand possible. In both [40] and [41], sequence graphs have been generated where vertices are labelled segments of DNA sequences. In [40] a pan-genome reference was introduced. This pan-genome has been built, after detecting homologous blocks on several reference genomes, by ordering and orienting them in a way that maximizes its agreement with the original references. The graph was then used as reference for a population to detect the variation or read mapping. In [41] however, a generalization of the Positional Burrows-Wheeler Transform (see [12]) to genome graphs is presented (gPBWT). The created graph is a collapsed representation of a set of genomes which was then used to efficiently query subhaplotype[4] matches. In this method, a bidirected genome graph has been built from the reference genome by adding alternative alleles to it, each input haplotype is then presented as a walk on the graph. Similar to the latter, in [32], positional markers have been used to encode the genetic variation within a Burrows-Wheeler Transform. Using this, a compact representation of the genomes of a population has been built (PRG). The goal of the study is to find the closest mixture of available genomes to the genome under analysis. Given the sites where alternative genomes differ from the reference genome, PRG graph is built as an acyclic, directed graph (see the figure 1.6 (A) adopted from [32]). The graph is linearised (Linear PRG) by encoding the variations from the reference genome and adding them to the current reference (see

[3]Pan-genomes contain a core genome along with a collection of variation between several sequenced genomes of a species.

[4]Haplotype refers to a collection of alleles which are all inherited from a single parent.

figure 1.6 (B) adopted from [32]).

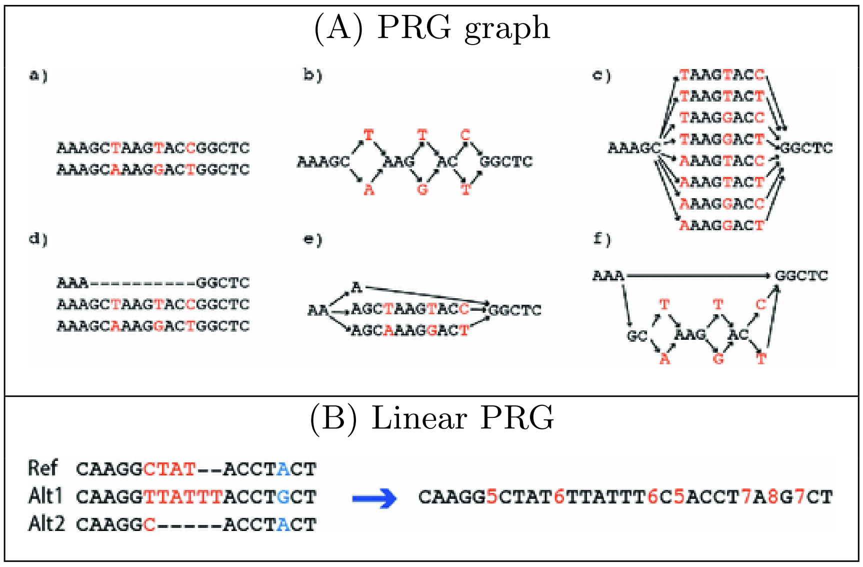


Figure 1.6: Population Reference Genome

To prove the ability of genome graphs in identifying new variations, in [42] genome graphs have been studied as an alternative for linear references to improve the ability of identifying variations and discarding the allele bias introduced by a single reference. They attempted to prove that adding the variants to the reference will improve the genome inferences. To reach this goal, several experiments have been done where in each of them, a method for graph construction has been used and the utility of each graph in read mapping and variant calling has been tested. On top of the above research, in some other studies such as [11] and [49], the main focus is defining sites on a graph and calling variants at the created sites. By combining several genomes and a collection of variations, a population graph has been introduced in [11] to aid in characterization of the MHC region with high sequence diversity. The method has been shown to improve the genome inference of a highly diverse region on the human genome (MHC) and identify the regions where the current references are incomplete. Ultrabubble has been defined in [49] as a generalization of Superbubble for bidirected graphs. Superbubble has been introduced in [45] as a motif to describe the sites on directed graphs. Superbubbles and Ultrabubbles are acyclic, directed subgraphs which are connected to the rest of the graph with only a node and there is only a single node on these motifs where an edge emerges out of them (a sink node). These motifs can be generated when new variations are added to the graph and they can pinpoint the relationships involving structural variants. Thus, they can be used as sites for variant calling.

In this dissertation, I am introducing an algorithm to overcome the limit imposed by using a single reference genome, I am developing a method that allows simultaneous comparison against multiple high-quality reference genomes. To this end, I am proposing an algorithm that creates a string graph, introduced in [38], as a multi-genome reference. It removes the bias against a single-genome reference and simplifies downstream analysis. To reduce the size and complexity of the reference, highly similar orthologous and paralogous regions are collapsed while more substantial differences are retained. This is in fact the major advantage of this method compared to the other existing approaches. As opposed to the mentioned methods, it generates clusters of homologous regions and facilitates the downstream analysis. Moreover, applying Shannon information (1.3.4) as the cost function[5] makes the algorithm independent of arbitrary parameters to adjust.

Furthermore, I developed a genome compression tool to evaluate the performance of the model. This compression tool can also be used for global genome comparison. In the next section, I briefly describe the algorithms I have used in building such a graph.

[5]The cost function is a function we desire to minimize in order to achieve the optimal solution. This concept will be discussed in more detail in 2.2.2.

1.3.4 General Terminology

Pairwise Alignment

Pairwise alignments are used to present the similarity of two pieces of biological sequences. The captured similarity may reveal some common characteristics of sequences such as homologous regions. To make the decision about the resemblance of two pieces of sequences, a scoring system is used. A common scoring system in creating alignments is a substitution matrix. Each member of a substitution matrix presents the probability of changing a nucletide (or an amino acid) to another nucleotide (or amino acid). BLOSUM [21] and PAM [8] are two known substitution matrices. Alignments can be created by aligning the full length sequences against each other (Global alignments) or only against the best local matches which can be found between sequences (Local alignments). A popular global alignment algorithm is the Needleman–Wunsch algorithm [39] where a dynamic programming approach is used to build a global alignment. This method is described in more detail later on in this chapter. However, a known local alignment algorithm is the Smith–Waterman algorithm [62]. Similar to the Needleman–Wunsch algorithm, the Smith–Waterman algorithm is also a dynamic programming algorithm, however it only captures optimal local alignments.
In this dissertation, local pairwise alignments are used as the first step to acquire the similarity of DNA sequences. In most of the experiment cases I have used a software called LAST [25] to create the local pairwise alignments, however my program is independent of the choice of aligner.

Markov Chain Model

In this work, Markov chain models are trained in order to mine the characteristics of sequence data. Markov chain models are based on the Markov property which is a known property in statistics and probability theory and is formulated as follows.

$$P(X|X_n, ..., X_0) = P(X|X_n)$$

It shows that given the present event, X_n, the probability of the future event is not dependent on the past events, $X_0, ..., X_{n-1}$. Markov chain models present a general form of probabilistic models for sequences. They can reveal some internal characteristic of the sequence since the probability of each character on a sequence is dependent on its previous characters. A classical Markov chain model can be graphically shown as a collection of states [13] and the probability of a transition from state j to $j + 1$ is obtained by

$$P(x_{j+1}|x_j)$$

Figure 1.7 adapted from [13] shows different states and transitions of a Markov chain on a DNA sequence. Markov chain

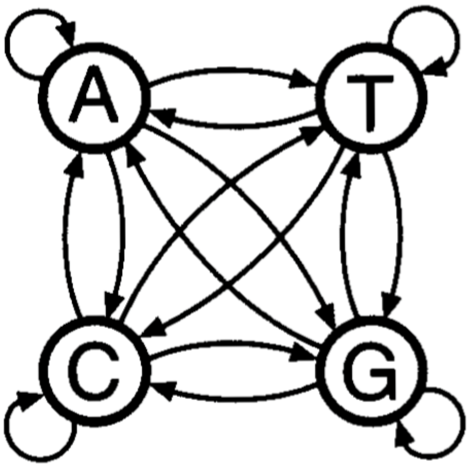


Figure 1.7: General overview of a Markov chain for DNA in its most simplified form

models can have different 'orders'. Orders specify the number of predecessor states which should be memorized by the model.

The classical one as it is shown above has the order 1. However, one has to keep in mind that the model complexity increases exponentially by increasing the orders.
Here, multiple Markov chain models with several orders are trained to get the property of both DNA sequences and created pairwise alignments.

Information Theory

The concept of information theory has been introduced in a ground-breaking paper by Shannon [58], in the case of communication over a noisy channel, where he quantified the 'information' for the first time. This theory has since been widely used in different fields of studies such as data compression. Information theory and quantifying the information is based on probability theory. One important concept introduced in information theory is 'entropy'. In [58], entropy has been defined as

$$H = -\sum_{j} p_j log_2(p_j)$$

Where H is the probability mass function and p_j is the probability of j happening. The introduced algorithm in this study is based on minimizing the amount of information and Shannon information is used as the cost function (More details can be found in 2.2.2).

Affinity Propagation Clustering Algorithm

The well-known $k - centre$ clustering algorithm [33] is based on an initial set of k randomly chosen data points. It runs many times to find the best initial set. Thus, it works well only when k is small enough and the chance of reaching a good initial set is high. On the other hand, a different approach was proposed by [17], called Affinity Propagation Clustering, where all the data points are initially considered as a potential centre. This

algorithm has been devised to cluster data points by passing messages between them. It clusters similar points together and returns a subset of representatives from the input data. To this end, similarity between pairs of data points are measured, then messages are sent between them until a high quality set of similar points (a cluster) emerges. In [17], the introduced algorithm has been applied on a broad range of data to prove the low error rate of the method.
Here, I use this algorithm to find clusters of similar segments of DNA sequences, using the model parameters as a metric to measure the similarity between sequences. This method is chosen over the $k - centre$ clustering algorithm for the following reasons. Firstly, because there is no need to set an initial number of clusters, secondly, the method guarantees the best solution and finally, it depicts a centre for each cluster.

k-edge-connected Graph

A connected graph is called k-edge-connected, if it stays connected after removing 0 to $k - 1$ edges. Figure 1.8 shows a 3-edge-connected graph.

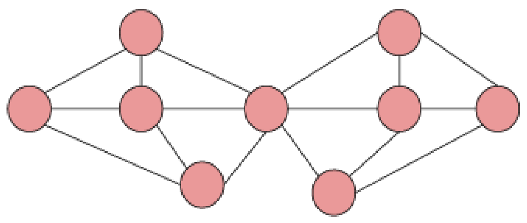


Figure 1.8: 3-edge-connected graph

In this dissertation, this concept is used to present the strongly overlapped pairwise alignments. $k = 2$ is chosen to present the alignments which are overlapped with at least 2 other alignments.

Suffix Tree

This data structure was first introduced in [64]. A suffix tree presents all the suffixes of a string of characters. Figure 1.9 [14] shows a suffix tree with all the suffixes of string $S = 121112212221$. Suffix trees have been widely used in computational biology for

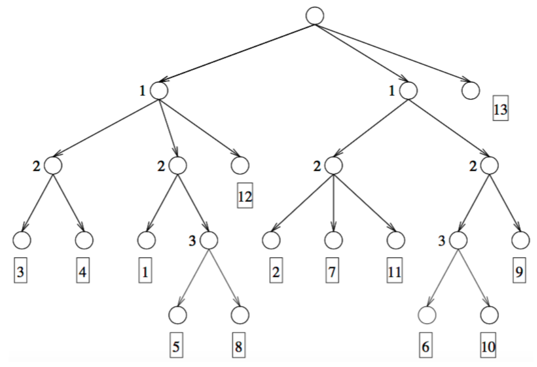


Figure 1.9: Suffix tree of string $S = 121112212221$

pattern matching, site recognition, making alignments *etc.* [3]. Here, I am using this tree structure to connect clusters' representatives if they happen frequently in the same order. For this purpose, strings of representatives' indices are generated and a suffix tree will be built over them including all the suffixes of available strings.

Arithmetic Encoding

Arithmetic encoding is an encoding technique which is used in lossless compression of a string by specifying a certain number of bits to each character on the string. It specifies more bits for the rare characters and fewer bits for the more frequent ones. The optimal number of bits for each character is calculated as $-log_2(P_c)$ where P_c is the probability of the character c happening. These probabilities are considered as binary intervals during the encoding procedure. Going through the string, in each step

the probability interval of the current character is chosen from the interval of the previous character until the end of the string is reached. As soon as the end of the string is reached, a value from the last interval is chosen to present the string. The encoded string can then be decoded by reading the binary interval of each character back.
Having the information obtained from the created graph, arithmetic encoding is used in this dissertation to compress the entire input sequences.
As has already been mentioned, mapping sequence reads can be considered as an application for a genome graph. Thus, an algorithm to map reads against the graph will also be proposed in this dissertation to handle mapping long reads against the generated graph.

1.4 Mapping Reads Against a Reference

Genome mappers were primarily built to map reads on a linear reference genome in order to determine where a new genome differs from an existing reference genome. A very well-known tool which has been implemented for this purpose is the Burrow-Wheeler Alignment (BWA) tool [27]. It has been designed to efficiently align short sequencing reads against a reference sequence, allowing mismatches and gaps. The method is based on the Burrow-Wheeler Transform (BWT) and backward searches for inexact matching.
As has been mentioned in the previous section, mapping sequencing reads is an occasion where graphs have been used in analysing NGS data and [56] has been named as one of the pioneering studies where a graph data structure has been used for mapping sequencing reads. Recently, with the rise of new sequencing technologies other studies have also focused on solving the mapping issue and tried to resolve the problem of mapping reads against a genome graph. As an example, the concept of *reference structure*

has been introduced in [48] as a collection of references, presented as a string graph, and a mapping scheme for each position on a string. Since different mapping approaches may lead to different mapping outcomes, this study proposes a solution to deal with the lack of standard mapping. Moreover, by using a graph as a reference it aids in handling more variation. The method provides a unique mapping result for a string. Later in [43], *Context Schemes* is introduced as a method for mapping reads of various lengths over different type of references, including a graph, unambiguously. *Context Schemes* only returns results when there is a unique best mapping, with this criterion uniformly defined for all reference bases [43]. To be able to map long reads as well as to detect the known variation among a population, a read mapper is proposed in [53] (VITRAM) which takes the length of sequencing reads and type of references into account. The algorithm is based on *Min-Hashing* the sub-strings of length q of several references. These sub-strings have been created for each window on a reference. Window length is slightly larger than the length of sequence reads. In some cases the obtained results from this method shows the higher precision for VITRAM than BWA.

To help in assembling genomes by mapping reads against a graph a heuristic algorithm has been presented in [28] to map sequencing reads over different paths of a *de Bruijn* graph. The pipeline is able to efficiently map millions of reads per CPU hour on a *de Bruijn* graph.

To allow for mapping reads directly on a directed cyclic graph, an alignment algorithm (V-ALIGN) has been proposed in [63] which aligns the input sequences directly without creating acyclic directed subgraphs.

Here, a mapping method is proposed for mapping long sequencing reads against a genome graph by looking for a path on a graph as the best match to a read. The following section introduces some algorithms which have been applied for the mapping pipeline.

1.4.1 General Terminology

Needleman-Wunsch Algorithm

This method has already been mentioned in 1.3.4 as an algorithm to create global pairwise alignments. The Needleman-Wunsch algorithm [39] is a dynamic programming method to compare biological sequences. It maximizes the similarity between input sequences using a given similarity matrix.
In this study this algorithm is used to build pairwise alignments between a part of a read and a vertex on a graph. The model parameters are used to present the similarities between sequences.

Dijkstra Algorithm

The Dijkstra algorithm is used in the proposed pipeline of this study to help in finding the best path on the reference graph that fits a read. This algorithm was first proposed by Dijkstra [10] to find the shortest path between two nodes on a graph. The performance cost of the original algorithm is $O(|V|^2)$ where $|V|$ is the number of nodes. However, using this idea, several optimizations have later been proposed to improve the Dijkstra algorithm performance. One of the optimizations that is used in this thesis utilises a 'Fibonacci heap' [16] instead of a queue. Using a Fibonacci heap as a priority queue decreases the running time of the Dijkstra algorithm to $O(|E| + |V| \log |V|)$ where $|E|$ is the number of edges. A Fibonacci heap is a collection of heap ordered trees where a child node key value is always bigger than or equal to its parent key value and the root of the tree has the lowest key value. The Fibonacci heap supports arbitrary deletion from an n-item heap in O(log n) amortized time and all other standard heap operations in O(1) amortized time [16].

Chapter 2

Multi-genome Reference

In this chapter, a method is proposed to create a directed graph which can be used as a multi-genome reference. The graph presents a simplified representation of several full length genomes by collapsing similar regions of them. To this end, Shannon information is used as the cost function and all the similar regions which are typically orthologous, that is, recently diverged regions, and more rarely paralogous, more diverged, regions, between and within genomes are then detected and clustered together. Only the representative member of each cluster is then used as a vertex on the graph. Two vertices are connected by an edge when they occur one after the other on at least one of the input genomes. Figure 2.1 illustrates the general schema of the algorithm. For simplicity, five short sequences were chosen to reveal all the steps of the algorithm.
The designed tool to build such a genome graph is described in the following sections of this chapter.
The software has been written in C++ and is available at `https://github.com/LeilyR/Multi-genome-Reference.git`.

2.1 Contribution

The Algorithm presented in this chapter has been designed by Jonas Müller and I. He also provided assistance in writing some part of the code.

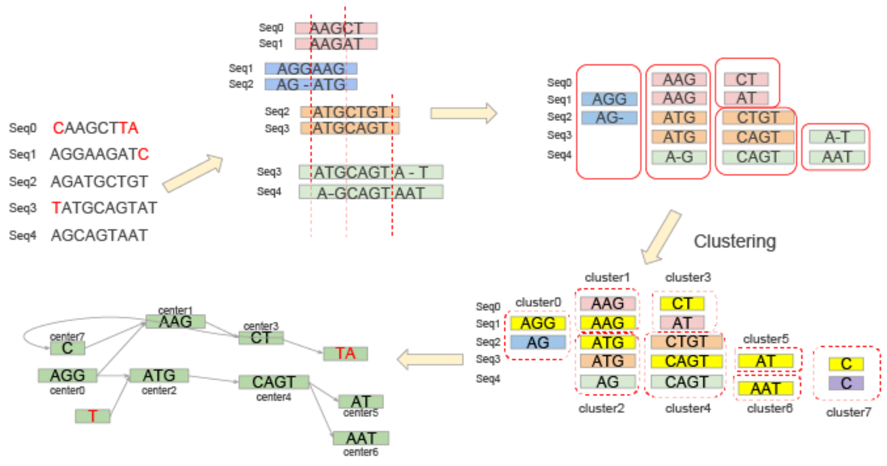


Figure 2.1: General overview of the algorithm
Step one: pairwise alignments are detected. Step two: Alignments are divided into non-overlapped segments. Step three: Related sequences are grouped into different clusters. Step four: centres of clusters are taken to be vertices of the graph.

2.2 Method

2.2.1 Input data

Preparing input files

To start working with this tool, input data should be saved in the accepted format for the program. The first step is creating a single FASTA file containing all the input genomes, and assigning a unique accession name to each of the genomes. The FASTA file format is a text based format to show DNA or protein sequences which was first introduced in [30]. Running the corresponding

command merges several FASTA files including different genome sequences into a single FASTA file and assigns a unique accession name to each of the input genomes. The original files names are chosen as the accession name of each input sequence. Accession names are then added at the beginning of their corresponding sequences' names. Knowing the accession names later aids in training accession specific models to optimally detect the characteristic of each genome.
As has already been mentioned, the program looks for similar regions on sequences and collapses them into a cluster. Thus, having the prepared FASTA file, all the similar regions between each two genomes and within a genome need to be detected. For that purpose, local pairwise alignments are generated. The program can read a Multiple Alignment Format [1] (MAF) file containing the alignments' information. Although the choice of the aligner is at the discretion of the user, in my experiments I mainly used LAST [25]. A MAF file includes all pairwise alignments needed as an input for running the tool along with the prepared FASTA file including all the genome sequences.

Reading input files

The FASTA file is read, and the name, accession and content of each sequence are saved in different data structures and an index is given to each of the genomes as an ID.
Afterwards, the MAF file is read and its information is saved for further usage. Each pairwise alignment is considered as an object for the alignment class where each object is constructed by its references' ID, coordinates and the encoded content on each of the references. To encode the content, each base is encoded by 3 bits ($A \to 000$, $C \to 100$, $T \to 010$, $G \to 110$, $other\ letters \to 001$, $.\ or\ - \to 101$). Reading the input data, several Markov chain models are trained over both genome sequences and the alignments to mine for their hidden characteristics.

2.2.2 Training Models

Markov chain models are widely used in the literature for segmenting genomic data ([67], [61], [22], [5] *etc.*). Here, a model has been designed which dynamically looks for the best order of a Markov chain model for each occurring base on a genome and each occurring modification between two genomes on a pairwise alignment.

On genome sequences

A maximum order ($S_{max} = 5$) is defined for training Markov chain models over DNA sequences, so the order cannot exceed S_{max}. On each sequence several models are trained to get the best order of the model (between 1 and S_{max}) for each base on a sequence. As previously stated, the method is based on minimizing Shannon information, therefore the cost of creating a base on a sequence is defined as

$$C_b = -log_2(P(b|B))$$

where $P(b|B)$ is the probability of base b occurring in a genome after a given context B. B is a string of previous bases on the genome with the length equal to the order of the model. The best order of a model is when this cost value is the lowest. The cost of creating a genome is then defined as

$$C_G = \sum_{b \in G} C_b$$

Where C_b is the cost of any base (b) on the genome (G).
While training the models on sequences, model parameters including a flag for visiting each sequence and the probability of each base occurring after a context of length S_{max} for that sequence is computed. This information is then written on a binary file. These parameters are later used in arithmetically decoding the entire input genome data.

To look for the best order, a list of all possible words of a given length l from 5 letters $(A, C, T, G\ and\ N)$ are created where l varies between 1 and S_{max}. They are saved on the format of a tree where the child node contexts are always 1 base longer than their parent nodes. By walking over each genome sequence and reading the occurring words, the probability of each base happening after any observed word is computed and the best one is chosen. For each genome sequence, the probability of all 5 bases after each word of length S_{max} is saved on the previously mentioned binary file as sequential intervals to be later used for arithmetically decoding the input genomes. Figure 2.2(A) shows how the model works for a certain order and how the cost values are calculated.

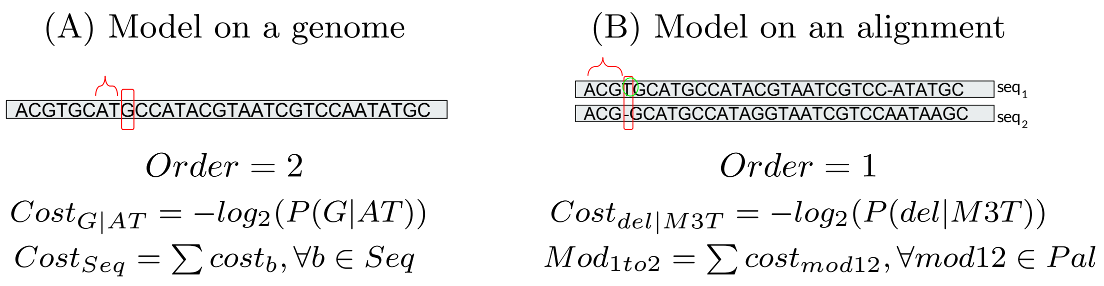

(A) Model on a genome
ACGTGCATGCCATACGTAATCGTCCAATATGC
$Order = 2$
$Cost_{G|AT} = -log_2(P(G|AT))$
$Cost_{Seq} = \sum cost_b, \forall b \in Seq$
(B) Model on an alignment
ACGTGCATGCCATACGTAATCGTCC-ATATGC
seq_1
ACG-GCATGCCATAGGTAATCGTCCAATAAGC
seq_2
$Order = 1$
$Cost_{del|M3T} = -log_2(P(del|M3T))$
$Mod_{1to2} = \sum cost_{mod12}, \forall mod12 \in Pal$

Figure 2.2: Simple illustration of how the models are trained
Red circles are presenting the current position, where the model is trained. The base in green circle is presenting the current base, where the modification between two sequences of the pairwise alignment is read.

On pairwise sequence alignments

Similar to what has been done for the input genomes, several Markov chain models are trained over pairwise sequence alignments. However, on the alignments we are dealing with modifications between two references, therefore there are more letters in the set of this alphabet than only 5 letters in the case of genomes in order to generate words from them. Thus, increasing the order of a model in this case can then lead to a very complex model.

To avoid this, maximum order $Al_{max} = 2$ is defined to generate modification patterns between genomes. All the modifications are encoded, and the encoded form of them are used to present the words.
To encode the patterns, each inserted and each modified base between two references of an alignment are encoded as an integer (5 integers for each case). However, deletions and matches between two references are binary encoded based on their lengths. A maximum length is defined for deleted bases ($2^{max_{del}}$) as well as matches ($2^{max_{match}}$). They were chosen in a way so as to be able to handle the pairwise alignments generated by an aligner. Since, on average, it is expected to see longer matches than deletions the chosen maximum length for matches is larger than the one that is set for deletion. Example 1 shows how deletions and matches are encoded.

Example 1 *Assume $max_{match} = 4$ and also assume that there is a match of length 10 between 2 references of an alignment. This match is binary encoded as $2^3 * 2^1$ and the powers of two will be used to present the match. However, if there will be a match of length 18 ($> 2^4$) it should be broken to a match of length 16 and a match of length 2 to avoid exceeding the max_{match} for the powers of 2. Each of the generated shorter matches then will be binary encoded.*

The encoded patterns are used as letters to create the words of different length between 1 and Al_{max}. Similar to the case of training models over genome sequences, a tree structure is used to detect the best level of models for each modification pattern. Figure 2.2(B) shows an example of how the models are trained. Walking over an alignment from beginning to end, the probability of a change happening after a word of a certain length along with the last occurred nucleotide base on that reference is computed. In fact, the cost of visiting a modification between two references on an alignment is calculated as

$$C_m = -log_2(P(m|M))$$

where $P(m|M)$ is the probability of the modification m happening between two references of an alignment, when context M is given. Context M is a word with a length between 1 and Al_{max} which was then extended by a single base that occurred immediately after this word. Since the model dynamically searches for the best order, the length of M is chosen in a way to make C_m the smallest.

At the end, two modification costs are defined for each alignment. Each of them presents the cost of changing one reference to the other to generate a pairwise alignment. Modification patterns which are observed after the words of length Al_{max}, as well as their costs, are written on the same mentioned binary output file. Modification costs of each of these patterns between each two accessions are saved as non overlapping intervals. On top of that, flags are assigned to present the occurrence of an alignment between two accessions. This information needs to be written on the same file so it can be used later for decoding genome sequences. Knowing these patterns and their cost gives us the opportunity of building an alignment when only the information of one of the references is given. One reference of an alignment is generated *de novo*. The second reference can be inferred by modifying the first.

After calculating the cost values, two modification costs are assigned for each alignment (M_{1to2}, M_{2to1}) which are the sum of the cost of all the occurring changes. On top of that, gain values are computed to present the value of each alignment. They reveal how much one gains by using the information of only one reference and creating the other one based on that. These values are obtained as follows

$$\begin{aligned} G_1 &= C_2 - M_{1to2} \\ G_2 &= C_1 - M_{2to1} \end{aligned} \tag{2.1}$$

To reduce the amount of information, the cost of observing an alignment between each pair of accessions is calculated and alignments with the average gain value less than this cost will get discarded at this step.

2.2.3 Homology Filtering

The main bottle neck in this method is eliminating the redundant regions from different alignments to avoid analysing each region on an input genome more than once. This step can aid in solving this issue by reducing the search space. Filtering is done by reweighing the alignments in a way that potential homologous regions may get a higher weight and are more likely to be chosen as well as be kept for the next step. This step can be considered as an optional step to decrease the running time by reducing the size of the data that feeds into the next step and it is highly recommended for the extremely redundant input data. To this end, all the remaining alignments from the previous step are reweighed and a new weight is assigned for each of them. They are then evaluated based on this new weight rather than their average gain value. As it is mentioned above, by computing the new weight, we are seeking to increase the value of potential homologous regions. Keeping these sequence segments as valuable (informative) parts of sequences and discarding the parts with a low weight will result in reducing the Shannon information.

For each alignment the program searches for the number of input sequences that confirm this alignment. The more such sequences, the higher the alignment's weight becomes. This is done in three steps. First, the highly overlapped alignments with the current alignment are found. Second, they are checked for a shared reference. Finally, the number of such shared references are counted and the gain value of the current alignment is updated based on this number. Figure 2.3 illustrates the above steps. Formula 2.2 presents reweighing computation where $Gain_1$ and $Gain_2$ are the gain values obtained from training models and $NumSeq$ is the number of sequences which confirm the current alignment.

$$AverageOldGain = (Gain_1 + Gain_2)/2,$$

$$UpdatedGain = AverageOldGain * (1 + NumSeq/2). \quad (2.2)$$

Alignments will then be sorted by their updated weight in a descending manner and only the first fraction of them with the higher weight are chosen to go to the next step.

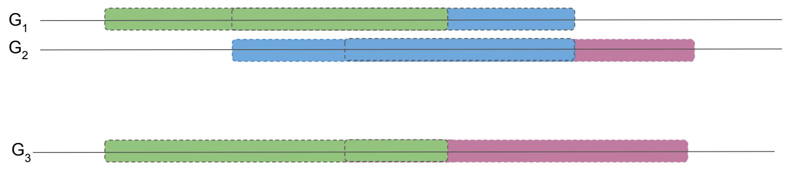


Figure 2.3: Searching for homologous regions

G_1, G_2 and G_3 are three input genomes. The blue alignment is highly overlapped with the pink and the green alignments. That these regions are typically contiguous in a genome is confirmed by the coloured region on G_3.

2.2.4 Creating Independency

Since partial redundancy between alignments results in analysing a single piece of sequence more than once, they are detected and eliminated after training the models. To this end, overlapped alignments are cut into shorter, non-overlapped pieces. One main obstacle in cutting alignments into shorter segments is that redundancy may get propagated between many alignments. Therefore, cutting overlap amidst two alignments can force multiple cuts in several other alignments at the same time and increase the complexity of the program exponentially.
To solve this issue, cutting is done recursively. At the first step,

an initial level of redundancy ($R\%$) is chosen. Although R can be varied, in our experiments we always set it to a number greater than or equal to 90, to capture as much redundancy as possible. All the alignments which are partially overlapped more than $R\%$ are detected and the overlaps between them are shown as edges on a graph where nodes are indices of the corresponding alignments. Going through all the alignments, the number of edges on the graph grow, and at the end the 2-edge connected sub-graphs of the graph are kept. Nodes (alignments) on these sub-graphs are cut into non-overlapped pieces and are saved into a container along with the alignments which were on the graph but were not part of any 2-edge connected sub-graph. Figure 2.4 illustrates this step.

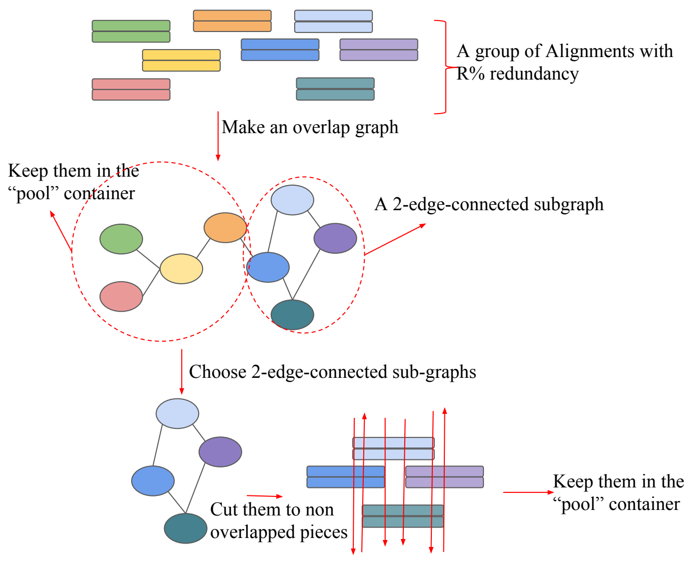

A group of Alignments with R% redundancy
Make an overlap graph
Keep them in the "pool" container
A 2-edge-connected subgraph
Choose 2-edge-connected sub-graphs
Cut them to non overlapped pieces
Keep them in the "pool" container

Figure 2.4: Cutting partially overlapped alignments with the overlap rate above $R\%$

For the rest of the alignments, those with an overlap rate less than $R\%$, go for the next step, and the same procedure will be repeated after decrementing R by 5. The obtained result from each step is saved on the same data container. Cutting is stopped when R is no longer above 50. All the alignments which are obtained from the recursive steps, are partitioned into groups of non-redundant alignments, therefore there is partial redundancy only within a partition but not between them. Partitioning aids in the analysis of several groups of alignments in parallel as well as minimizing the search space for the next step. On each partition, overlapped sites are detected and alignments are broken into smaller pieces from the two sides of the overlapped regions. To cut the alignments, first the split points on each reference of an alignment are detected and the gain values of the alignments after cutting are computed. If the values are larger than the base cost (the cost of making an alignment between two accessions) the current alignment will be cut from the detected split points otherwise cutting gets terminated. In fact, the cutting is stopped only when the resulted alignments become too short to be informative.

To cut the long alignments with many split points, all the above recursions still would not be sufficient to get the result in a reasonable time frame, so to make it more optimized in each of the above steps, if the number of redundant alignments are more than a fixed set number (Max_{cut}), they are partitioned into groups of alignments with less than or equal to Max_{cut} alignments in each group. The cutting step is then done on each of these groups separately. Finally, all the alignments obtained from cutting members of each group are then split into fully non-overlapped pieces, considering all the possible cutting points between them.

Once the partial redundancies are fully eliminated, related segments of sequences are grouped together in a way that there is no redundancy between different groups. However, alignments in a group share a full length sequence. The content of each group will then be checked for the level of similarities between its contents. Figure 2.5 shows these groups of related alignments.

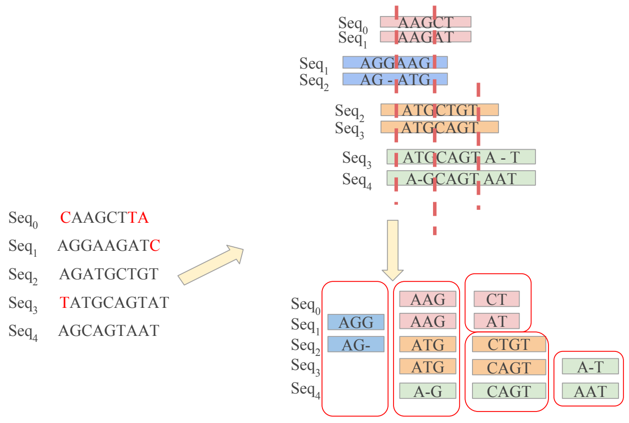


Figure 2.5: Removing redundancy, preparing groups of fully overlapped alignmnets

2.2.5 Clustering

To measure the amount of similarity within a group of related sequences obtained from cutting partially overlapped alignments, the program uses the Affinity propagation (AFP) clustering algorithm [17] to cluster similar pieces together. Running the clustering algorithm over independent groups of sequences prevents assigning a single piece of sequence more than once to several clusters, neither as a centre of a cluster nor as a member of it. Moreover, it allows for parallel analysis between different groups of related sequences.
The AFP clustering algorithm looks for the similarity between each two pieces of sequences by searching for the modification costs computed after training the models. Clusters are found by applying the formula 2.3 which takes the cost of creating each piece of sequence as well as the cost of modifying it to any other sequence into account. By finding the minimum value of 2.3, a representative (center) is assigned to each group of similar sequences.

$$Argmin_E(\sum_{i\in E} C_i + \sum_{j\in E, j\neq i} M_{ij}). \tag{2.3}$$

E in 2.3 represents a group of related sequences, C_i is the cost of creating sequence i and M_{ij} is the cost of modifying sequence i to sequence j. Applying formula 2.3 may result in several clusters for any group of related sequences.
The advantages of using AFP algorithm over $k - mean$ clustering algorithms is that there is no need for initializing the desired number of clusters, in fact the algorithm itself returns the best number of clusters by finding the $Argmin$ in formula 2.3.
Clustering sequences aids in detecting potential homologous (orthologous) regions. Knowing the centre of a cluster as a representative for a group of homologous regions helps in simplifying the presentation of input genomes. Having a single sequence as a representative for a group of similar sequences, there will be no need for keeping all the members of a cluster but rather only its representative and its modification patterns to each of the members. This will result in removing the non-significant variations and therefore simplifies genotyping and saves memory space.

2.2.6 Adding Non-clustered Regions

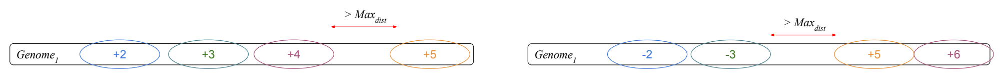
After the clustering step, a list of centres as well as their positions on a genome is created for each input genome. Knowing which positions on a genome are in one of the built clusters, the rest of the genome are considered as '$non - clustered$' regions. To keep the information of every single position on the genome, while building the genome graph, each $non - clustered$ region is also considered as a cluster with only a single piece of sequence as its centre (the representative piece). These centre-only clusters are then added to the same list of centres of input genomes along with their positions. This way, there will a representative for each region on a genome. If $non - clustered$ regions have the exact same content, they are clustered together and form a single cluster with a size bigger than one. So far, the regions which are not identical but similar enough to be clustered together have not been collapsed into a single cluster. However, this can be computed later using the costs obtained from the models. This will be an improvement for the graph construction and will lead to further minimization of information.

2.2.7 Merging Centres

Since the proposed method in this dissertation is based on reducing the amount of Shannon information, several centres can be merged into a single longer centre ($long - center$), if it results in reducing the amount of information. It can happen when a series of centres occur frequently on several input genomes one after the other in the same order. To check for such a situation, the gain of using a $long - center$ instead of several centres separately is calculated as follows:

$$G_{long} = N(add_0 + ... + add_n) - (N * add_{new} + add_0 + ... + add_n) \quad (2.4)$$

where N is the number of times that centres 0 to n happened after each other in the same order and $add_0, ..., add_n$ are their addressing values. In case of concatenation, each of the centres is addressed only once and after that each time only the $long - center$ will be addressed. Thus, merging will result in reducing the information if G_{long} is bigger than 0. This makes sense only when a cluster contains more sequences than a single centre. Therefore small clusters with the size equal to one are ignored in this step.
A maximum (Max_{dist}) is set for the allowed distance between the two closest centres on a genome. By walking over each of the input genomes, centres which are closer than Max_{dist} to their next neighbour are chosen and form a string. The length of the string increases until a distance longer than Max_{dist} is visited. Since the direction of each centre on a genome matters, centres are indexed on each genome considering their direction. If a centre occurs on a forward strand the index will be a positive integer, otherwise it will be a negative integer. Figure 2.6 shows the construction of such strings.



$String_1 = +2 + 3 + 4$

$String_2 = -2 - 3$
$String_3 = +5 + 6$

Figure 2.6: Creating strings as potential $long - centers$

After creating such strings from all the input genomes, a suffix tree is built including all the suffixes of these strings. By traversing the tree the number of visits for each branch is computed. These numbers show how many times each potential $long - center$ has happened on input genomes. The gain values are then computed using the formula 2.4 and the $long - center$ with the highest gain value is chosen. Afterwards, the suffix tree will be updated using the index of this $long - center$ instead of the indices of its corresponding centres. Gains are computed and the $long - center$ with the highest gain value is then chosen. The same procedure is repeated until there will be no more $long - center$ with the positive gain value. $Long - centers$ are then added to the list of centres for each input genome along with their position on that genome. Using the $long - centers$, the alignments between each centre and its members are also updated. These alignments may contain long gaps to compensate for the distance between original centres that the $long - center$ is made of.
This step can be considered as an optional step, users can decide to build the graph with or without $long - centers$. Making $long - centres$ may result in a more simplified data structure to present several genomes, however, it may decrease the mapping precision. Thus, it is considered as an application dependent step.

2.2.8 Multi-genome Reference

Now we have all the information necessary to build a graph as a *Multi-genome reference*. The proposed graph is nothing else but a presentation of ordered lists of centres, including $non-clustered$ regions and $long-centers$, of all the input genomes. Graph $G(V, E)$ is built where V is a set of centre indices which are used as vertices on G and E is a set of edges showing the order of centres on each of the input sequences. Figure 2.7 shows a graphical illustration of G made of two input genomes. The directions of the arrows determine the direction of DNA strands. On the figure '$-$' presents reverse and '$+$' presents forwards strands. The centres that are shown by dotted lines are $non-clustered$ regions. Two identical $non-clustered$ regions are on $Genome_1$ and $Genome_2$ thus they are clustered and form a single cluster (index $= 9$).

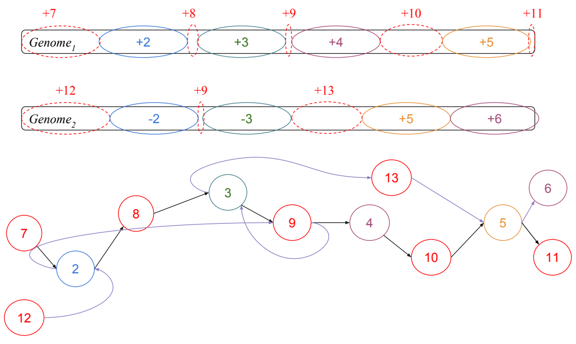


Figure 2.7: *Multi-genome reference*

To visualise the graph, the program offers a variety of outputs which can be chosen by users. An adjacency list is generated which presents the order of centres on each of the genomes. A DOT[1] [59] representation of the graph is then built which includes all these adjacencies. On this graph, each centre is presented with its index as an integer number bigger than or equal

[1]A descriptive language to present graphs

to 1. The direction of the connection between vertices is determined by a '+' or '−' symbol that is added to the index number. They show the presence of the centres on the forward or the reverse strand respectively. Figure 2.8 shows part of the DOT graph for the graph shown in Figure 2.7.

```
/Graph referenceGraph {
        7+ -> 2+ ;
        2+ -> 8+ ;
        8+ -> 3+ ;
        3+ -> 9+ ;
        12+ -> 2-;
        -2 -> 9+ ;
        9+ -> 3- ;
        ...
}
```

Figure 2.8: An example of a DOT graph

To have access to the content of centres and their coordinates, a FASTA [30] file can be produced that contains the content of each centre. The name of the sequences in this FASTA file are indices of the centres. Along with this file an additional file in the form of a plain text file is produced which includes the coordinates of all the centres. Figure 2.9 shows part of the FASTA and the text file which was created for the graph shown in Figure 2.7. Each line on the created plain text file contains the information of a centre separated by ':'. The first part is the index of the centre, the second part is the name of the input genome this centre was chosen from and the next two parts are the position of the centre on the input genome and the length of it.
Graphical Fragment Assembly (GFA) format has been proposed by [2] for representation of DNA sequence assembly graphs. Since then, it has been used by different tools to demonstrate the utility of genome graphs for a variety of purposes such as SNP calling, mapping and so on. To make the output of our algorithm compatible for other tools, a GFA file is created to reveal the structure of the built *Multi-genome reference* graph. This GFA file contains the index of each vertex and its content, the edge

```
>2
ACTCGTGGGCAATCGTAAC
>3
AATTCGTGAACATGCCATAAA
>4
AAACCTTTACATGGGCGCGAGTC
>5
CTGCGTGCATATAAGGCTGATGCAA
```

2:Genome$_1$:10:19
3:Genome$_1$:36:21
4:Genome$_1$:64:23
5:Genome$_2$:85:25

Figure 2.9: The left column shows a few lines of a FASTA file which includes centres' content and the right column shows its joint text file with centres' coordinates on the input genomes.

between each two vertices and a path for each input genome that represents all the centres on that genome in the order of occurrence. In the next chapter, I present the usage of this output for mapping sequence reads against the implemented graph.
Last but not least, the program has the ability of presenting the clusters in a MAF format and shows the content of each cluster as a multi-sequences alignment with the centre of the cluster being the first sequence on an alignment.

2.2.9 Compression

As has been mentioned before, the model parameters were already written on a binary file. This information includes:
1. The cost of visiting each modification between two accessions after a pattern of length Al_{max}.
2. The cost of observing a base on an accession after a string of length S_{max}.
3. Flags to demonstrate the positions where an alignment starts or ends on input genomes.
4. Flags to present reaching the end of a sequence and reaching the end of an accession.

In this step, the entire input sequences are arithmetically encoded using this information and the created graph.
First, all the centres need to be arithmetically encoded base by base. To this end, centres of clusters with at least one non-centre

member are weighed by counting the number of the members in their clusters. Weights are then used to assign a unique flag that represents each of these centres. For each centre, after writing its corresponding flag on the mentioned binary file, all its bases are arithmetically encoded. Centres are read base by base and the cost of creating each base obtained from the model is used to encode that base. The encoded centres are written in the same binary file.
Afterwards, the list of centres on each genome is used to detect the positions on the genome that fell into a cluster. By walking through a genome from the beginning to the end, each base is arithmetically encoded using the cost of creating it (obtained from the model) if the region is not in any of the weighed clusters. As soon as such a position is reached the set flag from the model that shows the presence of an alignment between two sequences, as well as the unique flag of the corresponding cluster, are encoded. Then all the modification patterns between the centre of the cluster and the current region on the genome are encoded using the computed modification costs from the model. At the end, the set flag for visiting the last position on an alignment is encoded. If the next position is again in a cluster the same procedure is repeated, otherwise the bases are encoded until the next such position or end of the sequence is reached. By reaching the end of a sequence and end of an accession their flags are also encoded. All the encoded values are written on the same file.
Comparing the size of the obtained file after encoding all the input sequences with size of the input FASTA file reveals the compression rate. This rate shows how good the model fits the data (see section 2.3 for the experimental results). If the patterns were captured well the rate should be high. Moreover, the obtained compression rate can be used as a metric to measure the distance between input genomes. The higher the rate, the more similar the input genomes are.

2.2.10 Decompression

To retrieve the input genomes, all the sequences can be decoded without losing any information. The encoded file obtained at the end of the previous step is the only requirement to decode all the genomes losslessly. By reading the file all the written parameters of the model are retrieved. Having this information, the encoded sequences are decoded. First, all the encoded centres are decoded and then all the genome sequences are decoded knowing all the flags, costs and content of centres.

2.3 Result

To assess the performance of the algorithm several data sets were chosen as input to run the program on and to study the result obtained from them. The pilot experiment was done on *E.coli* genomes with the length of about $5Mb$ per genome. Several input data were defined having a different number of *E.coli* genomes. The result from running the program on them has given an estimate of the performance of the method as well as its complexity. Input files were created with 2, 4, 8 and 16 *E.coli* genomes with the size of $10,140,102$ bases ($10Mb$), $19,456,051$ bases ($20Mb$), $39,923,568$ ($40Mb$) and $80,686,085$ bases ($80Mb$) respectively. For each of them a MAF file has been generated presenting all the local pairwise alignments between and within the input genomes. LAST [25] has been used for this purpose with its default setup. Table 2.1 presents the number of alignments and their average length in each case.
To check for the complexity of the method the relation between time and the number of alignments, normalized by their average length, were computed. Figure 2.10 demonstrates this relation. A jump is observed in Figure 2.10 (A) between the input with 4 and 8 genomes. The jump might be a result of the lack of similarity in the input with only two genomes. As it is shown in 2.1, the number of obtained alignments between 2 *E.coli* genomes is

Input	Total number of alignments	Between genomes alignments	Average length (base)
2 *E.coli* genomes	$38,155$	$17,722$	793
4 *E.coli* genomes	$111,200$	$82,068$	952
8 *E.coli* genomes	$247,660$	$223,282$	$1,500$
16 *E.coli* genomes	$536,637$	$520,865$	$2,514$

Table 2.1: Number of pairwise alignments, number of between genomes alignments and the average length of alignments for each of 4 different input data sets.

significantly lower than the other cases. Figure 2.10 (B) shows the same relation after omitting the corresponding point for the experiment with 2 *E.coli* genomes. The blue lines show the $y = x$ line and the green curve presents $y = x^2$ and the black line shows the $y = 11 * x$ line.
For each input, the size of the built graphs were compared after running the program with and without generating $long{-}centers$. Table 2.2 shows the number of vertices and edges for each experiment. Despite a slight increase in the number of vertices by creating the $long - centers$, the number of edges dropped significantly. This reduction in the number of edges results in simplifying the representation of input genomes.
To look for the simplicity of the data structure presented by the proposed method in this dissertation, the size of the built graph with this method has been compared to the graphs made by other tools. The variation graph (Vg) [18] and the $REVEAL$ [29] program were run to build a graph on the input FASTA file with two *E.coli* genomes. The number of vertices and edges of the built graph by each of them is shown in Table 2.3. Comparison between these numbers and what has been shown in Table

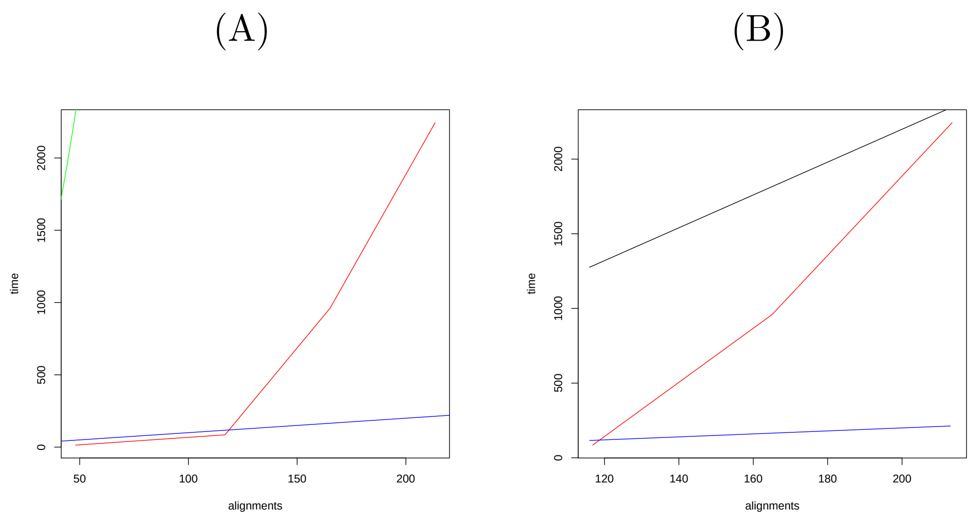


Figure 2.10: Relation between running time (minutes) and the number of input alignments normalized by their average length (number of alignments/average length) is shown in red. Blue line is $y = x$, green curve is $y = x^2$ and black line is $y = 11 * x$.

2.2 clearly demonstrate the simplification which has been made by our method in building a data structure to represent several genomes.
To appraise the capability of the algorithm in handling larger and more complex genomes, the next experiments have been designed on *S.cerevisiae* genomes with the length of about $12Mb$ per genome, and several chromosome number one of *A.thaliana* with the length of $30Mb$ per chromosome respectively.
In the first experiment a FASTA file containing four *S.cerevisiae* genomes has been used as input and the pairwise alignments were gained after running LAST on it. To avoid receiving alignments with a very low quality, the scoring option of LAST has been set to $-e120$. The size of the genomes in this input FASTA file is $46,150,487$ bases ($46Mb$) and the number of obtained alignments is $501,018$ with the average length of $275,830$ bases. Comparing these numbers with those observed in the previous experiment shows a substantial increase in the number and the length of

Input	Vertices	Edges	Vertices (with *long-center*)	Edges (with *long-center*)
2 *E.coli* genomes	$1,719$	$2,484$	$1,728$	$1,962$
4 *E.coli* genomes	$2,259$	$3,372$	$2,465$	$2,755$
8 *E.coli* genomes	$5,656$	$8,782$	$6,303$	$6,609$
16 *E.coli* genomes	$11,557$	$20,826$	$12,947$	$17,127$

Table 2.2: Graph size on *E.coli* genomes: number of edges and vertices after running the program with and without merging centres to $long-centers$

Input	Vertices (Vg)	Edges (Vg)	Vertices (REVEAL)	Edges (REVEAL)
2 *E.coli* genomes	$415,701$	$374,468$	$91,296$	$121,798$

Table 2.3: Size of the created graphs by Vg and REVEAL on 2 *E.coli* Genomes

the alignments as a result of increasing the size of the genome. Having obtained both MAF and FASTA files, the program was run with the option of not generating $long-centers$. The same setup for LAST has been used to create pairwise alignments on three chromosome number one of *A.thaliana*. The total length of the genomes in this input FASTA file is $89,515,425$ bases ($89Mb$) and the number of generated pairwise alignments is $621,355$ with the average length of $1,619$ bases. As in the previous case, the program was run without considering the $long-centers$. The computer that has been used to run the program on the last two experiments is an AMD Opteron(tm) machine with the CPU size of 2.4GHz. and 350G of RAM. The number of vertices and edges on the built *Multi-genome reference* graph for each of the above experiments are shown in the Table 2.4.

Input	Vertices	Edges
4 *S.cerevisiae* genomes	16,313	23,658
3 chr.1 of *A.thaliana*	10,061	15,534

Table 2.4: The number of vertices and edges on the *Multi-genome reference* graph

After running the program, the size of the encoded file obtained from each of the experiments was compared to the size of the original input FASTA file, this comparison revealed a high compression rate for all the cases. The compression rates gained from my algorithm were also compared with the compression rates acquired from the *bzip*2 [57] as a general data compressor and the *MFcompress* [52] as a genome specific compression tool. The performance of my tool is up to 8 fold better than *gzip*. Table 2.5 presents these values. All the mentioned values from my algorithm in this table were obtained after running the program without considering the $long-centers$. Since the *bzip*2 algorithm is a general data compression and does not take the sequence characteristics into account, therefore as a result the same rate is observed for almost all the cases (about 2 bit/base). Reaching a compression rate close to what has been generated from a cutting edge genome compression tool (*MFcompress*) reveals the power of our trained models in capturing the pattern on the sequences and alignments by taking the input genomes' intrinsic characteristics into account. In fact in the case of *A.thaliana* my algorithm performed even better than *MFcompress* which could be a result of better capturing the patterns after training the model on a larger set of data.
The information about the data which has been used for all the above experiments can be found in Appendix A.

Input	Binary file (size in byte)	Comp.Rate (*my tool*)	Comp.Rate (*bzip*2)	Comp.Rate (*MFcompress*)
2 *E.coli* genomes	$1,604,355$	1.25	2.3	1.35
4 *E.coli* genomes	$2,317,902$	0.93	2.3	0.83
8 *E.coli* genomes	$4,411,419$	0.87	2.3	0.65
16 *E.coli* genomes	$9,263,262$	0.91	2.3	0.52
4 *S.cerevisiae* genomes	$5,089,735$	0.83	2.3	0.81
3 chr.1 of *A.thaliana*	$10,455,467$	0.93	2.12	1.12

Table 2.5: Comparison between the compression rates (bit/base) obtained from our algorithm, *bzip*2 and *MFcompress*. The rates obtained from our algorithm were computed after running the program without looking for $long - centers$

2.4 Discussion

The proposed algorithm presents a robust representation of the input genome data by minimizing the Shannon information without the need of adjusting any arbitrary parameters. Comparing its output with the graphs generated from the other tools proves the simplicity of this representation. It can easily be seen in Table 2.3 where two other software generated a rather large graph while representing the same set of data. Another advantage of this method over the other programs is the return of clusters of sequences while generating the graph which can be used for within and between cluster analysis and aid in genotyping. Last but not least, my proposed method returns a compressed form of the entire genome sequences. The obtained compression rate can then be used as a metric to measure the similarity of several genomes. The more similar the genomes are the higher the compression rate becomes. However, on top of all the advantages my method has, the only drawback of the method is the expensive step of cutting partially overlapped alignments which leads to increasing the running time (2.10). To solve this issue a recursive cutting step has been proposed to decrease the complexity of the cutting step by reducing the number of alignments needed to be cut in each step.

Chapter 3

Mapping Sequencing Reads on a Graph

A challenge in having a graph as a *multli-genome reference* is mapping sequencing reads against it. Almost all the existing mapping methods are applied to acyclic graphs and many of the approaches are not able to handle long reads. Here a method is presented to map reads against a genome graph with no restriction on the type of the graph (cyclic or acyclic) and no limit on the length of the sequencing reads. It is able to look for variations between several genomes and the sequencing reads simultaneously. For each sequencing read the proposed algorithm looks for the best path on the graph and returns a patchwork of several genomes which fits the reads the best. Figure 3.1 presents its schema. All the steps will be described in detail in the following sections.
The software has been written in C++ and is available at `https://github.com/LeilyR/Mapping.git`.

3.1 Contribution

Jonas Müller and Marion Dubarry assisted in designing the algorithm presented in this chapter.

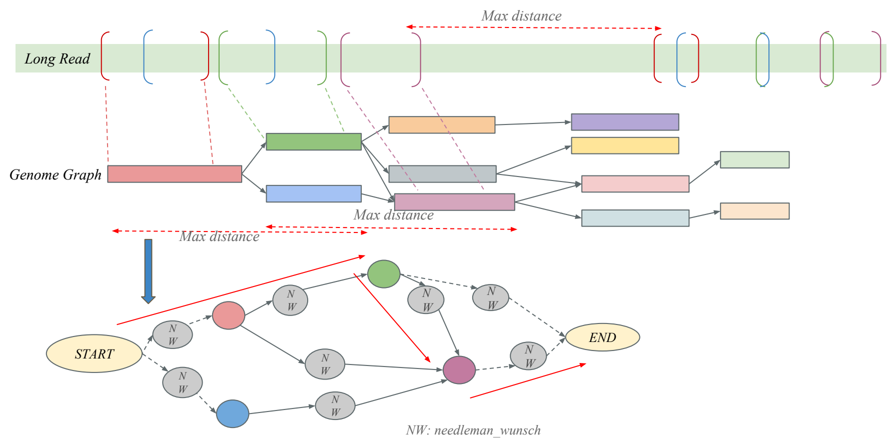


Figure 3.1: Alignments between a sequencing read and the graph are found, coloured parenthesis show the aligned regions. The read is partitioned into groups of alignments in a way that distance between each two groups of alignment is larger than a set *maximum distance*. Each group of alignments is processed separately and the best path on the graph which fits this part of the read is found. To find the best path, a weighted graph is built with vertices being alignments and edges being their order. Using the *Needlemann-Wunsch* algorithm, pairwise alignments are generated between the non-aligned regions on the read and non-aligned regions on the graph which occurred between two alignments. Finally, the shortest path is found on this graph.

3.2 Method

3.2.1 Input Data

To run the algorithm, a DOT graph or a GFA (Graphical Fragment Assembly Format [2]) graph is needed as an input along with a FASTA file, including the sequence content of vertices on the graph and the reads. Furthermore, a MAF file is required containing all the pairwise alignments between sequencing reads and the vertices on the graph.

Prepare input data

Similar to the previous chapter, accession specific Markov chain models train on the input data, thus the input FASTA file, including both reads and the graph information, needs to contain a unique accession ID. To prepare the required input data, first a unique accession ID is assigned to all the vertices from the graph and another ID is assigned to all the sequencing reads. Accessions are added at the beginning of the sequences' name in the prepared FASTA file which is later used as an input for running the program.
To generate a MAF file, including all the local pairwise alignments, any pairwise aligner algorithm can be used. However, one has to keep in mind that the name of sequences while generating the alignments should be the same as their name in the prepared FASTA file in the previous step. To reach this compatibility, two FASTA files, one including the reads and the other including only the graph vertices, should be created with the same assigned accession IDs. During the next steps, the recognised similar regions by the aligner are used to look for the best matches between the graph and each of the reads.

Read input data

Both FASTA and MAF files including all the sequences' content and all the alignments between them are then read by the program. The same pairwise alignment class as in previous chapter (2.2.1) is used to save the alignments information. As has been mentioned before, each alignment is distinguished by its sequences name, contents, and positions on the references. Despite using the same class to read the alignments, in this case alignments are always saved in a way that the forward strand of reads are seen. On top of that, several data containers will be filled in with the information obtained from the FASTA file

including the name, accession and content of sequences (both reads and vertices).
Afterwards, the given GFA or the DOT graph is read and a list of adjacencies will be saved. The direction of each pair of adjacent vertices are determined by the + or − read from the DOT or GFA file. The directions are taken into account while making the list of adjacencies. An integer index is assigned to each vertex. The vertices with the reverse direction will get an index of less than zero while a positive integer is assigned to the vertices with the forward direction.

3.2.2 Training Markov Chain Models

The similar model class as in section 2.2.2 is used here with a small alteration while training over the alignments. In the mapping case there are always two accessions, one belongs to the vertices of the reference graph and the other belongs to all the sequencing reads. So, the Markov chain models with different orders are first trained over all the sequences of these two accessions where each vertex as well as each read is considered a single sequence. The best order is chosen for each seen pattern of each accession. Afterwards, several models with different orders are dynamically trained over the alignments between vertices and reads. For each occurring modification between two sequences of a pairwise alignment the best order of the model is chosen which makes the cost of modifying a sequence to the other one the smallest. Then, for each alignment the cost of creating a sequence as well as modifying it to another one are computed and the gain values are calculated using these costs. Although the models are used to score the quality of mapping, the numbers of matches should be taken into account. However, on the other hand in this case models are trained over relatively short sequences, therefore some of the match cases may not be observed during the training. This will result in receiving a large cost value for visiting them even if they are what one expects to see and cause an error in detecting the best vertex for some part of a read. There is a higher chance for this issue to happen if the length of the vertices are as short as a few nucleotide bases or in an extreme case a single base. To solve this issue, the formula to compute the cost of modifying sequences to one another is altered to take the number of matches into account. In fact, each modification cost of an alignment is divided by $T * M$ where T is an adjustable positive integer and M is the number of matches which have been seen on that alignment. The gain values are then calculated as in formula 2.1 and alignments with the positive average gain are kept for the next step.

3.2.3 Mapping the reads

To map reads against the graph, the best path on the graph that matches each read should be found. The following work-flow is repeated on each read after training the models. The next steps can be done in parallel for several reads at the same time.
For each read, a list of all the alignments between the read and the reference graph is made and they are ordered by their position on that read. Afterwards, they are grouped together as long as each alignment is overlapped with its neighbouring alignment on the read. Therefore, there will be overlap only within a group of alignments but not between them. These non-overlapped groups are then ordered by their position on the read and are checked for their distance to their neighbouring groups. If the distance between them on the read is less than a set maximum distance (Max_{dist}), they will be collapsed into a single group. The rationale behind the Max_{dist} is in fact to not get lost on the reference graph since this number is used in the next step to choose a sub-graph close to an alignment to look for the paths on it. At the end, a single best path will be found for each of these groups of alignments. The following steps will be repeated for each of the above groups.

Making an alignment graph

Here an alignment graph $G_{al}(V, E)$ is built where V is a set of vertices and E is a set of edges. Every $v \in V$ is an alignment index and every $e \in E$ represents the order of two adjacent vertices. Occurring alignments are ordered by their starting position on the read. The algorithm starts from the first alignment in each group, meaning the one which started before all the other alignments on the read. It then proceeds alignment by alignment until the last alignment of the group is visited. For each alignment, all the alignments which have their starting position on the read after it, but have no overlap with it on the read and are not further than the Max_{dist} to it, are saved as the set of successor alignments for the current one. If the successors set is not empty, a sub-graph of the reference graph including the current alignment is generated.
Detecting the sub-graph is done by using the Breadth-first search algorithm [36]. Searching on the reference graph is started from the current vertex as the source node and it continues until the distance between the visiting node and the source node exceeds the Max_{dist}. This part of the graph is chosen as the desired sub-graph. All the found successive alignments are checked for containing a part of a vertex from this sub-graph. Figure 3.2 (A) illustrates the above procedure. If there are such successor pairwise alignments, the shortest path on the reference graph to reach each of them is found. For this purpose, the Dijkstra algorithm with Fibonacci heaps [35], as priority queues, is used to find the best connection between the current alignment and each of its neighbours. To generate a connected path between two alignments, three cases are taken into account. Firstly, if the neighbouring alignments do not share a vertex with the current one (figure 3.3, *Case I*). Secondly, if they share the vertex on the forward direction (figure 3.3, *Case II*) and finally if they share one on the reverse direction (figure 3.3, *Case III*). The remaining sequences between two alignments both on the reference graph

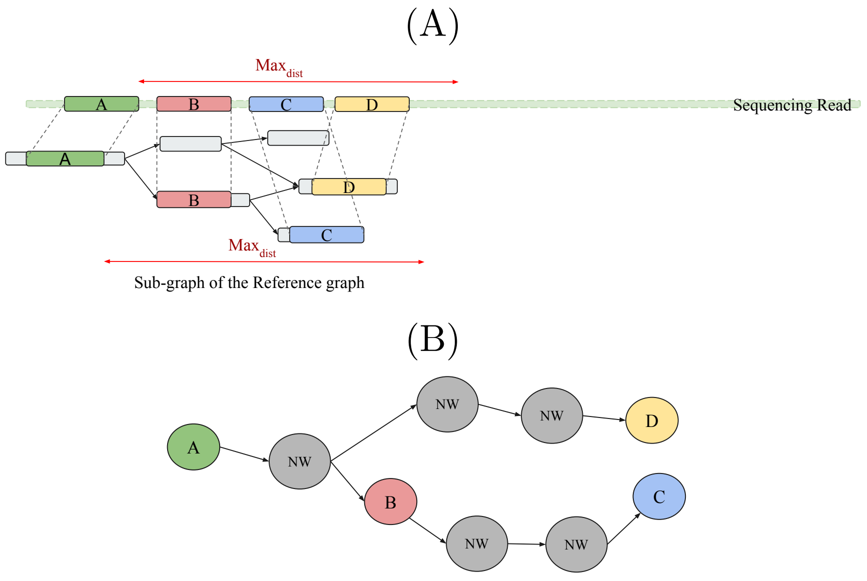


Figure 3.2: (A) Successor alignments and the built sub-graph for the pairwise alignment A, (B) Generated alignment graph after finding the best path to each of A successors. Alignments which were indexed by 'NW' were built using the Needleman-Wunsch algorithm to fill in the gap between existing pairwise alignments

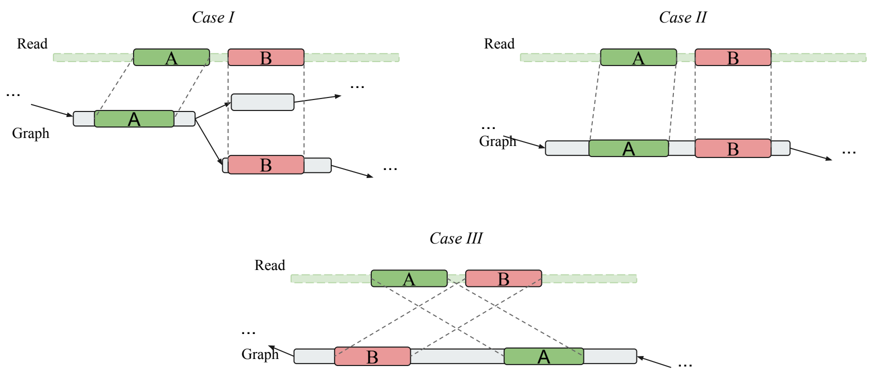


Figure 3.3: Case I: The successor alignment (B) is not sharing a graph vertex with the current alignment (A).Case II: Two alignments share a vertex on a graph on the forward direction. Case III: They are sharing a vertex but on the reverse direction.

and on the reads are detected and, using the Needleman-Wunsch algorithm, corresponding alignments for these regions are generated. If two neighbouring alignments share a vertex there is no need to look for the shortest path since there is only a single existing path between them. Thus, there is no need to use the Dijkstra algorithm and using the Needleman-Wunsch algorithm generates a single pairwise alignment to fill the gap in between the two existing alignments. However, in cases where a vertex is not shared more alignments may be produced where each contains one vertex as a reference. The Needleman-Wunsch algorithm is run considering the scores obtained from the trained Markov chain models to capture the best possible pairwise alignment for the non-aligned region between two pairwise alignments. The modification cost of changing the reference vertex to its corresponding sequence segment on the read, obtained from the Markov chain models, is used as the weight of the edge while looking for the best path between alignments. By finding all such paths for all the neighbouring alignments of the current pairwise alignment, graph G_{al} is expanded. The located alignments on these paths are chosen as vertices on this graph and their order on each path is shown by edges. Figure 3.2 (B) illustrates the generated alignment graph for the given sub-graph in figure 3.2 (A). The built G_{al} is later used to identify the best match between the reference graph and the read, thus it is necessary to define two virtual *Start* and *End* vertices on it. After checking all the successor alignments and expanding the alignment graph G_{al}, the position of the current alignment on the read is checked. If it is close enough to the beginning of the read or the beginning of the group of alignments it belongs to, it will be connected to the defined *Start* node. In fact, it should be closer than Max_{dist} to the beginning of the read or closer than Max_{dist} to be beginning of the group of alignments it belongs to, minus half of the Max_{dist}, to avoid the overlap between two subsequent groups of alignments. In other words, the alignment in the later case should be closer than Max_{dist} to

the $M = Pos_{first\ al} - Max_{dist}/2$, where $Pos_{first\ al}$ is the position of the first alignment of this group of alignments. To make such a connection, the read content between position 0 or M up to the current alignment position is aligned to a part of the reference graph prior to the current vertex. To find the part of the reference genome that this sequence is needed to be aligned to, the Breadth-first search algorithm is used to detect the predecessor vertices which are closer than Max_{dist} to the current vertex. Paths on this sub-graph are aligned to the chosen piece sequence of the read. These alignments are added as vertices to G_{al} along with the corresponding edges which represents their order. The *Start* vertex is connected to the first such alignment on each path.
The position of the current alignment is not only checked for its eligibility of being connected to the *Start* vertex but is also checked for being eligible to be connected to the *End* vertex. For this purpose, the last position of the alignment on the read is checked for being closer than Max_{dist} to the end of the read or being closer than Max_{dist} to the $N = Pos_{last\ al} + Max_{dist}$. The sequence content of this part of the read is then aligned to a part of the reference graph which is just after the current vertex. The same sub-graph as the one which has been used for creating alignments between the current alignment and its neighbours is also used here to generate pairwise alignments between each path on it and the chosen part of the read. The generated alignments are added as vertices to the G_{al} and their order of occurrence on the paths is represented by the edges that connect them. The last such alignment on each path is connected to the *End* vertex. The above procedure is repeated for all the alignments in a group and the graph G_{al} will be expanded. After visiting the last alignment of the group, G_{al} will no longer grow. Therefore, the best match for this part of the read is detected in the form of a path on the G_{al} graph.

Finding the shortest path on the alignment graph

The best mapping result for each read is in fact a collection of paths where each of them is a path with the lowest weight on one of the built G_{al} graphs on the read. To find these paths, the Dijkstra algorithm is applied on each of the built alignment graphs. The weight of each edge is the cost of modifying the vertex to the read of the alignment that the edge enters to. This cost is computed as one of the modification costs of the alignment employing the parameters of the trained Markov chain model. Walking on a G_{al} graph starts from the *Start* vertex and will end as soon as the vertex *End* is reached. To detect a single path, in case of having a graph with more than one component, expensive edges with the weights higher than the highest available cost are generated to make a connection between the separated components of this graph. The result will be a graph with only a single component. For each vertex on the G_{al} graph, expensive edges are connecting a vertex to its non-adjacent, non-overlapped vertices which occurred after the current vertex, both on the read and on the reference graph. To assign a high weight to each of these edges, the number of bases between two alignments on the read is calculated and a high weight is computed as follows:

$$Weight = (2.5 * N_{bases} + (Max_{dist}/2 * 2.5) + C \tag{3.1}$$

where N_{bases} is the number of bases between the two non-overlapped alignments and C is a constant value. In my experiments C has always been set to 1000 to guarantee that $Weight$ is costlier than any existing edge. The obtained weight from the formula 3.1 is then summed up with the modification cost of the entering alignment (the alignment which an edge enters to).
The chosen path with the lowest weight represents the mapping result for the part of a read which is covered by one of the alignments' groups. On a read with more than one such group of alignments, more than one of these paths are generated. Each of these paths covers an independent region on the long sequencing

read.

3.2.4 Output

Each obtained 'shortest path' includes several pairwise alignments between a read and several vertices on the reference graph. These pairwise alignments are then written on a MAF file ordered by their position of occurrence on a sequencing read.

3.3 Result

To check the accuracy of the proposed pipeline for mapping reads, the designed algorithm has been tested with simulated perfect reads. For the first experiment a reference graph was generated over 5 *E.coli* genomes without considering the $long-centres$. One of the input genomes has been chosen to simulate sequencing reads. 155 perfect long reads with the equal length of $30,030$ bases have been simulated, the simulation has been done by selecting regions of $30,030$ bases from a genome. LAST [25] was used to detect the pairwise alignments between the reads and the vertices on the graph. $23,559$ alignments were initially created. However, after running the Markov chain models only those with a high gain were kept as 'hits'. The mapping code was run to find the best match for each of these reads. Since the positions were known, the result was compared to ground truth, defined as the number of bases mapped to their precise origin. Running the mapping program took 11 minutes and 93.4% of the bases were mapped to the correct position on the reference graph while the rest were mapped elsewhere. Almost all these mismapped bases were located at the beginning or at the end of each read.
The next experiment was designed similarly over a different set of data. In this case perfect reads were simulated with various lengths between 102 and $15,000$ bases. Reads were all generated from one the reference genomes which were used in making the

graph. The graph was built over 4 different *S.cerevisiae* genomes. The number of simulated reads in this case was 2, 586. As in the previous experiment, LAST was applied to detect the pairwise alignments and it resulted in generating 54, 310 alignments. Running the mapping program over all of the reads, including the time needed for training the models, took 7 hours. Observing the result, the error rate was calculated. As has been mentioned for the the previous test run, all the bases were checked to ensure that they were mapped to where they belonged to. In this experiment only 3.94% of the bases mapped wrongly and the other 96.06% were mapped correctly.
To examine the ability of the pipeline in using the graphs generated by the other programs as a reference for mapping reads, a graph was built over 3 assemblies of *E.coli* using the REVEAL [29] tool. This tool generates a graph by looking for the maximum exact matches between input genomes. Thus, there are vertices with a very short length, sometimes as short as a single nucleotide. The complexity of the graph is much higher, the number of the paths to traverse is much bigger but the graph has no cycle and the output is always an acyclic graph. The same simulated perfect reads from the first experiment (155 reads with the length equal to 30, 030 bases) were used here and were mapped against the REVEAL [29] graph. Again LAST [25] was used to help in detecting the hits. However, due to the larger number of vertices the number of detected pairwise alignments is almost doubled. 42, 863 alignments were initially detected which were later filtered after running the models if they had a low average gain. The output was studied for the number of wrongly mapped bases. The error rate was much lower than the previous cases and only 0.014% of bases were mapped wrongly. However, the running time was much higher for this test and on average 5 minutes, including the training time, were needed on the same machine to map each read against the graph.
The pipeline was also tested by mapping a larger set of reads on the graph created by the REVEAL [29] tool. The chromosome

1 of two *A.thaliana* accessions were used as input to generate the graph and perfect reads were simulated from one of them. The number of reads was 451 and they all had a length equal to $15,000$ bases. Again, a high accuracy was obtained and only 0.15% of bases were mapped wrongly.
After running the algorithm several times on perfect reads, it was tested for handling error prone reads. For this purpose, error prone reads were simulated using the PBSIM [44] tool. The tool simulates SMRT sequencing reads with various lengths. 375 reads with the total number of $1,074,816$ bases has been generated from an *E.coli* assembly. The graph contains 3 assemblies of *E.coli*, one of which is the one that sequencing reads were generated from, that has been made by the REVEAL tool. It has then been used as the reference to map the simulated sequencing reads on it. Similar to the previous experiments, pairwise alignments between these reads and the graph were discovered using the LAST program. Running time on the simulated error prone reads is faster than the perfect reads due to the significant smaller number of hits. Only 4.4% of the bases were mapped on a wrong vertex or a wrong base and more than 95% of the bases were mapped correctly to where they belong to on the original genome.

3.4 Discussion

The above experiments revealed that there is a trade off between the running time and the mapping accuracy by choosing either the graph proposed in the previous chapter of this dissertation or the one which was generated by the REVEAL program. The graph built by REVEAL branches into two vertices whenever a single nucleotide mismatch occurs between two genomes as opposed to my proposed approach where the similar sequences were kept as a cluster and only one of them is chosen as a vertex on the graph which results in a more simplified graph with longer

vertices. The shorter running time obtained by running the mapping pipeline over our proposed graph is a result of the simplified structure of the input graph, however, it resulted in a slight drop in accuracy (see section 3.3). Some improvements that would be possible to apply in order to reduce the running time in the case of complex graphs would entail changing the way 'hits' are found and so try to reduce the number of them by filtering for fewer 'hits' with a much higher probability of being true matches. One could also generate longer contigs rather than using each read by looking for the overlap between them and then use these longer pieces instead of every single read. This is possible since the program has no limitation in accepting the long reads.
Last but not least, the time needed to train the model can be saved by training the model only once over data that comes from the sequencing machine and by using the same model parameters as long as the same machine is used to generate the sequencing reads.

Chapter 4

Conclusion

During the last two decades, the amount of generated genome data has increased exponentially. It has resulted in a large collection of assembled genomes with a high magnitude of diversity within a species and presented scientists with the challenge of finding a reliable way of studying the variations between them. The classical approach to solving this problem has been to employ a single reference genome. However, using a single genome as a reference imposes a bias since every base on any other sequence is compared with only a single base on the reference. Therefore, it does not provide a robust way of capturing SNPs or structural variations in the presence of a high level of diversity. On the other hand, many genomes have already been completely sequenced, thus, shifting from a single reference genome to a multi-genome reference will return a more robust way of representing the hidden structure of the data and its variants, as well as removing the bias against using only a single reference genome.

As has been mentioned in chapter 1, several studies have already been carried out which try to employ the information obtained from more than a single genome on a reference, however applying several full length genomes in making such a data structure is still a challenge. To address this challenge, in this dissertation an algorithm was proposed to generate a data structure that represents several full length genomes. Similar to a variety of other methods (see 1.3.2), here a genome graph has been introduced. As opposed to previous methods, it contains the information obtained from all the genomes rather than a reference genome supplemented by a collection of variations. The method provides individual specific Markov chain models which allows for analysing genomes with very different magnitudes of difference together and aids in capturing the hidden structure of each input genome. An advantage of the proposed algorithm is that it searches for the similar segments of sequences and clusters them together. Clustering helps in studying repeated regions and saving memory storage by keeping less information. Moreover, it enables us to perform both between and within cluster analysis. Therefore, the algorithm can be easily applied in global genome analyses.

To cluster the similar regions, different groups of independent sequences are generated. Sequences of each group are related to each other in a sense that each two of them belong to a pairwise alignment and they are independent from the sequences in other groups in a sense that there is no overlap between their sequences. Producing the independent groups in a way that has been done in this study has a high cost and significantly increases the running time of the program for large genomes. This might be improved by filtering the input alignments or modifying the algorithm for cutting the partially overlapped alignments. In spite of the high cost of the algorithm in handling large genomes, we believe that this will be a great leap forward in generating such references in the near future.

The introduced algorithm in this research uses the information obtained from the graph and the models' parameters to compress the input data. Observing a high compression rate after arithmetically encoding the input genomes proves the power of applied models in capturing the structure of input genomes. The

proposed compression tool can be used as a metric to measure the common information of individual genomes and shows their distances.
Finally, since the decisions are always made in a manner that minimizes Shannon information, the presented approach is a non-parametric approach and as such does not rely on any arbitrary parameter choices.
In summary, as opposed to the previous algorithms, this algorithm presents a way to analyse multiple full length genomes by minimizing the Shannon information. Moreover, by employing individual models, it helps in perfectly capturing the information of their contents, and last but not least, clusters similar segments of several genomes which aids in studying homologous regions and simplifying downstream analysis. Despite the above advantages, the algorithm has a high complexity in generating independent pieces of sequence which can be altered to reduce the time and the complexity of the algorithm in handling a higher number genomes or larger genomes.
Developing such an algorithm assists us in finding matches for longer reads on a graph as a reference. Finding the best match for each sequencing read on the graph rather than a single reference will give us the opportunity of detecting more variations and studying the genomes which are closer to a combination of several genomes. To search for the best match between a graph as a reference and a sequencing read, another algorithm has been introduced in this dissertation. By traversing the graph the algorithm finds the path that fits each read the best. First, highly similar regions between vertices of the graph and the reads (hits) are found, then the paths between each two such regions are checked and the best one is chosen. To look for the best path, several Markov chain models were trained prior to traversing the graph. The models specify a cost for each modification between a vertex and a read, these costs are used as weights for detecting the best path. The algorithm returns a high accuracy in mapping reads, however, the size of the edges of a graph may affect

its running time. To deal with that issue, generating longer contigs from several sequencing reads and mapping them instead of each single read and reducing the number of hits by looking only for the best ones are recommended.

Appendices

Appendix A

Data

Data sets which were used to generate a *multi-genome reference* over 2, 4, 8 and 16 strains of *E.coli* can be found here:

1- NCBI reference sequence NC_000913.3
(`https://www.ncbi.nlm.nih.gov/nuccore/NC_000913.3?report=fasta`)
2- NCBI reference sequence: NC_002695.1
(`https://www.ncbi.nlm.nih.gov/nuccore/NC_002695.1?report=fasta`)
3- NCBI reference sequence: NC_010473.1
(`https://www.ncbi.nlm.nih.gov/nuccore/NC_010473.1?report=fasta`)
4- NCBI reference sequence: NC_012967.1
(`https://www.ncbi.nlm.nih.gov/nuccore/NC_012967.1?report=fasta`)
5- NCBI reference sequence: AE014075.1
(`https://www.ncbi.nlm.nih.gov/nuccore/AE014075.1?report=fasta`)
6- NCBI reference sequence: NC_004431.1
(`https://www.ncbi.nlm.nih.gov/nuccore/NC_004431.1?report=fasta`)
7- NCBI reference sequence: NC_007946.1
(`https://www.ncbi.nlm.nih.gov/nuccore/NC_007946.1?report=fasta`)
8- NCBI reference sequence: NC_008253.1
(`https://www.ncbi.nlm.nih.gov/nuccore/NC_008253.1?report=fasta`)
9- NCBI reference sequence: NC_011750.1
(`https://www.ncbi.nlm.nih.gov/nuccore/NC_011750.1?report=fasta`)
10- NCBI reference sequence: NC_011751.1
(`https://www.ncbi.nlm.nih.gov/nuccore/NC_011751.1?report=fasta`)

11- NCBI reference sequence: NC_018658.1
(https://www.ncbi.nlm.nih.gov/nuccore/NC_018658.1?report=fasta)
12- NCBI reference sequence: NC_009800.1
(https://www.ncbi.nlm.nih.gov/nuccore/NC_009800.1?report=fasta)
13- NCBI reference sequence: NC_017634.1
(https://www.ncbi.nlm.nih.gov/nuccore/NC_017634.1?report=fasta)
14- NCBI reference sequence: AE005174.2
(https://www.ncbi.nlm.nih.gov/nuccore/AE005174.2?report=fasta)
15- NCBI reference sequence: NC_008563.1
(https://www.ncbi.nlm.nih.gov/nuccore/NC_008563.1?report=fasta)
16- NCBI reference sequence: NC_017633.1
(https://www.ncbi.nlm.nih.gov/nuccore/NC_017633.1?report=fasta)

Data sets which were used to generate a *multi-genome reference* over 4 *S.cerevisiae* strains can be found here:
https://www.yeastgenome.org/strain/AWRI1631
https://www.yeastgenome.org/strain/FL100
https://www.yeastgenome.org/strain/CLIB215
https://www.ncbi.nlm.nih.gov/Traces/wgs/?val=AEWL01#contigs

A.thaliana Data:
Kubica, Bemm, Weigel, unpublished

Bibliography

[1] Multiple alignment format.
`https://genome.ucsc.edu/FAQ/FAQformat.html#format5`.

[2] A proposal of the graphical fragment assembly format, 2014.

[3] Srinivas Aluru. *Handbook of computational molecular biology.* CRC Press, 2005.

[4] Anton Bankevich, Sergey Nurk, Dmitry Antipov, Alexey A Gurevich, Mikhail Dvorkin, Alexander S Kulikov, Valery M Lesin, Sergey I Nikolenko, Son Pham, Andrey D Prjibelski, et al. Spades: a new genome assembly algorithm and its applications to single-cell sequencing. *Journal of computational biology*, 19(5):455–477, 2012.

[5] Nayantara Bhatnagar, Pietro Caputo, Prasad Tetali, Eric Vigoda, et al. Analysis of top-swap shuffling for genome rearrangements. *The Annals of Applied Probability*, 17(4):1424–1445, 2007.

[6] James K Bonfield, Kathryn F Smith, and Rodger Staden. A new dna sequence assembly program. *Nucleic acids research*, 23(24):4992–4999, 1995.

[7] Michael Burrows and David J Wheeler. A block-sorting lossless data compression algorithm. 1994.

[8] MO Dayhoff, RM Schwartz, and BC Orcutt. A model of evolutionary change in proteins, atlas of protein sequence and struc-

ture 1978, vol. 5, suppl. *Dayhoff, MO, National Biomedical Research Foundation, Washington DC*, 1978.

[9] FA de Bruijn. A combinatorial problem. 1946.

[10] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[11] Alexander Dilthey, Charles Cox, Zamin Iqbal, Matthew R Nelson, and Gil McVean. Improved genome inference in the mhc using a population reference graph. *Nature genetics*, 47(6):682–688, 2015.

[12] Richard Durbin. Efficient haplotype matching and storage using the positional burrows–wheeler transform (pbwt). *Bioinformatics*, 30(9):1266–1272, 2014.

[13] Richard Durbin, Sean R Eddy, Anders Krogh, and Graeme Mitchison. *Biological sequence analysis: probabilistic models of proteins and nucleic acids.* Cambridge university press, 1998.

[14] Martin Farach. Optimal suffix tree construction with large alphabets. In *Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on*, pages 137–143. IEEE, 1997.

[15] Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 390–398. IEEE, 2000.

[16] Michael L Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, 34(3):596–615, 1987.

[17] Brendan J Frey and Delbert Dueck. Clustering by passing messages between data points. *science*, 315(5814):972–976, 2007.

[18] Erik Garrison, Jouni Sirén, Adam M Novak, Glenn Hickey, Jordan M Eizenga, Eric T Dawson, William Jones, Shilpa Garg, Charles Markello, Michael F Lin, et al. Variation graph toolkit improves read mapping by representing genetic variation in the reference. *Nature biotechnology*, 2018.

[19] Sara Goodwin, John D McPherson, and W Richard McCombie. Coming of age: ten years of next-generation sequencing technologies. *Nature Reviews Genetics*, 17(6):333–351, 2016.

[20] Frank Harary and George E Uhlenbeck. On the number of husimi trees i. *Proceedings of the National Academy of Sciences*, 39(4):315–322, 1953.

[21] Steven Henikoff and Jorja G Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences*, 89(22):10915–10919, 1992.

[22] Mark Howison, Felipe Zapata, Erika J Edwards, and Casey W Dunn. Bayesian genome assembly and assessment by markov chain monte carlo sampling. *PloS one*, 9(6):e99497, 2014.

[23] Lin Huang, Victoria Popic, and Serafim Batzoglou. Short read alignment with populations of genomes. *Bioinformatics*, 29(13):i361–i370, 2013.

[24] Xiaoqiu Huang and Anup Madan. Cap3: A dna sequence assembly program. *Genome research*, 9(9):868–877, 1999.

[25] Szymon M Kiełbasa, Raymond Wan, Kengo Sato, Paul Horton, and Martin C Frith. Adaptive seeds tame genomic sequence comparison. *Genome research*, 21(3):487–493, 2011.

[26] Eric S Lander, Lauren M Linton, Bruce Birren, Chad Nusbaum, Michael C Zody, Jennifer Baldwin, Keri Devon, Ken Dewar, Michael Doyle, William FitzHugh, et al. Initial sequencing and analysis of the human genome. *Nature*, 409(6822):860–921, 2001.

[27] Heng Li and Richard Durbin. Fast and accurate short read alignment with burrows–wheeler transform. *Bioinformatics*, 25(14):1754–1760, 2009.

[28] Antoine Limasset, Bastien Cazaux, Eric Rivals, and Pierre Peterlongo. Read mapping on de bruijn graphs. *BMC bioinformatics*, 17(1):237, 2016.

[29] Jasper Linthorst, Marc Hulsman, Henne Holstege, and Marcel Reinders. Scalable multi whole-genome alignment using recursive exact matching. *bioRxiv*, 2015. URL: `https://github.com/jasperlinthorst/REVEAL`.

[30] David J Lipman and William R Pearson. Rapid and sensitive protein similarity searches. *Science*, 227(4693):1435–1441, 1985.

[31] Bo Liu, Hongzhe Guo, Michael Brudno, and Yadong Wang. debga: read alignment with de bruijn graph-based seed and extension. *Bioinformatics*, 32(21):3224–3232, 2016.

[32] Sorina Maciuca, Carlos del Ojo Elias, Gil McVean, and Zamin Iqbal. A natural encoding of genetic variation in a burrows-wheeler transform to enable mapping and genome inference. In *International Workshop on Algorithms in Bioinformatics*, pages 222–233. Springer, 2016.

[33] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967.

[34] Shoshana Marcus, Hayan Lee, and Michael C Schatz. Splitmem: a graphical algorithm for pan-genome analysis with suffix skips. *Bioinformatics*, 30(24):3476–3483, 2014.

[35] Robin Message. A simple c++ fibonacci heap implementation. `https://github.com/robinmessage/fibonacci.git`.

[36] Edward F Moore. The shortest path through a maze. In *Proc. Int. Symp. Switching Theory, 1959*, pages 285–292, 1959.

[37] Eugene W Myers. Toward simplifying and accurately formulating fragment assembly. *Journal of Computational Biology*, 2(2):275–290, 1995.

[38] Eugene W Myers. The fragment assembly string graph. *Bioinformatics*, 21(suppl_2):ii79–ii85, 2005.

[39] Saul B Needleman and Christian D Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453, 1970.

[40] Ngan Nguyen, Glenn Hickey, Daniel R Zerbino, Brian Raney, Dent Earl, Joel Armstrong, W James Kent, David Haussler, and Benedict Paten. Building a pan-genome reference for a population. *Journal of Computational Biology*, 22(5):387–401, 2015.

[41] Adam M Novak, Erik Garrison, and Benedict Paten. A graph extension of the positional burrows-wheeler transform and its applications. In *International Workshop on Algorithms in Bioinformatics*, pages 246–256. Springer, 2016.

[42] Adam M Novak, Glenn Hickey, Erik Garrison, Sean Blum, Abram Connelly, Alexander Dilthey, Jordan Eizenga, MA Saleh Elmohamed, Sally Guthrie, André Kahles, et al. Genome graphs. *bioRxiv*, page 101378, 2017.

[43] Adam M Novak, Yohei Rosen, David Haussler, and Benedict Paten. Canonical, stable, general mapping using context schemes. *Bioinformatics*, 31(22):3569–3576, 2015.

[44] Yukiteru Ono, Kiyoshi Asai, and Michiaki Hamada. Pbsim: Pacbio reads simulator—toward accurate genome assembly. *Bioinformatics*, 29(1):119–121, 2012.

[45] Taku Onodera, Kunihiko Sadakane, and Tetsuo Shibuya. Detecting superbubbles in assembly graphs. In *International Workshop on Algorithms in Bioinformatics*, pages 338–348. Springer, 2013.

[46] Benedict Paten, Mark Diekhans, Dent Earl, John St John, Jian Ma, Bernard Suh, and David Haussler. Cactus graphs for genome comparisons. *Journal of Computational Biology*, 18(3):469–481, 2011.

[47] Benedict Paten, Dent Earl, Ngan Nguyen, Mark Diekhans, Daniel Zerbino, and David Haussler. Cactus: Algorithms for genome multiple sequence alignment. *Genome research*, 21(9):1512–1528, 2011.

[48] Benedict Paten, Adam Novak, and David Haussler. Mapping to a reference genome structure. *arXiv preprint arXiv:1404.5010*, 2014.

[49] Benedict Paten, Adam M Novak, Erik Garrison, and Glenn Hickey. Superbubbles, ultrabubbles and cacti. In *International Conference on Research in Computational Molecular Biology*, pages 173–189. Springer, 2017.

[50] Paul A Pevzner, Haixu Tang, and Glenn Tesler. De novo repeat classification and fragment assembly. *Genome research*, 14(9):1786–1796, 2004.

[51] Pavel A Pevzner, Haixu Tang, and Michael S Waterman. An eulerian path approach to dna fragment assembly. *Proceedings of the National Academy of Sciences*, 98(17):9748–9753, 2001.

[52] Armando J Pinho and Diogo Pratas. Mfcompress: a compression tool for fasta and multi-fasta data. *Bioinformatics*, 30(1):117–118, 2013.

[53] Jens Quedenfeld and Sven Rahmann. Variant tolerant read mapping using min-hashing. *arXiv preprint arXiv:1702.01703*, 2017.

[54] Benjamin Raphael, Degui Zhi, Haixu Tang, and Pavel Pevzner. A novel method for multiple alignment of sequences with repeated and shuffled elements. *Genome Research*, 14(11):2336–2346, 2004.

[55] Frederick Sanger, Steven Nicklen, and Alan R Coulson. Dna sequencing with chain-terminating inhibitors. *Proceedings of the national academy of sciences*, 74(12):5463–5467, 1977.

[56] Korbinian Schneeberger, Jörg Hagmann, Stephan Ossowski, Norman Warthmann, Sandra Gesing, Oliver Kohlbacher, and Detlef Weigel. Simultaneous alignment of short reads against multiple genomes. *Genome biology*, 10(9):R98, 2009.

[57] Julian Seward. bzip2 high-quality data compressor. `http://www.bzip.org`.

[58] Claude E Shannon and Warren Weaver. The mathematical theory of communication. 1948.

[59] Michele Simionato. An introduction to graphviz and dot. 2004. URL: `http://www.linuxdevcenter.com/pub/a/linux/2004/05/06/graphvizdot.html`.

[60] Jared T Simpson and Richard Durbin. Efficient construction of an assembly string graph using the fm-index. *Bioinformatics*, 26(12):i367–i373, 2010.

[61] Aaron D Skewes and Roy D Welch. A markovian analysis of bacterial genome sequence constraints. *PeerJ*, 1:e127, 2013.

[62] Temple F Smith and Michael S Waterman. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, 1981.

[63] Kavya Vaddadi, Naveen Sivadasan, Kshitij Tayal, and Rajgopal Srinivasan. Sequence alignment on directed graphs. *bioRxiv*, page 124941, 2017.

[64] Peter Weiner. Linear pattern matching algorithms. In *Switching and Automata Theory, 1973. SWAT'08. IEEE Conference Record of 14th Annual Symposium on*, pages 1–11. IEEE, 1973.

[65] KA Wetterstrand. Data from the genome sequencing program of the national human genome research institute. `https://www.genome.gov/sequencingcostsdata`.

[66] Ryan Wick. Effect of kmer size. `https://github.com/rrwick/Bandage/wiki/Effect-of-kmer-size`.

[67] Byung-Jun Yoon. Hidden markov models and their applications in biological sequence analysis. *Current genomics*, 10(6):402–415, 2009.

[68] Daniel R Zerbino and Ewan Birney. Velvet: algorithms for de novo short read assembly using de bruijn graphs. *Genome research*, 18(5):821–829, 2008.

[69] Yu Zhang and Michael S Waterman. An eulerian path approach to local multiple alignment for dna sequences. *Proceedings of the National Academy of Sciences of the United States of America*, 102(5):1285–1290, 2005.